ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ 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

Разработка предикатов нарушения симметрии для построения детерминированных конечных автоматов при помощи сведения к задаче о выполнимости

Автор квалификационной работы *Закирзянов И.Т.* (подпись)

Руководитель *Ульянцев В.И.* (подпись)

Консультанты:

а) По экономике и организации производства _____ (подпись)

б) По безопасности жизнедеятельности и экологии _____ (подпись)

в) _____ (подпись)

К защите допустить

Зав. кафедрой *Васильев В.Н.* (подпись)

«___» _____ 2015 г.

Санкт-Петербург, 2015 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Детерминированные конечные автоматы (ДКА) – это модели, которые распознают регулярные языки [1]. Задача построения ДКА одна из наиболее изученных [2] в области грамматического вывода. С помощью решения поставленной задачи может быть решено множество востребованных задач. Построение ДКА так или иначе встречается во многих задачах синтаксического распознавания, вычислительной биологии и обработки естественного языка [3].

Для построения автоматов в задачах данного типа успешно применяется подход, основанный на сведении к задаче о выполнимости [4]. Для сужения пространства поиска решений применяются предикаты нарушения симметрии, однако существующие предикаты нарушения симметрии недостаточно эффективны. Недостатком существующих методов является то, что они неприменимы в случае зашумленных данных, а также то, что они не являются достаточно гибкими для решения более широкого класса задач.

Целью настоящей работы является разработка предикатов нарушения симметрии, которые позволят применять метод сведения к задаче о выполнимости для решения определенного класса задач, связанных с построением ДКА по обучающим словарям.

Перед началом исследования были сформулированы следующие задачи.

- Модифицировать существующее сведение к задаче о выполнимости для решения задачи с зашумленными данными.
- Разработать предикаты нарушения симметрии на основе алгоритмов обхода графов.
- Адаптировать предлагаемые предикаты для решения задачи с зашумленными данными.
- Сравнить предлагаемый подход с существующими решениями как для задачи с точными данными, так и с зашумленными.

В ходе работы над исследованием была подготовлена и опубликована статья в материалах к международной конференции LATA 2015 (9th Internation-

al Conference on Language and Automata Theory and Applications) [5]. Также автор выступил на «IV Всероссийском Конгрессе молодых ученных» с докладом по теме исследования и был отмечен дипломом «за лучший научно-исследовательский доклад студента».

ГЛАВА 1. ЗАДАЧА ПОСТРОЕНИЯ ДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ И МЕТОДЫ ЕЕ РЕШЕНИЯ

В данной главе приведена постановка задачи построения ДКА, проведен обзор существующих методов решения. Также описан подход, основанный на сведении к задаче о выполнимости, который лежит в основе данного исследования.

1.1. Задача построения ДКА

Задача построения детерминированных конечных автоматов заключается в нахождении ДКА с минимальным количеством состояний, удовлетворяющего заданному обучающему словарю: набору строк с метками принадлежности некоторому языку. Искомый ДКА не должен допускать слова, помеченные отрицательной меткой, и должен допускать помеченные положительной меткой. В [6] было показано, что нахождение ДКА с заданным ограничением на размер (количество состояний) сверху является NP-полной задачей. Кроме того в [7] было показано, что она не может быть решена приближенно за полиномиальное время.

1.2. Методы решения задачи построения ДКА

Несмотря на теоретическую сложность, существует несколько довольно эффективных алгоритмов построения ДКА [2]. Наиболее распространенным подходом являются решения, основанные на алгоритме слияния состояний (evidence driven state-merging, EDSM) [8]. Ключевая идея алгоритмов данного семейства – построение префиксного автомата (augmented prefix tree acceptor, APTA) по исходному словарю и последовательное слияние состояний в нем до тех пор, пока не останется возможных слияний. Иными словами, EDSM – это полиномиальный жадный алгоритм, который довольно эффективно ищет локальный оптимум. Впервые EDSM был представлен на соревновании *Abbadingo DFA learning* [8] и стал победителем. Для улучшения алгоритма слияния состояний было предложено несколько специализированных процедур поиска

(например, [9,10]). Один из наиболее успешных подходов – это EDSM алгоритм с красно-синей структурой (red-blue framework) [8], также известный как алгоритм синего фронта (Blue-fringe algorithm).

Второй подход к решению задачи о построении ДКА основан на эволюционных вычислениях. Ранние работы включают в себя [11,12]. Позже авторы [13] представили эффективную схему построения ДКА с помощью метода спуска со случайными мутациями (multi-start random hill climber), который использовался, чтобы оптимизировать матрицу переходов строящегося ДКА. Авторы обнаружили, что предложенный эволюционный алгоритм (EA) превосходит EDSM на небольших целевых ДКА, когда обучающий набор строк разрежен. Для больших ДКА с 32 состояниями EA полностью проигрывает EDSM.

Задачей соревнования *GECCO 2004 Noisy DFA competition* [14] было построение целевого ДКА по зашумленным данным – 10 процентов меток обучающих строк были изменены на противоположные. В [15] Лукас и Рейнольдс показали, что за ограниченное время эволюционный алгоритм способен в ряде случаев построить искомый ДКА даже с таким высоким уровнем шума. Авторы сравнили данный алгоритм с результатами соревнования *GECCO* и обнаружили, что полностью выигрывают у всех участников. Таким образом, этот алгоритм – лучший из известных на данный момент алгоритмов для построения ДКА по обучающему словарю с шумом.

Следует заметить, что при своей эффективности рассмотренные выше подходы являются приближенными и не всегда находят минимальные решения.

В ряде случаев лучшим решением задачи построения ДКА по точному словарю является решение сведение к задаче SAT [4], которое было представлено на соревновании *the StaMInA* (State Machine Inference Approaches) [16] и стало безоговорочным победителем. Главная идея данного подхода – сведение задачи построения ДКА к задаче выполнимости булевой формулы (Boolean satisfiability problem, SAT). Таким образом, можно использовать высокооптимизированные современные методы решения задачи SAT, основанные на алгоритме Дэвиса-Патнема-Логемана-Лавленда (DPLL algorithm) [17]. Программ-

ные средства для решения задачи о выполнимости активно развиваются (для краткости в дальнейшем будем называть такое средство SAT-solver). Единожды придумав сведение и реализовав его, можно использовать все более новые и совершенные программные решения. Это возможно, так как для всех программных средств существует стандарт входных данных. Булева формула должна быть представлена в конъюнктивной нормальной форме (КНФ) и записана в формате DIMACS [18]. Основным достоинством предлагаемого подхода является то, что он в отличие от эволюционных алгоритмов точен, то есть позволяет точно определять существование искомого ДКА по зашумленным данным.

Следует отметить, что сведение к задаче SAT также эффективно применяется для решения таких задач, как ограниченная проверка моделей (bounded model checking) [19], решение SQL ограничений с помощью инкрементального сведения [20], анализ JML-аннотированных последовательных программ [21], построение управляющих конечных автоматов [22].

1.3. Предикаты нарушения симметрии

Решения многих задач оптимизации содержат в себе симметрию – группу одинаковых по критерию оптимизации решений, которые могут быть получены друг из друга путем несложных преобразований. Чтобы ускорить процесс поиска решения, можно уменьшить пространство поиска с помощью применения *предикатов нарушения симметрии*. В задаче построения ДКА самой очевидной симметрией является группа изоморфных автоматов. Если известно, что некоторый автомат не является решением поставленной задачи, то хочется избежать рассмотрения и проверки всех изоморфных ему автоматов, количество которых равно $n!$, где n – размер автомата.

