Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Модификации эвристик SAT-решателей для генерации детерминированных конечных автоматов по примерам поведения

Обучающийся / Student Трапезников Семен Станиславович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M3439
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2017
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Руководитель ВКР/ Thesis supervisor Ульянцев Владимир Игоревич, кандидат технических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

Документ подписан
Трапезников Семен Станиславович
02.06.2022

(эл. подпись/ signature)

Трапезников Семен Станиславович
(Фамилия И.О./ name and surname)

Руководитель ВКР/ Thesis supervisor

Документ подписан
Ульянцев Владимир Игоревич
02.06.2022

(эл. подпись/ signature)

Ульянцев Владимир Игоревич
(Фамилия И.О./ name and surname)

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

АННОТАЦИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ SUMMARY OF A GRADUATION THESIS

Обучающийся / Student Трапезников Семен Станиславович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M3439
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2017
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Модификации эвристик SAT-решателей для генерации детерминированных конечных автоматов по примерам поведения
Руководитель ВКР/ Thesis supervisor Ульянцев Владимир Игоревич, кандидат технических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ DESCRIPTION OF THE GRADUATION THESIS

Цель исследования / Research goal
Расширение области применения SAT-решателей при решении задачи построения ДКА по примерам поведения путем разработки эвристик, использующих приоритет и активность переменных.
Задачи, решаемые в ВКР / Research tasks
Изучить работу "Learning Minimal DFA: Taking Inspiration from RPNI to Improve SAT Approach" и предложенные в ней эвристики. Изучить другие эвристики, в частности предикаты нарушения симметрии на основе алгоритма поиска в ширину из "Методы генерации детерминированных конечных автоматов с использованием сокращения пространства поиска при решении задачи выполнимости". Провести экспериментальное исследование алгоритма построения ДКА по примерам поведения при комбинации различных эвристик. На основе полученных результатов разработать новые эвристики. Сравнить разработанные эвристики с существующими подходами. Изучить эффективность разработанных эвристик на разных SAT-решателях.
Краткая характеристика полученных результатов / Short summary of results/findings
Разработана эвристика активности BFS предикатов нарушения симметрии. Проведено сравнение с существующими решениями, в котором эвристика показывает уменьшение времени решения задачи примерно на 50%. Показана эффективность эвристики на

различных SAT-решателях.

Обучающийся/Student

Документ подписан
Трапезников Семен Станиславович
02.06.2022

(эл. подпись/ signature)

Трапезников Семен Станиславович

(Фамилия И.О./ name and surname)

Руководитель ВКР/ Thesis supervisor

Документ подписан
Ульянцев Владимир Игоревич
02.06.2022

(эл. подпись/ signature)

Ульянцев Владимир Игоревич

(Фамилия И.О./ name and surname)

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ / OBJECTIVES FOR A GRADUATION THESIS

Обучающийся / Student Трапезников Семен Станиславович
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет информационных технологий и программирования
Группа/Group M3439
Направление подготовки/ Subject area 01.03.02 Прикладная математика и информатика
Образовательная программа / Educational program Информатика и программирование 2017
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Бакалавр
Тема ВКР/ Thesis topic Модификации эвристик SAT-решателей для генерации детерминированных конечных автоматов по примерам поведения
Руководитель ВКР/ Thesis supervisor Ульянцев Владимир Игоревич, кандидат технических наук, Университет ИТМО, факультет информационных технологий и программирования, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed
Техническое задание и исходные данные к работе:
Требуется разработать эвристики для задачи построения ДКА по примерам поведения с помощью SAT-решателя.
Эвристики должны использовать приоритет и активность переменных.
Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов):
Пояснительная записка должна содержать описание предметной области и эвристик, используемых при решении задачи построения ДКА.
Требуется описать разработанные эвристики, провести экспериментальное сравнение с существующими решениями.

Форма представления материалов ВКР / Format(s) of thesis materials:
Пояснительная записка, программный код, презентация

Дата выдачи задания / Assignment issued on: 21.12.2021

Срок представления готовой ВКР / Deadline for final edition of the thesis 15.05.2022

Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: да /

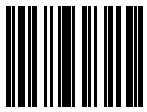
yes

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/ Thesis supervisor

Документ подписан
Ульянцев Владимир Игоревич
02.06.2022

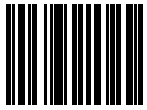


(эл. подпись)

Ульянцев Владимир Игоревич

Задание принял к исполнению/ Objectives assumed BY

Документ подписан
Трапезников Семен Станиславович
02.06.2022

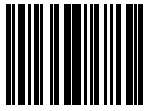


(эл. подпись)

Трапезников Семен Станиславович

Руководитель ОП/ Head of educational program

Документ подписан
Станкевич Андрей Сергеевич
03.06.2022



(эл. подпись)

Станкевич Андрей Сергеевич

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Детерминированные конечные автоматы (ДКА) — это конечный автомат, способный принимать или отклонять заданную ему строку, представляющую из себя конечную последовательность символов, путем прохождения через последовательность состояний, определяемых заданной последовательностью. ДКА распознает множество регулярных языков, что делает его полезным при решении задач в различных областях науки, например синтаксическом распознавании, сравнении с образцом, а в задачах вычислительной биологии.

Часто построение ДКА происходит с помощью примеров поведения: двух словарей, первый из которых содержит в себе слова, которые автомат принимает, а второй содержит слова, которые автомат отклоняет. Существует множество различных алгоритмов построения ДКА по примерам поведения, среди которых выделяют подход, основанный на сведении к задаче выполнимости булевых формул [6]. Современные алгоритмы, решающие задачу данным методом, сводятся к тому, чтобы с помощью специального кодирования преобразовать примеры поведения в набор булевых предикатов задачи SAT, которые в свою очередь будут переданы в SAT-решатель, специальное программное средство, позволяющее решать задачу выполнимости булевых формул. Для преобразования словарей поведения в набор предикатов используется кодирование, предложенное Хойлом и Вервером [11], а также различные дополнительные избыточные эвристики и предикаты.

Большинство научных работ, посвященных решению задачи построения ДКА по примерам поведения с помощью сведения к задаче выполнимости булевых формул заключаются в том, что исследователи на основе кодирования Хойла и Вервера разрабатывают свои эвристики, позволяющие уменьшить время работы алгоритма. При этом они часто игнорируют существование других работ по этой теме и эвристики, разработанные в этих работах.