В [4] нарушение симметрии было реализовано с помощью фиксирования цветов вершин в некоторой клике префиксного автомата, полученной с помощью жадного *алгоритма поиска максимальной клики* (max-clique algorithm) на предварительном шаге сведения к задаче о выполнимости. В данной же работе предлагаются новые предикаты нарушения симметрии, которые могут быть до-

бавлены в булеву формулу, соответствующую проблеме построения ДКА с помощью сведения к задаче SAT. Предлагаемые предикаты фиксируют нумерацию состояний ДКА в порядке обходов в глубину (depth-first search, DFS) или в ширину (breadth-first search, BFS). Таким образом, для каждого класса эквивалентности будет рассмотрен только один представитель, пронумерованный, соответственной, в порядке BFS или в порядке DFS обходов. Предлагаемые предикаты не могут быть применены совместно с алгоритмом поиска максимальной клики, но этот подход более гибкий, а в некоторых случаях и более эффективный.

1.4. Описание сведения задачи построения ДКА к задаче о выполнимости

В данном разделе подробно описан подход DFASAT, основанный на сведении задачи построения ДКА к задаче о выполнимости булевой формулы, предложенный в [4]. Данный подход лежит в основе разработанного в ходе данного исследования решения. Для того чтобы значительно уменьшить пространство поиска решения задачи SAT, авторы применяют несколько шагов алгоритма EDSM перед сведением. Но так как EDSM не гарантирует минимальность решения, то в данной работе этот шаг опускается.

1.4.1. Формальная постановка задачи

Цель задачи построения ДКА по заданному словарю – найти ДКА A минимального размера такой, что все строки из S_+ – множества строк с положительной меткой – допускаются автоматом A, а все строки из S_- – множества строк с отрицательной меткой – не допускаются. Размер автомата A определяется количеством состояний C, из которых он состоит. Алфавит $\sum = \{l_1, \ldots, l_L\}$ искомого автомата A состоит из всех символов из S_+ и S_-. Пример минимального ДКА для $S_+ = \{ab, b, ba, bbb\}$ и $S_- = \{abbb, baba\}$ показан на рис. 1. В данной работе, без уменьшения общности, предполагается, что состояния ДКА пронумерованы от 1 до C, и начальное состояние имеет номер 1.

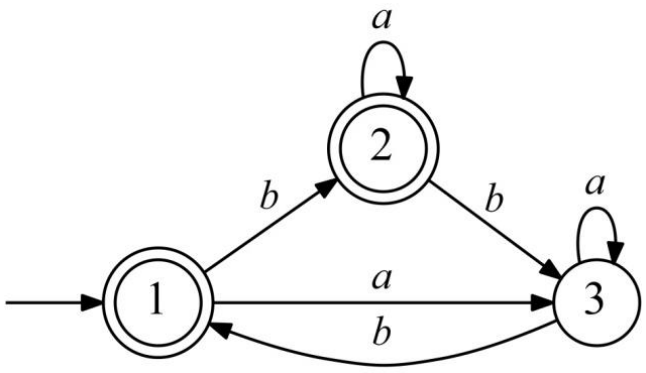

Рис. 1. Пример ДКА

В [4] авторы представили компактное сведение (compact translation) задачи построения ДКА к задаче SAT. Опишем эту технику, поскольку она лежит в основе предлагаемого подхода, и разработанные предикаты нарушения симметрии дополняют ее.

1.4.2. Построение префиксного автомата

Первый шаг метода слияния состояний и метода сведения к SAT – построение префиксного автомата (augmented prefix tree acceptor, APTA), сконструированного по множествам S_+ и S_-. APTA – это древовидный автомат, в котором пути, соответствующие двум строкам, достигают одинакового состояния v тогда и только тогда, когда данные строки имеют одинаковый префикс, заканчивающийся символом, соответствующим вершине v. Обозначим множество всех состояний префиксного автомата как V; корень данного автомата как v_r; множество принимающих состояний как V_+; множество отвергающих состояний как V_-. Также, для состояния v (кроме v_r) обозначим предка как $p(v)$, а символ, по которому есть переход из предка – как $l(v)$. Префиксный автомат для вышеупомянутых $S_+ = \{ab, b, ba, bbb\}$ и $S_- = \{abbb, baba\}$ изображен на рис. 2.

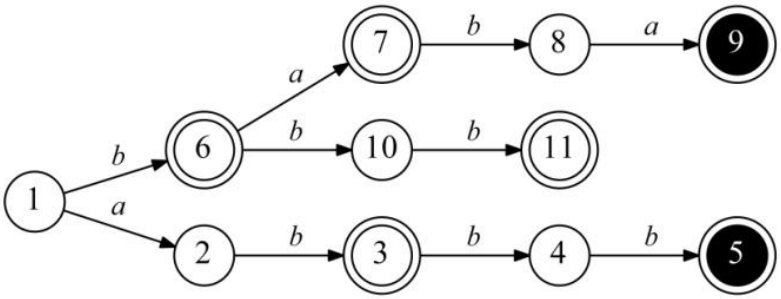

Рис. 2 Пример префиксного автомата

1.4.3. Построение графа совместимости

Вторым шагом техники, предложенной в [4], является построение *графа совместимости* (consistency graph, CG) вершин построенного префиксного автомата. Множество вершин графа совместимости совпадает с множеством состояний префиксного автомата. Две вершины *v* и *w* в графе совместимости соединены ребром (и называются несовместными), если слияние *v* и *w* в префиксном автомате приведет к несовместности: принимающее состояние будет объединено с отвергающим. Обозначим множество ребер графа совместимости как *E*. Граф совместимости для префиксного автомата, изображенного на рис. 2, показан на рис. 3.

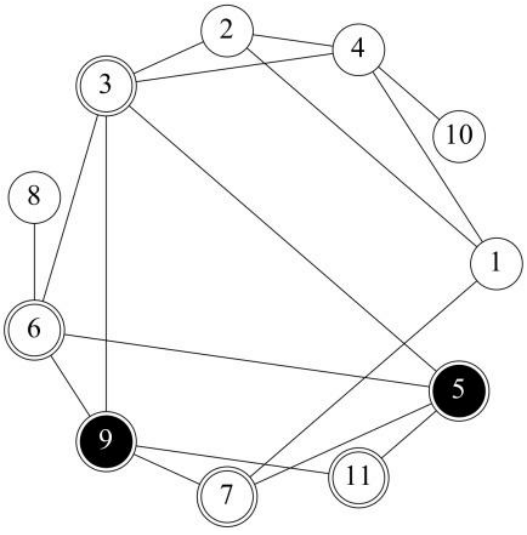

Рис. 3 Пример графа совместимости

1.4.4. Построение булевой формулы

Ключевая часть алгоритма – кодирование задачи построения ДКА в булевой формуле в конъюнктивной нормальной форме (КНФ) и использование известных программных средств для решения задачи SAT и нахождения выполняющей подстановки. Для заданного словаря и фиксированного размера C искомого ДКА SAT-solver находит выполняющую подстановку, которая однозначно определяет ДКА размера C, соответствующий заданному словарю, или сигнализирует, что такой подстановки не существует.

Главной идеей рассматриваемого сведения является использование различных цветов для каждого состояния искомого ДКА и нахождение соответствующего отображения состояний префиксного автомата в эти цвета. Иначе говоря, необходимо найти такую раскраску префиксного автомата, что слияние состояний одинакового цвета в одно не приведет к несовместности. Для удобства будем обозначать цвета числами от 1 до C, которые и будут задавать нумерацию искомого ДКА.

Три типа переменных используются в предложенном компактном сведении:

1. Переменные *цвета* $x_{v,i} \equiv 1 (v \in V; 1 \le i \le C)$ – состояние v префиксного автомата имеет цвет i;
2. Переменные *переходов* $y_{l,i,j} \equiv 1 (l \in \Sigma; 1 \le i, j \le C)$ – переход искомого автомата из вершины i по символу l ведет в вершину j;
3. Переменные *допуска* $z_i \equiv 1 (1 \le i \le C)$ – состояние i искомого автомата является допускающим.

Прямое кодирование, описанное в [4], использует только переменные $x_{v,i}$; переменные $y_{l,i,j}$ и z_i – вспомогательные и используются только в компактном сведении, которое описано ниже.

Компактное сведение использует девять типов условий (в скобках, там, где необходимо, указаны соответствующие дизъюнкты, которые должны быть добавлены в КНФ-формулу):

1. $x_{v,i} \Rightarrow z_i \ (v \in V_+; 1 \le i \le C)$ – определяет переменные допуска z_i для допускающих состояний $(\neg x_{v,i} \vee z_i)$;
2. $x_{v,i} \Rightarrow \neg z_i \ (v \in V_-; 1 \le i \le C)$ – определяет переменные допуска z_i для отвергающих состояний $(\neg x_{v,i} \vee \neg z_i)$;
3. $x_{v,1} \vee x_{v,2} \vee \ldots \vee x_{v,C} \ (v \in V)$ – каждому состоянию префиксного автомата *v* соответствует хотя бы один цвет;
4. $x_{p(v),i} \wedge x_{v,j} \Rightarrow y_{l(v),i,j} \ (v \in V \setminus \{v_r\}; 1 \le i, j \le C)$ – переход по символу $l(v)$ в ДКА определен, если определены цвета состояния *v* и его родителя $p(v)$ $(y_{l(v),i,j} \vee \neg x_{p(v),i} \vee \neg x_{v,j})$;
5. $y_{l,i,j} \Rightarrow \neg y_{l,i,k} \ (l \in \Sigma; 1 \le i, j, k \le C; j < k)$ – существует не более одного перехода из состояния *i* по символу *l* $(\neg y_{l,i,j} \vee \neg y_{l,i,k})$;
6. $\neg x_{v,i} \vee \neg x_{v,j} \ (v \in V; 1 \le i < j \le C)$ – каждому состоянию префиксного автомата *v* соответствует не более одного цвета;
7. $y_{l,i,1} \vee y_{l,i,2} \vee \ldots \vee y_{l,i,C} \ (l \in \Sigma; 1 \le i \le C)$ – существует как минимум один переход из состояния *i* по символу *l*;
8. $y_{l(v),i,j} \wedge x_{p(v),i} \Rightarrow x_{v,j} \ (v \in V \setminus \{v_r\}; 1 \le i, j \le C)$ – цвет состояния *v* префиксного автомата определен, если определены цвет родителя $p(v)$ и соответствующий переход $(\neg y_{l(v),i,j} \vee \neg x_{p(v),i} \vee x_{v,j})$;
9. $x_{v,i} \Rightarrow \neg x_{w,i}, ((v,w) \in E; 1 \le i \le C)$ – цвета состояний, соединенных ребром в графе совместимости, должны быть различны $(\neg x_{v,i} \vee \neg x_{w,i})$.

Таким образом, полученная формула состоит из $O(C^2|V|)$ дизъюнктов, и если в результаты работы программного средства для решения задачи о выполнимости будет найдена выполняющая подстановка, то по ней можно будет построить искомый ДКА.

1.4.5. Нахождение выполняющей подстановки и построение ДКА

Следующим шагом предлагаемого сведения является запись полученной формулы в файл в специальном DIMACS формате [18]. Затем выбирается некоторый современный эффективный SAT-solver, который обрабатывает файл и пытается найти выполняющую подстановку. Если SAT-solver сообщит о том, что такой подстановки не существует, то, значит, не существует автомата размера C, удовлетворяющего заданному словарю. Здесь могут быть применены различные техники по поиску требуемого значения C. Авторы оригинальной статьи [4] используют итеративный подход, чтобы найти ДКА минимального размера. Начальный размер C искомого ДКА равен размеру какой-нибудь большой клики, найденной в графе совместимости. Тогда если увеличивать C на единицу до тех пор, пока формула не станет выполнимой, то будет найден ДКА минимального размера. В данной работе используется аналогичный метод.

Смежные вершины графа совместимости не могут иметь одинаковый цвет, поэтому, очевидно, что все вершины в клике будут иметь разный цвет, а значит искомый автомат будет иметь не меньший размер. Для того чтобы найти большую клику в графе совместимости, используется следующий простой жадный алгоритм. Найдем вершину графа с максимальной степенью и добавим в клику. Затем, пока это возможно, будем искать вершину с максимальной степенью, соединенную со всеми вершинами уже включенными в клику.

В случае, когда SAT-solver сообщает о том, что выполняющая подстановка найдена, то, используя переменные $y_{l,i,j}$ и z_i, можно построить искомый ДКА. Первые переменные задают переходы автомата, а вторые – статус состояния – допускающее или недопускающее. Однако, можно построить искомый ДКА, используя только переменные $x_{v,i}$ и префиксный автомат. Обычно это оказывается проще. Действительно, достаточно рассмотреть любой переход из состояния одного цвета в состояние другого цвета в префиксном автомате, чтобы восстановить переход между соответствующими состояниями ДКА. Анало-

гично, достаточно рассмотреть любое допускающее или недопускающее состояние некоторого цвета в АРТА, чтобы восстановить правильный статус соответствующего состояния в автомате.

ГЛАВА 2. МОДИФИКАЦИЯ СВЕДЕНИЯ ЗАДАЧИ ПОСТРОЕНИЯ ДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ К ЗАДАЧЕ О ВЫПОЛНИМОСТИ И ПРЕДИКАТЫ НАРУШЕНИЯ СИММЕТРИИ

В данной главе в первом разделе представлена разработанная модификация сведения задачи построения ДКА к задаче о выполнимости булевой формулы, позволяющая решать задачу построения ДКА по зашумленным данным. Во втором разделе представлены разработанные предикаты нарушения симметрии на основе алгоритмов обхода графов, являющиеся основой данной работы.

2.1. Построение ДКА по зашумленному словарю

Сведение, описанное в предыдущей главе, работает в случае построения ДКА по точному словарю. В текущем разделе будет показано, как модифицировать данное сведение для того, чтобы применять его для зашумленных данных. Предположим, что не более чем K меток данных обучающих строк были изменены на противоположные. Решение представленной задачи было целью соревнования *GECCO 2004 Noisy DFA competition* [14] (где K было равно 10 процентам от количества обучающих строк). Лучшим подходом для решения поставленной задачи по итогам соревнования оказался эволюционный алгоритм. В [15] был предложен эволюционный алгоритм с некоторыми модификациями, и на данный момент он является наилучшим для построения ДКА по словарю с шумом.

В данной работе предлагается модифицировать компактное сведение из [4] для решения поставленной задачи. Идея предлагаемой модификации довольно проста: для каждого помеченного состояния префиксного дерева определим переменную, которая показывает, что метка данной вершины *могла быть* изменена на неверную. Формально, для каждого состояния $v \in V_{\pm} = V_{+} \cup V_{-}$ определим переменную f_v, которая истинна тогда и только тогда, когда метка состояния v *может быть* неверной. Используя эти переменные, можно модифицировать сведение, рассмотренное в предыдущем разделе, так, чтобы учитывать возможные ошибки в данном словаре. Чтобы добиться

этого, поменяем дизъюнкты, определяющие переменные допуска z_i (первый и второй пункты в предыдущем разделе): из-за возможной ошибки, они выполняются только в том случае, если переменная f_v ложна. Таким образом, новые дизъюнкты, определяющие переменные допуска z_i, могут быть выражены следующим образом: $\neg f_v \Rightarrow (x_{v,i} \Rightarrow z_i)$ для $v \in V_+$; $\neg f_v \Rightarrow (x_{v,i} \Rightarrow \neg z_i)$ для $v \in V_-$.

Чтобы ограничить количество ошибок числом *K*, будем использовать дополнительный массив, состоящий из *K* целочисленных переменных. Этот массив хранит номера состояний в префиксном дереве, для которых метки *могут быть* изменены на противоположные. Таким образом, f_v истинна тогда и только тогда, когда этот массив содержит в себе номер состояния *v*. Чтобы избежать рассмотрения изоморфных перестановок массива, потребуем, чтобы массив был отсортирован по возрастанию.