В 2019 году канадскими исследователями Авелланеда и Петренко была выпущена работа [4], в которой они предлагают эвристики решения задачи построения ДКА, которые являются новыми и необычными для работ, посвященных данной теме. В частности, они модифицируют SAT-решатель для изменения активности переменных. К сожалению, данная работа также не рас-

сматривает существование других работ и эвристик, посвященных задаче построения ДКА по примерам поведения.

Целью настоящей работы является расширение области применения SAT-решателей при решении задачи построения ДКА по примерам поведения путем разработки эвристик, использующих приоритет и активность переменных.

Перед началом исследования были сформулированы следующие задачи.

- Изучить работу Авелланеды и Петренко [4] и предложенные исследователями эвристики.
- Изучить другие эвристики, в частности предикаты нарушения симметрии на основе алгоритма поиска в ширину [16].
- Провести экспериментальное исследование алгоритма построения ДКА по примерам поведения при комбинации различных эвристик.
- На основе полученных результатов разработать новые эвристики.
- Сравнить разработанные эвристики с существующими подходами.
- Изучить эффективность разработанных эвристик на разных SAT-решателях.

ГЛАВА 1. ЗАДАЧА ПОСТРОЕНИЯ ДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ ПО ПРИМЕРАМ ПОВЕДЕНИЯ И ОБЗОР РАССМАТРИВАЕМЫХ В ИССЛЕДОВАНИИ ЭВРИСТИК

В данной главе приводится описание задачи построения детерминированных конечных автоматов, рассматриваются существующие решения задачи. Далее разбирается подход, лежащий в основе данного исследования, и приводится обзор статьи [4].

1.1. Задача построения ДКА и определения

Детерминированный Конечный Автомат (ДКА) это набор из пяти элементов $(Q, \Sigma, \delta, s, A)$, где Q — множество состояний автомата, Σ — алфавит, $\delta : Q \times \Sigma \to Q$ — полная функция переходов, $s \in Q$ — стартовое состояние, $A \subset Q$ — множество допускающих состояний. Число состояний ДКА $|Q| = n$ называют его размером. ДКА удобно представлять в виде ориентированного графа, где вершины графа, это состояния автомата, а ребра — переходы между состояниями по символу алфавита. Соответственно, некоторые вершины графа являются допускающими. Также пронумеруем используемый алфавит следующим образом: $\{l_1, \ldots, l_{|\Sigma|}\} = \Sigma$. Пример ДКА изображен на рисунке 1.

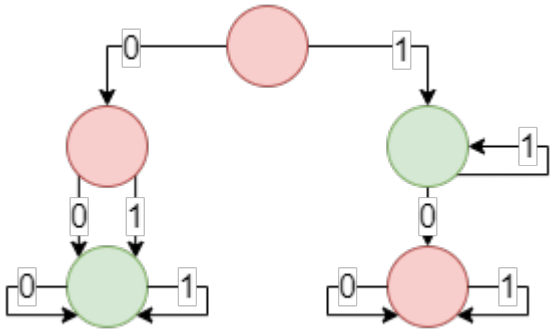


Рисунок 1 – Пример ДКА

Автомат допускает слово w — конечную последовательность символов алфавита Σ в следующем случае. Изначально автомат находится в стартовом состоянии. Далее автомат считывает последовательно символы слова и делает переходы между состояниями в соответствии с функцией переходов δ. Если по достижении конца слова было достигну допускающее состояние, то автомат допускает слово.

Примеры поведения это пара конечных непересекающихся множеств (словарей) $S = (S^A, S^R)$: слова, допускаемые автоматом, и слова, отклоняемые автоматом соответственно. Пример словарей, соответствующих ДКА и рисунку 1 изображен на рисунке 2.

Accept
00
01
1
11
111

Reject
0
101
1010
1101
1110

Рисунок 2 – Пример словарей поведения

Задача построения ДКА по примерам поведения заключается в нахождении минимального по числу состояний детерминированного конечного автомата, допускающего слова из словаря S^A и отклоняющего слова из S^R. Поведение с любыми, другими словами, не определено, то есть может быть любым.

Раскрашенное префиксное дерево представляет из себя стандартное префиксное дерево у которого каждая вершина раскрашена в определенный цвет в зависимости от того, в каком словаре находится слово, соответствующее вершине. Определим раскрашенное префиксное дерево следующим образом:

$(T, \Sigma, \xi, t, F^A, F^R)$, где T — множество состояний префиксного дерева, Σ — алфавит, $\xi : T \times \Sigma \to T$ — функция переходов, $t \in T$ — стартовое состояние, $F^A \subset T$ — множество допускающих состояний, $F^R \subset T$ — множество не допускающих состояний. $F^A \cap F^R = \emptyset$. Соответственно, будем считать, что вершина раскрашена в зеленый цвет, если является допускающей, в красный, если не допускающей, а иначе белой. Пример префиксного дерева, построенного по словарям из рисунка 2, изображен на рисунке 3.

1.2. Методы решения задачи построения ДКА

В [10] было показано, что задача нахождения ДКА с ограничением сверху на число состояний является NP-полной задачей, а в [14] доказана невозможность приближенного решения за полиномиальное время. Тем не менее

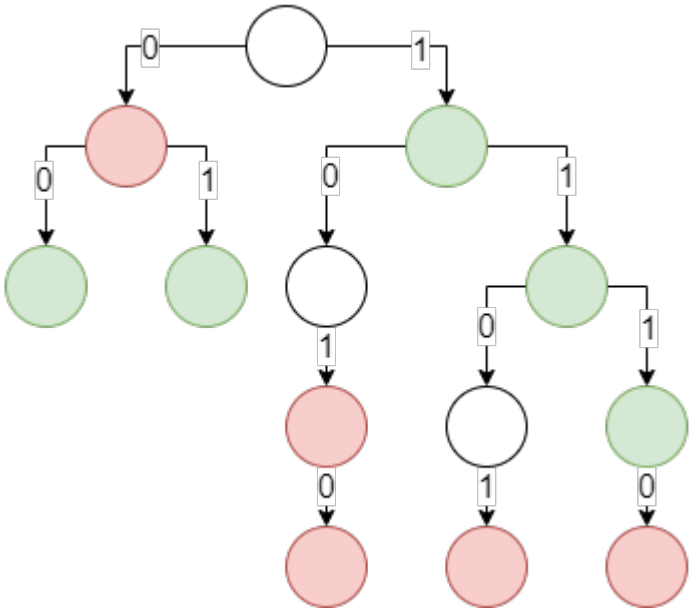


Рисунок 3 – Пример раскрашенного префиксного дерева

существует несколько подходов к решению задачи, которые позволяют с той или иной точностью находить искомый ДКА по примерам поведения.

Эвристические алгоритмы основаны на идее слияния состояний префиксного дерева. Алгоритмы, использующие данный подход строят префиксное дерево по исходным словарям и затем последовательно жадно объединяют состояния до тех пор, пока это возможно.

Метавристические алгоритмы основаны на эволюционных, генетических или муравьиных алгоритмах. Задача генерации ДКА по примерам поведения в данном случае сводится к задаче оптимизации. Недостатком метаэвристических алгоритмов является их неточность. Алгоритмы не гарантируют нахождение какого-либо ответа за конечное время.

Недостатком описанных выше подходов является тот факт, что они является приближенными и не всегда находят минимальное решение, потому в ряде случаев задачу построения ДКА по примерам поведения решают путем сведения к другим NP-полным задачам. В частности, к задаче булевой выполнимости SAT. Данный подход позволяет использовать при решении задачи SAT-решатели (SAT-solver) - программные средства, предназначенные для решения задачи булевой выполнимости.

1.3. SAT-решатели

SAT-решатели — это специальные программные средства, позволяющие решать задачу булевой выполнимости (SAT). В основе многих современных

SAT-решателей лежит алгоритм CDCL, который в свою очередь основан на алгоритме DPLL [5]. Помимо базового алгоритма в решателях используется множество различных эвристик, помогающих быстрее решать задачу. Существует соревнование SAT Competition, на котором в разных номинациях сравнивают SAT-решатели по числу решенных задач и затраченному времени. Разнообразие SAT-решателей позволяет выбирать среди них наиболее успешные для решения поставленой задачи.

1.4. Сведение к задаче булевой выполнимости

Так как задача построения ДКА по примерам поведения является NP-полной, она может быть сведена к задаче выполнимости булевых формул. Ввиду существования соревнования SAT все SAT-решатели имеют общий стандарт входных данных. Таким образом достаточно реализовать сведение задачи построения ДКА к SAT, а затем возможно использовать любой подходящий SAT-решатель. Основным преимуществом данного подхода является его точность. Алгоритм точно определяет существование искомого ДКА, а также находит минимальный автомат.

Процесс сведения задачи построения ДКА по примерам поведения к задаче булевой выполнимости начинается с построения раскрашенного префиксного дерева по двум словарям поведения. Процесс построения идентичен построению обычного префиксного дерева за исключением того, что вершины, соответствующие концам допускающих слов, раскрашиваем в зеленый, допускающий цвет, вершины, соответствующие не допускающим словам в красный, а все остальные оставляем белыми — не раскрашенными.

Пронумеруем вершины ДКА целыми числами $B = \{0, \ldots, n-1\}$. Для каждого состояния $q \in T$ введем переменную $c_q \in B$, задаваемую следующим образом:

$c_0 = 0$

$\forall q, p \in T$

если $q \in F^A \cap p \in F^R$ то $c_q \neq c_p$

если $\exists a \in \Sigma : (q, a, q'), (p, a, p') \in \delta$ то $(c_q = c_p) \Rightarrow (c_{q'} = c_{p'})$

Такую переменную стоит воспринимать как то, что в процессе построения ДКА вершина q префиксного дерева переходит в вершину c_q ДКА, сохраняя все переходы. Таким образом задача сводится к поиску таких значений c, чтобы они удовлетворяли условию выше.

Эти формулы могут быть преобразованы в SAT с использованием переменной $x_{q,i}, q \in T, i \in Q : x_{q,i} = True \Rightarrow c_q = i$. Что означает, что в процессе решения задачи вершина *q* префиксного дерева перейдет в вершину *i* ДКА. Ученые Хойл и Вервер предложили использовать набор вспомогательных переменных и избыточных предикатов для ускорения процесса решения [11]. Рассмотрим предложенное ими кодирование.

В дополнение к основной переменной $x_{q,i}$ вводят две вспомогательные:

— $y_{a,i,j}, a \in Sigma, i, j \in Q : y_{a,i,j} = True \Rightarrow (i, j, a) \in \delta$

В ДКА переход из вершины *i* по символу *a* ведет в вершину *j*.

— $z_i, i \in Q : z_i = True \Rightarrow i \in A$

Состояние *i* ДКА является допускающим.

Данные переменные помогают уменьшить число выводов из основных, упрощая доступ к информации об автомате. Далее используются следующие предикаты:

— $\forall q \in T, x_{q,0} \cup x_{q,1} \cup \ldots \cup x_{q,n-1}$

Каждая вершина префиксного дерева должна попасть хотя бы в одну вершину ДКА.

— $\forall q \in T, \forall i, j \in Q : i \neq j, \neg x_{q,i} \cup \neg x_{q,j}$

Каждая вершина префиксного дерева должна попасть максимум в одну вершину ДКА.

— $\forall q \in F^A, \forall p \in F^R, \forall i \in Q, (\neg x_{q,i} \cup z_i) \cap (\neg x_{p,i} \cup \neg z_i)$

Допускающая вершина префиксного дерева не может попасть в одну вершину ДКА с не допускающей.

— $\forall (q, a, p) \in \xi, \forall i, j \in Q, y_{a,i,j} \cup \neg x_{q,i} \cup \neg x_{p,j}$

Если вершина *q* префиксного дерева попала в вершину *i* ДКА, а её потомок по символу *a* попал в вершину *j*, то переход из вершины *i* ДКА по символу *a* ведет в вершину *j*.

— $\forall (q, a, p) \in \delta, \forall i, j \in Q, \neg y_{a,i,j} \cup \neg x_{q,i} \cup x_{p,j}$

Если переход из вершины *i* ДКА по символу *a* ведет в вершину *j*, а вершина *q* префиксного дерева попадает в вершину *i*, то её потомок по символу *a* попадет в вершину *j*.

— $\forall a \in \Sigma, \forall i, j, h \in Q : h < j, \neg y_{a,i,h} \cup \neg y_{a,i,j}$

Переход из каждой вершины ДКА по любому символу ведет в максимум одну вершину.