Чтобы представить этот массив в виде булевой формулы, определим переменные $r_{i,v}$ для $1 \le i \le K$ и $v \in V_\pm = \{v_1, \ldots, v_W\}$. $r_{i,v}$ истинна тогда и только тогда, когда в массиве на позиции *i* хранится номер состояния *v*. Чтобы соединить переменные f_v с $r_{i,v}$, добавим следующее ограничение: $f_v \Leftrightarrow (r_{1,v} \vee \ldots \vee r_{K,v})$ для всех $v \in V_\pm$.

Необходимо, чтобы выполнялось следующее утверждение: только одна из переменных $r_{i,v}$ может быть истинна для каждой позиции i в дополнительном массиве. Чтобы добиться этого, используем метод порядкового кодирования (the order encoding method), предложенный в [23]. Добавим дополнительные порядковые переменные $o_{i,v}$ для $1 \le i \le K$ и $v \in V_\pm = \{v_1, \ldots, v_W\}$. Положим, что для какого-то j переменная $o_{i,v}$ истинна для $v \in \{v_1, \ldots, v_j\}$ и ложна для $v \in \{v_{j+1}, \ldots, v_W\}$. Данное условие может быть выражено в булевой форме с помощью следующего условия: $o_{i,v_{j+1}} \Rightarrow o_{i,v_j}$ для $1 \le j < W$. Теперь положим, что $r_{i,v_j} \Leftrightarrow o_{i,v_j} \wedge \neg o_{i,v_{j+1}}$. Также добавим дизъюнкты $o_{i,v_j} \Rightarrow o_{i+1,v_{j+1}}$ (для $1 \le i < K$ и $1 \le j < W$), чтобы зафиксировать возрастающий порядок элементов в массиве.

Предложенные ограничения, приведенные в КНФ, перечислены в табл. 1; всего формула состоит из $O\left(C|V_{\pm}|+K|V_{\pm}|\right)$ дизъюнктов. Таким образом, чтобы модифицировать сведение для точной постановки задачи к задаче с зашумленным словарем, необходимо заменить дизъюнкты, определяющие переменные допуска z_i, и дизъюнкты, полученные из графа совместимости (пункты 1, 2 и 9 из списка типов дизъюнктов в предыдущем разделе), на дизъюнкты, перечисленные в табл. 1.

Таблица 1. Дизъюнкты для задачи с шумом

Условия	КНФ форма	Область значений
$\neg f_v \Rightarrow \left(x_{v,j} \Rightarrow z_j\right)$	$\neg x_{v,j} \vee z_j \vee f_v$	$1 \le j \le C; v \in V_+$
$\neg f_v \Rightarrow \left(x_{v,j} \Rightarrow \neg z_j\right)$	$\neg x_{v,j} \vee \neg z_j \vee f_v$	$1 \le j \le C; v \in V_-$
$f_v \Rightarrow \left(r_{1,v} \vee \ldots \vee r_{K,v}\right)$	$\neg f_v \vee r_{1,v} \vee \ldots \vee r_{K,v}$	$v \in V_{\pm}$
$r_{i,v} \Rightarrow f_v$	$\neg r_{i,v} \vee f_v$	$1 \le i \le K; v \in V_+$
$r_{i,v_j} \Rightarrow o_{i,v_j}$	$\neg r_{i,v_j} \vee o_{i,v_j}$	$1 \le i \le K; 1 \le j \le W$
$r_{i,v_j} \Rightarrow \neg o_{i,v_{j+1}}$	$\neg r_{i,v_j} \vee \neg o_{i,v_{j+1}}$	$1 \le i \le K; 1 \le j < W$
$o_{i,v_j} \wedge \neg o_{i,v_{j+1}} \Rightarrow r_{i,v_j}$	$\neg o_{i,v_j} \vee o_{i,v_{j+1}} \Rightarrow r_{i,v_j}$	$1 \le i \le K; 1 \le j < W$
$o_{K,v_W} \Rightarrow r_{K,v_W}$	$\neg o_{K,v_W} \vee r_{K,v_W}$	
$o_{i,v_{j+1}} \Rightarrow o_{i,v_j}$	$\neg o_{i,v_{j+1}} \vee o_{i,v_j}$	$1 \le i \le K; 1 \le j < W$
$o_{i,v_j} \Rightarrow o_{i+1,v_{j+1}}$	$\neg o_{i,v_j} \vee o_{i+1,v_{j+1}}$	$1 \le i < K; 1 \le j < W$

2.2. Предикаты нарушения симметрии

Как уже было сказано раннее в разделе 1.3., предикаты нарушения симметрии используются для уменьшения пространства поиска решения. Изоморфизм автоматов является отношением эквивалентности, что легко доказывается проверкой соответствующих аксиом. В задаче построения ДКА множество ав-

томатов разбивается на классы эквивалентности относительно данного отношения. Если доказано, что какой-то представитель класса эквивалентности не является решением исходной задачи, то рассматривать все остальные автоматы из данного класса нет никакого смысла.

Авторы [4] предлагают использовать в качестве предикатов нарушения симметрии ту же большую клику, которую используют для начала итеративной стратегии поиска минимального автомата. В любой правильной раскраске графа совместимости все состояния в клике должны иметь различный цвет. Таким образом, можно зафиксировать цвета состояний в клике перед выполнением основных вычислений. Тогда, если размер найденной клики равен k, то для каждого класса эквивалентности будет рассмотрено только $(n - k)!$ изоморфных автоматов, а не $n!$, что в среднем дает очень хороший прирост производительности. Позже будет показано, что нарушение симметрии с помощью клики несовместимо с предлагаемыми предикатами.

В задаче с зашумленными данными нельзя использовать несовместность вершин в префиксном дереве, так как нельзя определить ситуацию, когда принимающее состояние сливается с отвергающим, потому что неизвестны правильные метки строк. Таким образом, нет возможности построить и использовать граф совместимости, и подход к нарушению симметрии с помощью клики, соответственно, неприменим.

2.2.1. Предикаты нарушения симметрии на основе поиска в глубину

В данном разделе предлагается способ, как зафиксировать нумерацию состояний автомата, чтобы избежать рассмотрения изоморфных ДКА во время решения задачи SAT. Основной идеей предлагаемого подхода к нарушению симметрии является фиксирование номеров состояний в порядке обхода в глубину (depth-first search, DFS). Таким образом, для каждого класса эквивалентных по отношению изоморфизма автоматов будет рассмотрен только один представитель.

Рассмотрим схему работы алгоритма DFS. Для каждого непосещенного состояния ДКА необходимо найти все непосещенные смежные состояния и повторить поиск для них. Первым делом алгоритм рассматривает стартовое состояние (не уменьшая общности, можно предположить, что его номер – 1). Затем по очереди рассматриваются все дети этого состояния, для каждого из которых алгоритм повторяется. Рассмотрим вспомогательную структуру данных в виде массива с переходами. В массив будем добавлять номера вершин в порядке их обработки алгоритмом, а переходы будут копиями переходов ДКА, которые использовались во время обхода. Так как переходы в ДКА в поставленной задаче помечены символами из Σ, то введем дополнительное требование, чтобы дети обрабатывались в алфавитном порядке. Будем называть ДКА DFS-пронумерованным, если вспомогательный массив заполнен последовательными числами в возрастающем порядке, начиная с единицы. Пример DFS-пронумерованного ДКА с шестью состояниями показан на рис. 4 (жирным выделены ребра, которые использовались во время обхода в глубину – так называемое DFS-дерево); вспомогательный массив для этого обхода показан на рис. 5. ДКА, изображенный на рис. 6, не является DFS-пронумерованным – алгоритм обхода в глубину обработает состояние 3 раньше чем состояние 2 (переход $1 \xrightarrow{a} 3$ будет рассмотрен раньше чем переход $1 \xrightarrow{b} 2$).

Предлагается добавить условия, которые оставят для рассмотрения только DFS-пронумерованные ДКА. Напомним, что в рассматриваемом сведении данной задачи построения ДКА к задаче SAT содержатся переменные $y_{l,i,j}$ $(l \in \Sigma; 1 \le i, j \le C)$, которые истинны тогда и только тогда, когда из состояния i переход по символу l ведет в состояние j.