— $\forall a \in \Sigma, \forall i \in Q, y_{a,i,0} \cup y_{a,i,1} \cup \ldots \cup y_{a,i,n-1}$

Переход из каждой вершины ДКА по любому символу ведет хотя бы в одну вершину.

1.5. Рассматриваемые в исследовании эвристики

Основную сложность в рассматриваемой задаче составляет создание булевой формулы, сводящей задачу построения ДКА к задаче SAT. Важными параметрами, влияющими на скорость работы алгоритма являются число и длинна предикатов, входящих в булеву формулу и количество используемых переменных. Помимо подобных численных факторов значительную роль могут играть дополнительные оптимизации и эвристики, которые могут быть реализованы как избыточные предикаты в булевой формуле или как параметры SAT-решателя. В данной работе рассматриваются такие эвристики как предикаты нарушения симметрии, призванные сократить число, рассматриваемых SAT-решателем автоматов, а также приоритеты переменных, ручной выбор которых при хорошем сочетании с предикатами нарушения симметрии позволяет значительно сократить время решения задачи. Также в данной работе ведется исследование эвристики инкрементального решения задачи генерации ДКА, которая предложена в [4].

1.5.1. Предикаты нарушения симметрии

Если не задавать ограничения на нумерацию вершина ДКА, то для искомого автомата существует $|S|!$ изоморфных автоматов. Хойл и Вервер в своей работе предложили способ нарушения симметрии, позволяющий рассматривать меньше изоморфных автоматов. Способ сводится к поиску максимальной клики в графе несовместимости. Но так как эта задача является NP-полной, было предложено искать не максимальную, а достаточно большую клику. Данный подход позволяет рассматривать меньше ДКА. Если была найдена клика размера k, то всего будет рассмотрено $(|S| - - - k)!$ автоматов, что с одной стороны в силу стремительного роста факториала сильно меньше, чем $|S|!$, но все еще может быть достаточно большим по той же причине. Более того, способ уменьшения симметрии с помощью поиска клики невозможно применять при использовании ряда эвристик и при решении некоторых модификаций задачи. Например, при решении задачи построения ДКА с зашумленными данными [16]. Существуют предикаты нарушения симметрии, которые являются

более эффективными, чем клика [3, 7, 15], но наиболее эффективными можно считать идеальное нарушение симметрии на основе алгоритма поиска в ширину.

1.5.2. Предикаты нарушения симметрии на основе алгоритма поиска в ширину

Идеальным нарушением симметрии является такое, когда для каждого класса эквивалентности ДКА остается единственный представитель относительно симметрии. В работе [16] предлагаются идеальные предикаты нарушения симметрии на основе алгоритма обхода в ширину. Идея предикатов заключается в том, чтобы зафиксировать нумерацию ДКА в порядке обхода в ширину. Для этого устанавливаются следующие правила на нумерацию автоматов:

— Каждая вершине ДКА имеет уникальный номер от 1 до $|Q|$.
— Дети каждой вершины упорядочены в лексикографическом порядке символов перехода
— На одной глубине номера вершин увеличиваются слева-направо.
— Номера вершин увеличиваются сверху-вниз

Вводятся следующие переменные:

— $p_{j,i}, 1 \leqslant i < j \leqslant |Q|$

Вершина *i* ДКА является родителем вершины *j*.

— $t_{i,j}, 1 \leqslant i < j \leqslant |Q|$

Существует переход из вершины *i* ДКА в вершину *j*.

— $t_{i,l,j}, 1 \leqslant i < j \leqslant |Q|, l \in \Sigma$

Существует переход из вершины *i* ДКА в вершину *j* по символу *l*, но не существует переходов по меньшим символам.

Далее вводятся следующие предикаты:

— $t_{i,j} \Rightarrow (y_{i,l_1,j} \cup \ldots \cup y_{i,l_{|\Sigma|},j}), 1 \leqslant i < j \leqslant |Q|$
— $y_{i,l,j} \Rightarrow t_{i,j}, 1 \leqslant i < j \leqslant |Q|, l \in \Sigma$
— $p_{j,i} \Rightarrow t_{i,j}, 1 \leqslant i < j \leqslant |Q|$
— $p_{j,1} \cup p_{j,2} \cup \ldots \cup p_{j,j-1}, 2 \leqslant j \leqslant |Q|$
— $m_{i,l,j} \Rightarrow y_{i,l,j}, 1 \leqslant i < j \leqslant |Q|, l \in \Sigma$
— $m_{i,l_n,j} \Rightarrow \neg y_{i,l_k,j}, 1 \leqslant i < j \leqslant |Q|, 1 \leqslant k < n \leqslant |\Sigma|$
— $(y_{i,l_n,j} \cap \neg y_{i,l_{n-1},j} \cap \ldots \cap \neg y_{i,l_1,j}) \Rightarrow m_{i,l_n,j}, 1 \leqslant i < j \leqslant |Q|, 1 \leqslant n \leqslant |\Sigma|$
— $p_{j,i} \Rightarrow \neg t_{k,j}, 1 \leqslant k < i < j \leqslant |Q|$

— $(t_{i,j} \cap \neg t_{i-1,j} \cap \ldots \cap \neg t_{1,j}) \Rightarrow p_{j,i}, 1 \leqslant i < j \leqslant |Q|$
— $p_{j,i} \Rightarrow \neg p_{j+1,k}, 1 \leqslant k < i < j \leqslant |Q|$
— $(p_{j,i} \cap p_{j+1,i} \cap m_{i,l_n,j}) \Rightarrow \neg m_{i,l_m,j+1}, 1 \leqslant i < j \leqslant |Q|, 1 \leqslant m < n \leqslant |\Sigma|$

1.5.3. Приоритет и активность переменных

Производительность SAT-решателей сильно зависит от выборов, которые они делают, назначая следующую свободную переменную. Свободные переменные — это переменные, которые еще не были назначены решателем как истинные или ложные. На каждой итерации решатель выбирает свободную переменную в соответствии с её активностью и назначает её как истину. Время решения задачи может быть значительно уменьшено при удачном выборе переменной. Для решения данной задачи решатели используют различные эвристики. Данные эвристики являются универсальными и пытаются уменьшить время решения независимо от того, какую именно задачу решает SAT-решатель [8, 9, 12].

Активность переменной — некоторая величина, показывающая то, насколько более приоритетной является переменная для следующего назначения. В частности в современных SAT-решателях в качестве активности переменной используется вещественное число. Чем больше это число, тем более приоритетной переменная является для назначения. Активность имеет свойство уменьшаться со временем.

Приоритет переменной — также величина, показывающая то, насколько более приоритетной является переменная для назначения. Но в отличие от активности приоритет не изменяется со временем.

Таким образом при решении задачи построения ДКА по примерам поведения появляется возможность помочь решателю сделать правильный выбор путем ручной задачи активности или приоритета переменных. В силу того, что решается конкретная задача, правильное задание активности или приоритета способно значительно уменьшить время решения задачи.

Переменные решателя подразделяются на два типа: *Основные* и *вспомогательные*. *Основные* переменные несут в себе главную информацию о ДКА. По этим переменным можно восстановить всю структуру автомата и, соответственно, значения всех остальных переменных. *Вспомогательные* переменные призваны помочь решателю быстрее решить задачу. Они могут как помо-

гать делать правильные выводы, так и наоборот, представлять из себя условия несуществования ДКА.

1.6. Обзор эвристик, основанных на алгоритме RPNI

В своей работе канадские исследователи Флорент и Александр предлагают эвристики, вдохновленные алгоритмом RPNI [13].

Алгоритм RPNI также решает задачу генерации ДКА по примерам поведения. Он заключается в том, что префиксное дерево принимается как начальная гипотеза. Далее алгоритм попарно перебирает вершины текущей гипотезы в лексикографическом порядке. Для каждой пары алгоритм проводит проверку на то, возможно ли объединить выбранные вершины и, если такое возможно, объединяет вершины, формируя новую гипотезу. Алгоритм перебирает и пытается объединять вершины до тех пор, пока такое возможно. Конечный автомат считается искомым ДКА. Проблемой алгоритма является его асимптотика, которая значительно зависит от размера префиксного дерева.

В первую очередь авторы определяют кодирование префиксного дерева. Так как нарушение симметрии предложенное Хойлом и Вервером неприменимо вместе с предлагаемыми в статье эвристиками, авторы предлагают собственные предикаты нарушения симметрии.

$$\forall q \in T, \forall i \in Q, (\bigvee_{q'<q} \neg x_{q',i}) \Rightarrow \neg x_{q,i+1}$$

Формально, данный предикат побуждает решатель выбирать вершину ДКА с наименьшим возможным номером, когда пытается найти пару к вершине префиксного дерева.

Также авторы предлагают ввести дополнительные предикаты, призванные уменьшить время решения задачи в случае, если искомого ДКА не существует.

В первую очередь вводится новая переменная $\forall q, p \in T, E_{q,p}$, которая означает, что состояния префиксного дерева q и p попадут в одну и ту же вершину ДКА.

Далее вводится три вспомогательных предиката:

— $\forall q \in F^A, \forall p \in F^R, \neg E_{q,p}$

Допускающая вершина префиксного дерева не может попасть в одну вершину ДКА с не допускающей. Данный предикат повторяет предикат из кодирования Хойла и Вервера, но делает это в более явном виде.

— $\forall (q, a, p), (q', a, p') \in \xi, E_{q,q'} \Rightarrow E_{p,p'}$

Если две вершины префиксного дерева попали в одну вершину ДКА, то и их потомки по каждому символу будут попадать в одну вершину ДКА.

— $\forall q, p \in T, i \in \{0, \ldots, n-1\}, (\neg E_{q,p} \cap x_{q,i}) \Rightarrow \neg x_{p,i}$

Данный предикат запрещает попадать в одну и ту же вершину ДКА тем вершинам префиксного дерева, про которые известно, что их нельзя объединять.

1.6.1. Инкрементальное решение задачи построения ДКА по примерам поведения

Первая эвристика, предложенная Флорентом и Александром, это инкрементальное решение задачи построения ДКА по примерам поведения. При большом размере словарей в SAT-решатель кодируется большое число предикатов, что влечет долгое время решения задачи. Тем временем авторы пытаются решать задачу при как можно большем размере словарей. В статье предлагается генерировать предикаты инкрементально, пытаясь решить задачу SAT на каждой итерации.

Алгоритм устроен следующим образом: предположим, что мы решаем задачу при фиксированном $|Q|$. Пусть π — некоторый набор слов, который кодируется и подается на вход в SAT-решатель, а Π — ДКА, соответствующий данному набору слов. Изначально $\pi = \emptyset$. Алгоритм выбирает лексикографически наименьшее слово из S, такое что его подстановка в Π, дает результат отличный от того, в каком словаре находится слово. Это слово добавляется в π и происходит попытка решения задачи SAT на множестве π. Далее снова идет выбор нового слова для добавления в π. Алгоритм повторяется до тех пор, пока либо не получит автомат Π, который корректно распознает все слова из S, в таком случае он и является искомым, либо пока SAT-решатель не даст отрицательный ответ, в таком случае искомого автомата не существует.

1.6.2. Эвристика предметной области

Вторая эвристика, предложенная Флорентом и Александром основана на свойстве SAT-решателей использовать приоритеты и активность переменной. Вдохновляясь алгоритмом RPNI [13], они предлагают отключить эвристики решателя, связанные с приоритетами и активностью и задавать приоритет переменных вручную.

В SAT-формуле, предложенной в статье, переменные $x_{q,i}$ назначаются основными, все остальные — вспомогательными. Эвристика используется для определения следующей основной переменной, по которой решатель должен будет сделать предположение. Для этого каждой переменной $x_{q,i}$ назначается соответствующее ей слово w префиксного дерева. Решатель делает выбор следующей переменной $x_{q',i'}$ для назначения в соответствии с лексикографическим порядком слов, соответствующих переменным. Каждую выбранную переменную решатель будет предполагать, как истинную.

ГЛАВА 2. ИСПОЛЬЗОВАНИЕ И МОДИФИКАЦИЯ ЭВРИСТИК ПРИОРИТЕТА И ИНКРЕМЕНТАЛЬНОГО РЕШЕНИЯ ЗАДАЧИ ПОСТРОЕНИЯ ДКА ПО ПРИМЕРАМ ПОВЕДЕНИЯ

В рамках исследования было проведено сравнение оригинальных предикатов нарушения симметрии, предложенных в [4], а также предикаты нарушения симметрии, основанных на поиске в ширину, предложенных в [16]. Результаты исследования показали, что предикаты нарушения симметрии, основанные на поиске в ширину, более эффективны. В связи с этим было принято решение далее использовать именно их.