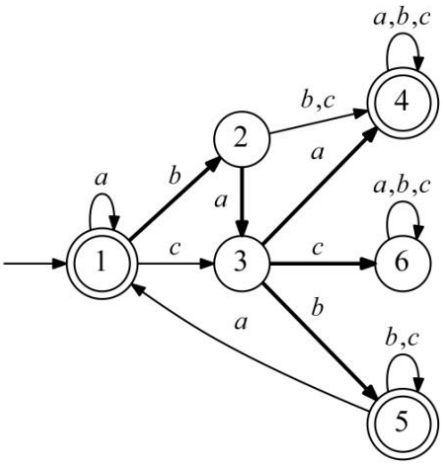

Рис. 4. Пример DFS-пронумерованного ДКА

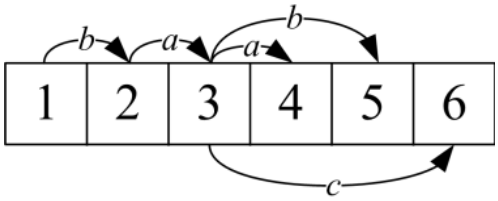

Рис. 5. Пример вспомогательного массива, построенного в ходе работы алгоритма DFS при обходе ДКА на рис. 4

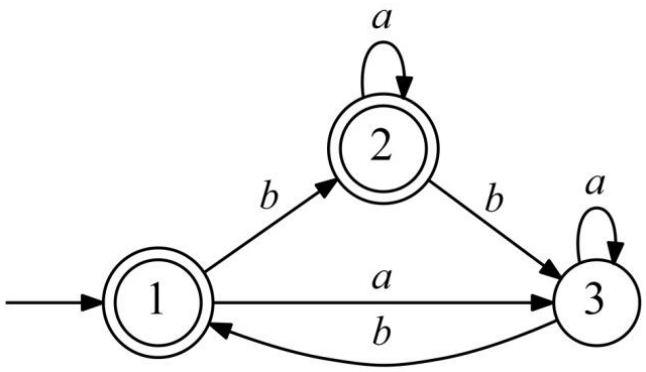

Рис. 6. Пример не DFS-пронумерованного ДКА

Для всех вершин необходимо определить родителя в DFS-дереве. Будем хранить родителей в переменных $p_{j,i}$ (для всех $1 \le i < j \le C$). $p_{j,i}$ истинна тогда и только тогда, когда состояние i является родителем состояния j в DFS-дереве. Все состояния, кроме начального, должны иметь родителя с меньшим номером. То есть

$$\bigwedge_{2\le j\le C}\left(p_{j,1}\vee p_{j,2}\vee\ldots\vee p_{j,j-1}\right).+$$

Определим переменные $p_{j,i}$ через переменные $y_{l,i,j}$, используя дополнительные переменные $t_{i,j}$. В DFS-пронумерованном ДКА состояние j должно быть добавлено в очередь во время рассмотрения состояния с максимальным номером i среди состояний, которые имеют переход в j:

$$\bigwedge_{1\le i<j\le C}\left(p_{j,i}\Leftrightarrow t_{i,j}\wedge\neg t_{i+1,j}\wedge\ldots\wedge\neg t_{j-1,j}\right),$$

где $t_{i,j}\equiv 1$ тогда и только тогда, когда существует переход между i и j. Определим эти дополнительные переменные, используя переменные $y_{l,i,j}$:

$$\bigwedge_{1\le i<j\le C}\left(t_{i,j}\Leftrightarrow y_{l_1,i,j}\vee\ldots\vee y_{l_L,i,j}\right).$$

В DFS-пронумерованном ДКА, родители состояний должны быть упорядочены. Если i – номер родителя состояния с номером j, а k – номер какого-то состояния больший i и меньший j ($i<k<j$), то не существует перехода из состояния k в вершину с номером q большим, чем j (рис. 7):

$$\bigwedge_{1\le i<k<j<q<C}\left(p_{j,i}\Rightarrow\neg t_{k,q}\right).$$

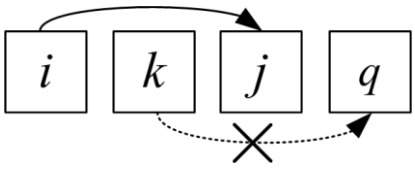

Рис. 7. Часть массива, иллюстрирующая предикаты упорядочивания родителей. Пунктиром показан переход, которого не должно быть в DFS-пронумерованном ДКА

Чтобы обеспечить DFS-пронумерованность ДКА, необходимо упорядочить детей в алфавитном порядке по символам переходов. Отдельно рассмотрим два случая: алфавит Σ состоит из двух символов $\{a,b\}$ и более чем из двух

символов $\{l_1,\ldots,l_L\}$. В случае с двухсимвольным алфавитом только два состояния могут иметь одного родителя i – j и k (где, не уменьшая общности, $j<k$). В этом случае переход, который начинается в состоянии i и помечен символом a, должен оканчиваться в состоянии j, а не в k:

$$\bigwedge_{1\le i<j<k<C}\left(p_{j,i}\wedge p_{k,i}\Rightarrow y_{a,i,j}\right).$$

Во втором случае требуется ввести третий тип переменных. Будем хранить минимальный, в алфавитном порядке, символ среди переходов между состояниями: $m_{l,i,j}$ истинна тогда и только тогда, когда существует переход $i\xrightarrow{l}j$, и не существует перехода между этими состояниями по меньшему, в алфавитном порядке, символу. Соединим эти переменные с переходами в ДКА, добавив следующие связывающие условия:

$$\bigwedge_{1\le i<j\le C}\bigwedge_{1\le n\le L}\left(m_{l_n,i,j}\Leftrightarrow y_{l_n,i,j}\wedge\neg y_{l_{n-1},i,j}\wedge\ldots\wedge\neg y_{l_1,i,j}\right).$$

Осталось только расставить состояния j и k, имеющие одного предка i, в алфавитном порядке по символам переходов между ними и предком i (рис. 8):

$$\bigwedge_{1\le i<j<k\le C}\bigwedge_{1\le m<n\le L}\left(p_{j,i}\wedge p_{k,i}\wedge m_{l_n,i,j}\Rightarrow\neg m_{l_m,i,k}\right).$$

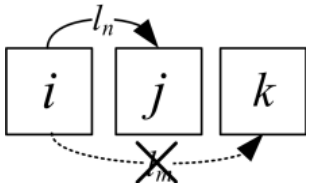

Рис. 8. Иллюстрация к предикатам, упорядочивающим состояния в алфавитном порядке. Если состояние i – это предок состояний j и k, l_n (l_m) – минимальные в алфавитном порядке символы на переходах между i и j (i и k), тогда l_m не может быть меньше чем l_n

Выше были описаны предикаты нарушения симметрии, которые выражаются перечисленными условиями. Предикаты (для случая с более чем двухсимвольным алфавитом) сводятся к $O\left(C^4 + C^3L^2\right)$ КНФ дизъюнктов.

2.2.2. Предикаты нарушения симметрии на основе поиска в ширину

В данном разделе рассматривается модификация предыдущего подхода с предикатами на основе поиска в глубину, которая зафиксирует нумерацию состояний автомата в порядке обхода в ширину (breadth-first search, BFS). Данная идея использовалась в функции *NatOrder* в методе слияния состояний, описанном в [24] и в преобразующем алгоритме *Move To Front*, использующемся в генетическом алгоритме из [25].

Алгоритм BFS использует структуру данных *очередь*, чтобы хранить промежуточные результаты во время обхода графа. Первым делом, в очередь кладется номер стартового состояния ДКА (не уменьшая общности, можно предположить, что это 1). Затем, до тех пор, пока очередь не пуста, из нее извлекается номер состояния i, и в очередь добавляются номера всех его детей, которые еще не были ранее добавлены в очередь. Аналогично DFS-подходу дети добавляются в алфавитном порядке. Будем называть ДКА BFS-пронумерованным, если его состояния пронумерованы в том порядке, в котором они доставались из очереди. Пример BFS-пронумерованного ДКА с шестью состояниями показан на рис. 9 (жирным выделены ребра, которые образуют BFS-дерево); очередь для этого обхода в ширину показана на рис. 10.

Все переменные, использованные для задания порядка DFS, используются для задания порядка BFS. Рассмотрим, какие условия надо изменить, чтобы поддерживать порядок обхода в ширину.

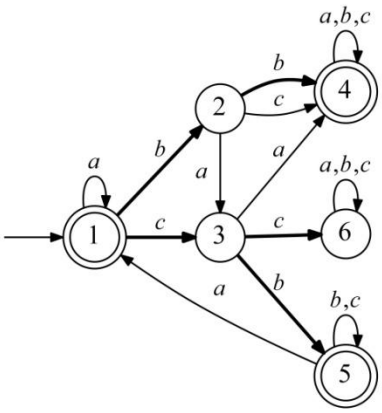

Рис. 9. Пример BFS-пронумерованного ДКА

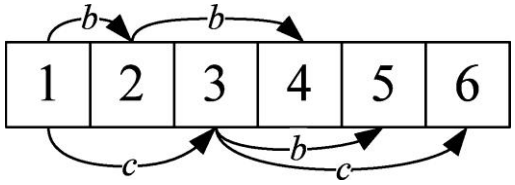

Рис. 10. Пример очереди, использованной в алгоритме BFS при обходе ДКА на рис. 8

В BFS-пронумерованном ДКА меняется определение переменных $p_{j,i}$. В BFS-пронумерованном ДКА состояние j должно быть добавлено в очередь во время рассмотрения состояния с минимальным номером i среди состояний, которые имеют переход в j:

$$\bigwedge_{1 \le i < j \le C} \left(p_{j,i} \Leftrightarrow t_{i,j} \wedge \neg t_{i+1,j} \wedge \ldots \wedge \neg t_{j-1,j} \right),$$

Также меняется условие на упорядоченность родителей состояний. Состояние j должно быть добавлено в очередь перед следующим состоянием $j+1$, поэтому номер k родителя состояния $j+1$ не может быть меньше, чем номер i родителя состояния j (рис. 11):

$$\bigwedge_{1 \le k < i < j < C} \left(p_{j,i} \Rightarrow \neg p_{j+1,k} \right).$$

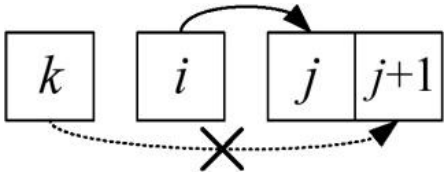

Рис. 11. Часть очереди, иллюстрирующая предикаты упорядочивания родителей. Пунктиром показан переход, который невозможен в BFS-пронумерованном ДКА

Во всех условиях, где рассматривалось два состояния с общим родителем, в силу алгоритма BFS, достаточно рассмотреть только соседние два состояния:

$$\bigwedge_{1\le i<j<C}\left(p_{j,i}\wedge p_{j+1,i}\Rightarrow y_{a,i,j}\right).$$

$$\bigwedge_{1\le i<j\le C}\bigwedge_{1\le k<n\le L}\left(p_{j,i}\wedge p_{j+1,i}\wedge m_{l_n,i,j}\Rightarrow\neg m_{l_k,i,j+1}\right).$$

Предикаты на основе BFS (для случая с более чем двухсимвольным алфавитом) сводятся к $O\left(C^3+C^2L^2\right)$ КНФ дизъюнктов, перечисленных в табл. 2. Реализация предложенных предикатов на основе обоих обходов графов и всех алгоритмов может быть найдена в *github*-репозитории международной лаборатории «Компьютерные технологии»[1].

[1] https://github.com/ctlab/DFA-Inductor

Таблица 2. Предикаты нарушения симметрии на основе BFS

Условия	КНФ форма	Область значений
$t_{i,j} \Rightarrow \left(y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j}\right)$	$\neg t_{i,j} \vee y_{l_1,i,j} \vee \ldots \vee y_{l_L,i,j}$	$1 \le i < j \le C$
$y_{l,i,j} \Rightarrow t_{i,j}$	$\neg y_{l,i,j} \vee t_{i,j}$	$1 \le i < j \le C;$ $l \in \Sigma$
$m_{l,i,j} \Rightarrow y_{l,i,j}$	$\neg m_{l,i,j} \vee y_{l,i,j}$	$1 \le i < j \le C;$ $l \in \Sigma$
$m_{l_n,i,j} \Rightarrow \neg y_{l_k,i,j}$	$\neg m_{l_n,i,j} \vee \neg y_{l_k,i,j}$	$1 \le i < j \le C;$ $1 \le k < n \le L$
$\left(y_{l_n,i,j} \wedge \neg y_{l_{n-1},i,j} \wedge \ldots \wedge \neg y_{l_1,i,j}\right) \Rightarrow m_{l_n,i,j}$	$\neg y_{l_n,i,j} \vee y_{l_{n-1},i,j} \vee \ldots \vee y_{l_1,i,j} \vee m_{l_n,i,j}$	$1 \le i < j \le C;$ $1 \le n \le L$
$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$p_{j,1} \vee p_{j,2} \vee \ldots \vee p_{j,j-1}$	$2 \le j \le C$
$p_{j,i} \Rightarrow t_{i,j}$	$\neg p_{j,i} \vee t_{i,j}$	$1 \le i < j \le C$
$p_{j,i} \Rightarrow \neg t_{k,j}$	$\neg p_{j,i} \vee \neg t_{k,j}$	$1 \le k < i < j \le C$
$\left(t_{i,j} \wedge \neg t_{i-1,j} \wedge \ldots \wedge \neg t_{1,j}\right) \Rightarrow p_{j,i}$	$\neg t_{i,j} \vee t_{i-1,j} \vee \ldots \vee t_{1,j} \vee p_{j,i}$	$1 \le i < j \le C$
$p_{j,i} \Rightarrow \neg p_{j+1,k}$	$\neg p_{j,i} \vee \neg p_{j+1,k}$	$1 \le k < i < j < C$
$\left(p_{j,i} \wedge p_{j+1,i} \wedge m_{l_n,i,j}\right) \Rightarrow \neg m_{l_k,i,j+1}$	$\neg p_{j,i} \vee \neg p_{j+1,i} \vee \neg m_{l_n,i,j} \vee \neg m_{l_k,i,j+1}$	$1 \le i < j < C;$ $1 \le k < n \le L$

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

В данной главе приведены результаты экспериментального исследования. Все эксперименты проводились на компьютере с процессором AMD Opteron 6378 2.4 GHz с установленной операционной системой Ubuntu 14.04. Все алгоритмы были реализованы на языке Java, для решения задачи SAT использовался пакет lingeling.

3.1. Алгоритм генерации данных

Использовался собственный алгоритм для генерации входных данных. Этот алгоритм строит множество строк по следующим параметрам: N – размер ДКА, A – размер алфавита, S – количество строк, которые должны быть сгенерированы, K – уровень шума в данных (процент сгенерированных обучающих слов, которые должны быть случайно изменены).

Алгоритм генерирует автомат размера N, затем по нему генерирует строки и сохраняет метки о том, принимаются ли они автоматом или нет. Если указан ненулевой уровень шума, то случайно выбирается необходимое число меток, и они меняются на противоположные.