В рамках исследования был рассмотрен случай одновременного использования предикатов нарушения симметрии предложенных в [4] и в [16].

2.1. Приоритеты и активность переменных предикатов нарушения симметрии, основанных на обходе в ширину

В процессе работы решатель активно использует информацию об активности переменных. Основываясь на активности решатель выбирает следующие переменные для назначения, а также использует величину активности для, принимая решение о том, какие предикаты использовать для дальнейших выводов. Рассмотрим использование активности на примере решателя *MiniSat* [2]. Данный решатель представляет из себя классику SAT-решателей: простой, но эффективный. Он прекрасно подходит для понимания принципов работы SAT-решателей и их основных эвристик.

Для реализации эвристики активности переменных решатель использует упорядоченную очередь пар, состоящих из переменной и вещественного числа, её активности, а также компаратор, представляющий из себя порядок данной очереди. В стандартной реализации SAT-решателя начальная активность каждой переменной является случайной величиной. На каждой итерации поиска решения решатель выбирает более приоритетную вершину, то есть обладающую наибольшей активностью и назначает её как истинную. После этого он уменьшает переменную угасания активности на фиксированную величину. Далее, если в процессе выводов решатель находит конфликтный предикат, то все переменные, которые в нем участвуют, умножают свою активность на переменную угасания. Так как она находится в промежутке между нулем и единицей, то по итогу активность переменных уменьшается. Когда перемен-

ная угасания становится слишком близкой к нулю, решатель пропорционально увеличивает все активности.

Данная эвристика активности переменной является новой в сфере решения задачи построения ДКА по примерам поведения. Ранее ни в одной работе не рассматривался случай модификации активности для решения конкретной задачи построения ДКА. Авторы [4] вводят новое понятие приоритета переменной, которое отличается от активности тем, что не изменяется со временем и не влияет на принятие решений за исключением выбора следующей переменной для назначения.

Приоритеты переменных, предложенные в [4], побуждают решатель в первую очередь назначать основные переменные $x_{q,i}$. Для этого авторы модифицируют SAT-решатель, убирая возможность как-либо использовать активность, кроме выбора следующей переменной для назначения. В качестве величины активности они используют список целых чисел. В список поочередно добавляют величины, соответствующие символам слова вершины *q* после которых размещают число *i*. Компаратором упорядоченной очереди решателя используется лексикографический порядок. Если найти значения всех основных переменных, появляется возможность вывести значения вспомогательных переменных, а также искомый ДКА.

Число переменных $x_{q,i}$ равняется $|T||Q|$, что является достаточно большим числом при увеличение размера словарей и ДКА. С другой стороны число переменных $p_{j,i}$ равняется $|Q|^2$, что значительно меньше, учитывая тот факт, что число примеров значительно превосходит число состояний ДКА. Несмотря на то, что зная значение этих переменных можно только восстановить структуру ДКА, но не его метки, решателю в такой ситуации может быть значительно проще найти правильный ответ, чем перебирать $|T||Q|$ переменных. В связи с этим было принято решение исследовать использование приоритетов и активности переменных предикатов нарушения симметрии, основанных на обходе в ширину, а в частности рассматривать приоритеты и активность переменной $p_{j,i}$.

Приоритетом переменной $p_{j,i}$ в отличие от $x_{q,i}$ будем считать вещественное число. Чем больше величина этого числа, тем более предпочтительной для назначения решателем является переменная. Переменные были упорядочены в BFS-порядке, то есть сверху-вниз, слева-направо. Каждой был назначен по-

рядковый номер равный $n \cdot i + j$, где $n = |Q|$. Так как, чем больше активность, тем быстрее она уменьшается, при дальнейшем использовании эвристик следует избегать слишком больших значений. Для этого было принято решение устанавливать активность переменных в промежутке между 0 и 10. Далее были разработаны следующий эвристики:

— Упорядочить переменные $p_{j,i}$ в возрастающем порядке по значению $n \cdot i + j$. Каждой переменной было сопоставлено значение $\frac{(n \cdot i + j) \cdot 10}{n^2}$. Данное значение использовать как активность решателя.

— Упорядочить переменные $p_{j,i}$ в убывающем порядке по значению $n \cdot i + j$. Каждой переменной было сопоставлено значение $\frac{(n \cdot i + j) \cdot 10}{n^2}$. Данное значение использовать как приоритет решателя.

— Упорядочить переменные $p_{j,i}$ в возрастающем порядке по значению $n \cdot i + j$. Каждой переменной было сопоставлено значение $\frac{(n^2 - (n \cdot i + j)) \cdot 10}{n^2}$. Данное значение использовать как активность решателя.

— Упорядочить переменные $p_{j,i}$ в убывающем порядке по значению $n \cdot i + j$. Каждой переменной было сопоставлено значение $\frac{(n^2 - (n \cdot i + j)) \cdot 10}{n^2}$. Данное значение использовать как приоритет решателя.

2.2. Эвристика инкрементального решения задачи

Первая часть исследования была посвящена изучению и модификации эвристики инкрементального решения задачи генерации ДКА по примерам поведения. После изучения эвристики, предложенной в [4], были сделаны выводы о том, что её основным недостатком является тот факт, что добавление слов в SAT-решатель начинается с нуля и по одному. В связи с этим было выбрано два направления исследования эвристики.

Первое направление эвристик — эвристики начального набора примеров. Решение, предложенное в [4] предполагает, что изначально в решателе отсутствуют примеры, а далее они добавляются по одному и для каждого решается задача SAT. Очевидно, что таким образом происходит много лишних попыток решить задачу, потому предлагается создать некоторый стартовый набор примеров и добавить его в SAT-решатель в самом начале исполнения. В исследовании рассматривались следующие эвристики:

— Добавить *r* случайных слов из объединенных словарей $S^A \cup S^R$.

Мы можем попытаться угадать хороший набор слов и сразу дать его решателю.

— Добавить r лексикографически первых слов из объединенных словарей $S^A \cup S^R$.

Так как алгоритм добавляет слова в лексикографическом порядке, то добавление этих слов сразу может помочь ему решить задачу быстрее.

Выбор числа r для каждого случая, также был частью исследования и опирался на среднее число примеров, добавленных в SAT-решатель, достаточных для получения искомого ДКА.

Второе направление эвристик также призвано уменьшить число запусков SAT-решателя. Идея эвристик заключается в том, чтобы на каждой итерации добавлять не одно, а несколько слов в решатель. Рассматривались следующие эвристики:

— Добавлять не только лексикографически наименьшее слово, а также случайно выбранное.