Рассмотрим подробнее, как генерируется автомат. Генерируется N состояний автомата, которые пронумерованы числами от 0 до $N-1$. Состояние с номером 0 является стартовым. Для всех состояний кроме стартового случайно выбирается номер родителя среди состояний с меньшим номером. Если у выбранного родителя уже есть максимально возможное количество детей, то повторяется процедуру выбора случайного родителя. После выбора родителя случайно выбирается символ перехода. Если у родителя уже есть исходящий переход, помеченный таким символом, то символ случайно выбирается еще раз. Полученный недостроенный автомат будет иметь одну компоненту связности и все вершины в нем будут достижимы. Действительно, каждое следующее состояние связано с одним из предыдущих, а значит, существует цепочка переходов, которые соединяют нулевое состояние с любым другим.

Затем последовательно для всех состояний перебираются символы алфавита и для тех символов, по которым нет исходящего перехода, генерируется переход в случайно выбранное состояние. И, наконец, для каждого состояния случайно определяется, будет оно принимающим или нет.

Рассмотрим, как генерируются строки по данному автомату. Генерируется последовательность символов алфавита определенной длины, для которой определяется метка принадлежности закодированному языку. Для определения длины используется следующий подход. Генерируется $C = 2^{l-2}$ слов длины l. Изначально, $l = 3$. Затем данный набор строк перемешивается в случайном порядке и записывается в файл.

Следует заметить, что иногда по построенному словарю может быть построен автомат меньшего размера, чем N. Поэтому следует проверить размер минимального ДКА для данного словаря с помощью какого-нибудь известного быстрого метода решения задачи SAT.

3.2. Построение ДКА по точным данным

Для построения ДКА по точным данным в ходе экспериментов использовались случайно сгенерированные данные. Сначала рассмотрим случай, когда целевой ДКА существует, и булева формула выполнима. Использовались следующие параметры: $N \in [10; 20]$, $A = 2$, $S = 50N$. Каждый эксперимент был повторен 100 раз. Ограничение по времени было установлено на 3600 секунд. Сравнивалось предложенное решение, использующее предикаты нарушения симметрии, основанные на обходе в глубину, решение, использующее предикаты нарушения симметрии, основанные на обходе в ширину, и лучший на данный момент подход, использующий предикаты нарушения симметрии, основанные на клике. Результаты показаны в табл. 2. Значение TL в ячейке таблицы говорит о том, что ни один из тестов данного размера не был решен за отведенный лимит времени соответствующим методом. Результаты показывают, что оба разработанных подхода улучшают результаты существующего решения. При этом при увеличении размера целевого ДКА от 14 и выше, подход осно-

ванный на обходе в ширину значительно выигрывает у подхода, основанного на обходе в глубину. BFS подход позволяет решить все поставленные задачи, в то время как DFS подход, начиная с размера ДКА от 15 и выше не успевает за установленный лимит времени решить некоторые примеры. Так, для $N = 20$, решено лишь 19 процентов задач. Подход, основанный на клике, не всегда укладывается в предел времени, уже начиная с $N = 11$. А для размера ДКА от 16 и выше вообще не справляется ни с одним примером.

Таблица 3. Построение ДКА по точным данным, $S = 50N$

N	DFS, сек	BFS, сек	Клика, сек	Успех DFS, %	Успех BFS, %	Успех клики, %
10	**80.14**	80.28	158.18	**100**	**100**	**100**
11	109.72	**109.12**	337.25	**100**	**100**	98
12	159.94	**151.60**	684.42	**100**	**100**	71
13	200.49	**196.32**	1146.27	**100**	**100**	48
14	301.05	**254.40**	923.16	**100**	**100**	10
15	406.44	**332.57**	1780.09	99	**100**	6
16	824.72	**560.39**	TL	**100**	**100**	0
17	1217.62	**631.86**	TL	95	**100**	0
18	1722.37	**685.71**	TL	90	**100**	0
19	2294.11	**778.88**	TL	73	**100**	0
20	2870.17	**903.09**	TL	19	**100**	0

3.3. Построение ДКА по зашумленным данным, когда целевой ДКА существует

Для построения ДКА по зашумленным данным в ходе экспериментов использовались случайно сгенерированные данные. Сначала рассмотрим случай, когда целевой ДКА существует, и булева формула выполнима. Использовались следующие параметры: $N \in [5;10]$, $A = 2$, $S \in \{10N, 25N, 50N\}$. Каждый эксперимент был повторен 100 раз. Ограничение по времени было установлено на 1800 секунд. Сравнивалось предложенное решение, использующее предикаты нарушения симметрии, основанные на обходе в ширину, с лучшим на данный момент подходом, основанном на эволюционном алгоритме, из [15]. Первые же эксперименты показали, что эволюционный алгоритм полностью выигрывает у нашего подхода при $K > 4$. Поэтому этот параметр менялся в диапазоне $1-4\%$.

Результаты показаны в табл. 4, 5 и 6. В ней представлены только те случаи, которые успели решиться за отведенное время. Эти результаты показывают, что подход, основанный на поиске в ширину, находит решение сопоставимо по времени с эволюционным алгоритмом только тогда, когда N мало (<7), уровень шума мал ($1-4\%$) и количество строк также мало ($<50N$). Подход, основанный, на DFS в целом уступает подходу, основанному на BFS.

Таблица 4. Построение ДКА по зашумленным данным, $S=10N$

N	K	DFS, сек	BFS, сек	EA, сек
5	2	0.38	**0.22**	1.22
5	4	0.67	**0.59**	1.1
6	2	1.74	**1.05**	2.94
6	4	3.55	3.34	**2.85**
7	1	7.3	**4.34**	21.36
7	3	19.5	**17.22**	19.16
8	1	37.2	**17.89**	30.29
8	2	276.5	163.92	**19.8**

Таблица 5. Построение ДКА по зашумленным данным, $S=25N$

N	K	DFS, сек	BFS, сек	EA, сек
5	1	0.86	**0.54**	2.77
5	2	3.56	2.42	**1.80**
6	1	9.27	**6.3**	11.65
6	2	26.7	13.3	**4.8**
7	1	67.96	31.01	**17.24**
7	2	534.83	286.76	**13.11**
8	1	498.5	239.46	**21.73**

Таблица 6. Построение ДКА по зашумленным данным, $S=50N$

N	K	DFS, сек	BFS, сек	EA, сек
5	1	6.79	**4.2**	6.07
5	2	18.42	12.87	**3.05**
6	1	38.48	20.76	**20.39**
6	2	229.8	107.94	**11.28**

3.4. Построение ДКА по зашумленным данным, когда целевой ДКА не существует

В последнем эксперименте рассматривался случай, когда целевой ДКА не существовал и, соответственно, булева формула была невыполнима. В этом эксперименте снова использовался случайно сгенерированный набор данных. Алгоритм по поиску ДКА запускался со следующими параметрами: $N \in [5; 7]$, $A = 2$, $S = 50N$, $K \in [1; 2]$ процента. Однако строки словаря генерировались по ДКА размером $(N+1)$. Следует заметить, что эволюционный алгоритм из [15] не может точно определить, что ДКА, удовлетворяющего заданному словарю, не существует. С другой стороны, все методы, основанные на сведении к задаче SAT, могут справиться с этой проблемой. Поэтому сравнивалась собственная реализация компактного сведения к задаче SAT без использования предикатов нарушения симметрии с собственной реализацией с использованием предикатов нарушения симметрии, основанных на обходе в ширину. Каждый эксперимент был повторен 100 раз. Ограничение по времени было снова установлено на 1800 секунд. Результаты представлены в табл. 7. Из нее видно, что основанная на BFS стратегия значительно уменьшает среднее время установления того факта, что ДКА не существует. Видно также, что подход на основе BFS эффективнее подхода на основе DFS при увеличении *N* и *K*.