Добавление случайного слова может помочь алгоритму быстрее понять структуру автомата.

— Помимо лексикографически наименьшего слова добавлять его поддерево фиксированного размера r. Для этого выбирается последняя вершина слова. Пока размер поддерева выбранной вершины меньше r, выбирается предок этой вершины, если размер его поддерева также меньше или равен r.

Использование сразу множества лексикографически близких слов уменьшит число запусков решателя.

2.2.1. Эвристика инкрементального решения задачи при неизвестном размере автомата

Эвристики, предлагаемые в [4] предполагается использовать при решении задачи генерации ДКА при известном размере автомата. Это значит, что гарантируется существование автомата и на каждой итерации добавления примеров решатель будет находить некоторый автомат. Имеет смысл также рассмотреть решение задачи генерации ДКА при неизвестном размере автомата. В данной случае широко используется следующее решение.

Выбирается стартовый размер ДКА n, как правило $n = 1$. Далее происходит попытка решения задачи построения ДКА по примерам поведения при

известном размере автомата $|Q| = n$. Если решение найдено, то полученный ДКА размера *n* и является искомым. Иначе *n* увеличивают на 1, полностью перезапускают SAT-решатель и снова пытаются решить задачу при известном размере автомата. Алгоритм заканчивается либо, когда найдет искомый ДКА при некотором *n*, либо когда превысит ограничение по времени.

Принципиальное отличие задач известного и неизвестного размера автомата заключается в том, что если мы пытаемся решить задачу генерации ДКА при правильном размере, то, если не допущены ошибки в предикатах, рано или поздно решатель найдет некоторое решение. С другой стороны, при попытке решить задачу с неправильным размером ДКА, решения найдено не будет, что в ряде случаев может занять значительно больше времени, чем при существующем решении.

Для решения задачи построения ДКА по примерам поведения при неизвестном размере автомата была разработана следующая эвристика, призванная уменьшить число запусков SAT-решателя в процессе решения задачи.

Вместо полного перезапуска SAT-решателя будем запоминать набор слов, на котором алгоритм остановился на предыдущей итерации. Если убрать последнее слово, добавление которого привело к тому, что алгоритм доказал отсутствие ДКА, получим набор слов, который гарантировано даст некоторый ДКА при новом *n*, большем на единицу. Так как данный набор слов был выбран на предыдущей итерации алгоритма, то он с высокой вероятностью является хорошим и способен привести к искомому ДКА при правильном выборе размера автомата. Суть эвристики заключается в том, чтобы при решении задачи построения ДКА на каждой итерации запоминать последний успешный набор слов. То есть такой набор, что решатель для него нашел некоторый ДКА. Далее при увеличении размера автомата сразу добавлять этот набор слов в решатель, а не начинать с пустого набора. Алгоритм эвристики изображен на листинге 1. Запускаемой функцией является *SolveByUnknownN*.

Листинг 1 – Алгоритм решения задачи при неизвестном размере автомата с эвристикой запоминания

```
function SolveByN(N, S, start)
    a = emptyDFA
    a.add(start)
    while a is not consistent with S do
        for word ∈ sorted(S) do
            if word is not consistent with a then
                a.add(word)
                result = a.solve()
                if result = null then
                    return {null, start}
                end if
                start.push(word)
                continue
            end if
        end for
        return {null, start}
    end while
    return {a, start}
end function
function SolveByUnknownN(MaxN, S)
    start = ∅
    for n ← [1; MaxN] do
        result = SolveByN(n, S, start)
        if result.first ≠ null then
            return result.first
        end if
        start = result.second
    end for
    return null
end function
```

ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

В данной главе приведены результаты экспериментального исследования.

3.1. Процесс тестирования

Эксперименты проводились на персональном компьютере с процессором *Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz* на операционной системе *Ubuntu 20.04 on Windows*. Алгоритмы были реализованы на языке *C++*, а окружение для тестирования на языке *Python*. В рамках исследования от авторов статьи [4] был получен код, используемый ученными в их работе. Внешняя оболочка алгоритма, а также процесс сбора логов были доработаны для взаимодействия с тестирующим окружением. Сравнение эвристик проводилось при следующих параметрах:

— $\Sigma = \{0, 1\}$. Данное значение было выбрано с целью достижения наибольшего числа состояний ДКА $|Q|$ при тестировании, так как именно этот параметр сильнее всего влияет на время решения задачи, и стремится следует к его максимизации.

— $|S| = 10000$. Эвристики, предложенные в [4], лучше всего себя показывают при большом размере словарей. Более того, решение задачи при большом размере словарей больше нагружает решатель, чем при малом.

— Каждый эксперимент повторялся $t = 50$.

— Ограничение по времени работы алгоритма было установлено в 1200 секунд. Данного значения достаточно, чтобы оценить время работы эвристик относительно друг друга при данных ограничениях.

Для каждого $|Q| \in \{1 \dots 100\}$ t раз генерируется случайный тест с текущим $|Q|$. Производится попытка решить задачу на данном тесте. Если алгоритм превышает установленное максимальное время решения, считается, что он задачу не решил. Все изображенные графики обрезаны сверху, так как максимально достигаемые значения времени работы негативно влияют на масштаб графика и делают его трудно читаемым.

3.2. Используемые SAT-решатели

В оригинальном алгоритме, реализованном в [4] в качестве SAT-решателя используется *MiniSat* [2]. Это минималистичный решатель с открытым кодом предназначенный для исследователей, и разработчиков, знакомящихся с SAT-решателями. Преимуществом решателя является простая и

понятная структура кода и отсутствие огромного нагромождения эвристик. Несмотря на простоту, решатель является победителем в ряде номинаций на *SAT 2005 competition*. Помимо *MiniSat* используемые эвристики были реализованы с помощью SAT-решателя *Glucose 4.1* [1]. Основным преимуществом данного решателя является то, что он реализован на основе *MiniSat*, что облегчает работу с его модификацией, но в нем, в отличие от *MiniSat* реализован ряд эвристик, которые делают решатель более эффективным, чем *MiniSat*. Данный решатель является победителем или призером *SAT competition* в 2009, 2011, 2013, 2014 и 2015 годах. В *Glucose 4.1* в добавок к активности переменных были реализованы приоритеты переменных.