Таблица 7. Построение ДКА по зашумленным данным, когда целевой ДКА не существует

N	K	DFS, сек	BFS, сек	WO, сек	Успех DFS, %	Успех BFS, %	Успех WO, %
5	1	**9.35**	9.73	278.79	**100**	**100**	**100**
5	2	26.3	**24.24**	980.09	**100**	**100**	77
6	1	52.3	**39.9**	TL	**100**	**100**	0
6	2	380.28	**288.81**	TL	**100**	**100**	0
7	1	837.99	**512.56**	TL	92	98	0
7	2	TL	TL	TL	0	0	0

ЗАКЛЮЧЕНИЕ

В данной работе были предложены предикаты нарушения симметрии, которые могут быть добавлены в булеву формулу, представляющую различные задачи построения ДКА. Добавив предлагаемые предикаты, можно сузить пространство поиска решения задачи SAT, требуя, чтобы состояния ДКА были пронумерованы в порядке обхода в глубину (depth-first search, DFS) или в порядке обхода в ширину (breadth-first search, BFS). В экспериментальной части было показано, что использование разработанных предикатов нарушения симметрии позволяет значительно улучшить время работы оригинального метода, который был лучшим в своем классе.

Также была рассмотрена задача построения ДКА по зашумленным данным. Была предложена модификация сведения точной задачи к задаче SAT из [4], которая применима для задачи с зашумленным словарем. Для реализации компактного сведения был использован метод порядкового кодирования. Было показано, что ранее предложенная техника для нарушения симметрии, основанная на поиске большой клики, неприменима для решения задачи с зашумленным словарем, в то время как предложенная техника, основанная на алгоритмах обхода графа, применима. Было показано, что основанная на данных алгоритмах стратегия может быть применена для задачи с шумом, когда автомата определенного размера, удовлетворяющего данному словарю, не существует. Лучший на данный момент подход к решению задачи построения ДКА по словарю, основанный на эволюционном алгоритме, не может установить факт отсутствия целевого ДКА. В экспериментальной части было показано, что рассматриваемый подход с предикатами нарушения симметрии на основе алгоритмов обхода графов значительно выигрывает у того же алгоритма, но без использования каких-либо предикатов. Также было показано, что предлагаемый подход сопоставим с эволюционным алгоритмом, если размер искомого ДКА мал, уровень шума мал и количество строк в словаре также мало.

Следует отдельно заметить, что на базе предлагаемых предикатов нарушения симметрии может быть разработан эффективный метод нахождения всех

автоматов минимального размера. Данная задача не имеет эффективного решения на данный момент.

Также в дальнейшем планируется свести задачу построения ДКА по зашумленному словарю к задаче Max-SAT, чтобы ограничивать количество возможных исправлений без использования дополнительного массива целочисленных переменных. Также планируется поэкспериментировать с альтернативными методами целочисленного кодирования.

СПИСОК ИСТОЧНИКОВ

1. *Hopcroft J., Motwani R., Ullman J.* Introduction to Automata Theory, Languages, and Computation. 2006. Boston: Pearson/Addison Wesley.
2. *De La Higuera C.* A bibliographical study of grammatical inference. Pattern recognition, 2005, vol. 38, no. 9, pp. 1332–1348. Elsevier.
3. *Mernik M., Hrncic D., Bryant B., Sprague A., Gray J., Liu Q., Javed F.* Grammar Inference Algorithms and Applications in Software Engineering. XXII International Symposium on Information, Communication and Automation Technologies, ICAT'09, 2009, pp. 1-7. IEEE.
4. *Heule M.J., Verwer S.* Exact DFA Identification Using SAT Solvers. 10th International Colloquium Conference on Grammatical Inference: Theoretical Results and Applications, ICGI'10, 2010, pp. 66–79. Springer Berlin Heidelberg.
5. *Ulyantsev V., Zakirzyanov I., Shalyto A.* BFS-Based Symmetry Breaking Predicates for DFA Identification. 9th International Conference on Language and Automata Theory and Applications, LATA'15, 2015, pp. 611–622. Springer International Publishing.
6. *Gold E.M.* Complexity of automaton identification from given data. Information and control, 1978, vol. 37, no. 3, pp. 302–320. Elsevier.
7. *Pitt L., Warmuth M.K.* The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of the ACM, 1993, vol. 40, no. 1, pp. 95–142. ACM.
8. *Lang K.J., Pearlmutter B.A., Price R.A.* Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. 4th International Colloquium on Grammatical Inference, ICGI'98, 1998, pp. 1–12. Springer Berlin Heidelberg.
9. *Lang K.J.* Faster Algorithms for Finding Minimal Consistent DFAs. Technical report, 1999, NEC Research Institute, USA.
10. *Bugalho M., Oliveira A.L.* Inference of regular languages using state merging algorithms with search. Pattern Recognition, 2005, vol. 38, no. 9, pp. 1457-1467. Elsevier.

11. *Dupont P.* Regular Grammatical Inference from Positive and Negative Samples by Genetic Search: the GIG Method. 2nd International Colloquium on Grammatical Inference and Applications, ICGI'94, 1994, pp. 236–245. Springer Berlin Heidelberg.
12. *Luke S., Hamahashi S., Kitano H.* "Genetic" Programming. Genetic and Evolutionary Computation Conference, GECCO'99, 1999, vol. 2, pp. 1098–1105. Morgan Kaufmann.
13. *Lucas S.M., Reynolds T.J.* Learning DFA: evolution versus evidence driven state merging. 5th IEEE Congress on Evolutionary Computation, CEC'03, 2003, vol. 1, pp. 351–358. IEEE.
14. *Lucas S.M.* Gecco 2004 noisy dfa results [Электронный ресурс]. Режим доступа http://cswww.essex.ac.uk/staff/sml/gecco/results/NoisyDFA/NoisyDFAResults.html, свободный. Яз. англ. (дата обращения 01.04.2015).
15. *Lucas S.M., Reynolds T.J.* Learning deterministic finite automata with a smart state labeling evolutionary algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, vol. 27, no. 7, pp. 1063–1074. IEEE.
16. *Walkinshaw N., Lambeau B., Damas C., Bogdanov K., Dupont P.* STAMINA: a competition to encourage the development and assessment of software model inference techniques. Empirical software engineering, 2013, vol. 18, no. 4, pp. 791–824. Springer US.
17. *Biere A., Heule M., van Maaren H.* Handbook of satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications. 2009. Amsterdam: IOS Press.
18. CNF files [Электронный ресурс]. Режим доступа http://www.satcompetition.org/2004/format-solvers2004.html свободный. Яз. англ. (дата обращения 23.05.2015).
19. *Amla N., Du X., Kuehlmann A., Kurshan R.P., McMillan K.L.* An Analysis of SAT-Based Model Checking Techniques in an Industrial Environment. Correct Hardware Design and Verification Methods – 13th IFIP WG 10.5 Advanced

Research Working Conference, CHARME 2005, 2005, pp. 254–268. Springer Berlin Heidelberg.

20. *Lohfert R., Lu J., Zhao D.* Solving SQL Constraints by Incremental Translation to SAT. New Frontiers in Applied Artificial Intelligence: 21st International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems, IEA/AIE'08, 2008, pp. 669–676. Springer Berlin Heidelberg.
21. *Galeotti J.P., Rosner N., Lopez Pombo C.G., Frias M.F.* TACO: Efficient SAT-Based Bounded Verification Using Symmetry Breaking and Tight Bounds. IEEE Transactions on Software Engineering, 2013, vol. 39, no. 9, pp. 1283–1307. IEEE.
22. *Ulyantsev V., Tsarev F.* Extended finite-state machine induction using sat-solver. 10th International Conference on Machine Learning and Applications and Workshops, ICMLA'11, 2011, vol. 2, pp. 346–349. IEEE.
23. *Chambers L.D.* Practical handbook of genetic algorithms: complex coding systems, Volume III. 1998. USA: CRC press.
24. *Hölldobler S., Nguyen V.* An Efficient of the At-Most-One Constraint. Technical report. Knowledge Representation and Reasoning Group 2013-04, 2013, Technische Universität Dresden, Germany.
25. *Lambeau B., Damas C., Dupont P.* State-Merging DFA Induction Algorithms With Mandatory Merge Constraints. 9th International Colloquium on Grammatical Inference, ICGI'08, 2008, pp. 139–153. Springer-Verlag Berlin Heidelberg.