3.3. Алгоритм генерации данных

В работе использовался алгоритм генерации данных, предложенный в [4]. Алгоритм генерирует примеры поведения для автомата размера $|Q|$, алфавитом Σ, суммарным размером словарей $|S|$ и максимальной длинной слова l.

В первую очередь алгоритм генерирует минимальный ДКА размера $|Q|$ и алфавитом Σ. Для этого создается новый случайный ДКА указанных размеров до тех пор, пока не будет получен минимальный ДКА. После этого от корня автомата $|S|$ раз берется случайный путь длинны от 1 до l. В зависимости от типа финального состояния слово, соответствующее выбранному пути относят либо в допускающий, либо в не допускающий словарь.

3.4. Сравнение предикатов нарушения симметрии

Помимо оригинальных предикатов нарушения симметрии, предложенных в [4] были реализованы предикаты нарушения симметрии на основе алгоритма поиска в ширину из [16]. Далее при указанных выше ограничениях было проведено сравнение предикатов нарушения симметрии из [4], предикатов нарушения симметрии на основе алгоритма поиска в ширину, а также случая, использующего оба типа предикатов. Результаты тестирования показаны на рисунке 4.

— Оригинальные предикаты нарушения симметрии, предложенные в [4], изображены на графике как *french predicates*.
— Предикаты нарушения симметрии, основанные на алгоритме поиска в ширину, изображены на графике как *bfs predicates*.

— Подход использующий оба типа предикатов изображен как *both predicates*.

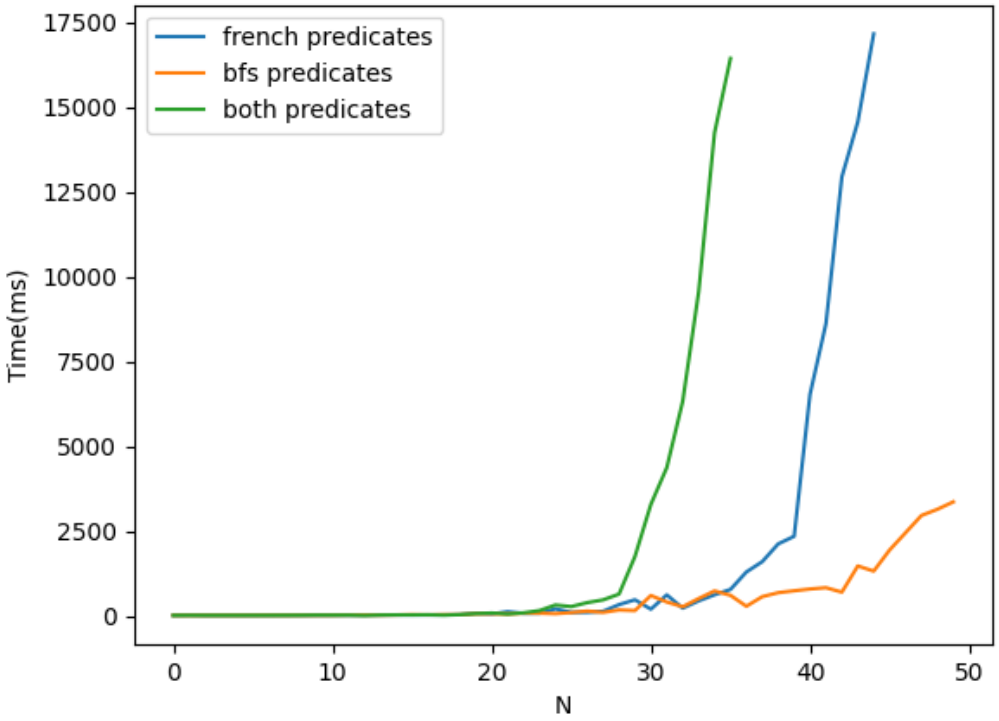


Рисунок 4 – Среднее время работы алгоритма с различными предикатами нарушения симметрии

Результаты показывают, что использование предикатов нарушения симметрии на основе алгоритма поиска в ширину уменьшает время работы алгоритма по сравнению с предикатами, предложенными в [4]. В то же время одновременное использование предикатов нарушения симметрии из [4] и основанных на алгоритме поиска в ширину увеличивает время исполнения программы. Подход, основанный на одновременном использовании предикатов перестает укладываться по времени с $|Q| = 27$, а с $|Q| = 38$ вовсе не успевает решить ни одной задачи. В то же время подход, использующий предикаты нарушения на основе алгоритма поиска в ширину начиная с $|Q| = 35$ показывает лучшие результаты, чем предикаты из [4]. Начиная с этого значения графики начинают расходиться.

3.5. Приоритеты и активность переменных

Было проведено тестирование и сравнение приоритетов и активностей переменных, описаных в пункте 2.2. Для этого используемый SAT-решатель

был модифицирован с целью поддержки приоритетов переменных. Сравнивалось использовании переменных $p_{j,i}$ в качестве приоритета или активности решателя в прямом и обратном порядке. Также был рассмотрен случай использования стандартной активности переменных, реализованной в SAT-решателе. Результаты тестирования показаны на рисунке 5.

— Оригинальные приоритеты, предложенные в [4], изображены на графике как *lexicographical priority*.
— Приоритет переменных основанный на алгоритме поиска в ширину и отсортированный в возрастающем порядке изображен на графике как *bfs priority up*.
— Приоритет переменных основанный на алгоритме поиска в ширину и отсортированный в возрастающем порядке изображен на графике как *bfs priority down*.
— Активность переменных основанная на алгоритме поиска в ширину и отсортированныя в возрастающем порядке изображена на графике как *bfs activity up*.
— Активность переменных основанная на алгоритме поиска в ширину и отсортированныя в возрастающем порядке изображена на графике как *bfs activity down*.
— Стандартная активность переменных реализованная в SAT-решателе изображена на графике как *default activity*.

Результаты показывают, что использование переменных $p_{j,i}$ в качестве приоритета решателя не эффективно, как и использование стандартной активности решателей. В то-же время использование переменных $p_{j,i}$ как активность решателя уменьшает время работы программы по сравнению с лексикографическим приоритетом переменных $x_{q,i}$, используемым в [4]. Приоритеты переменной $p_{j,i}$, а также стандартная активность решателя значительно замеляются в работе после $|Q| = 30$. Начиная с $|Q| = 45$ использование переменной $p_{j,i}$, как активности решателя показывает меньшее время работы, чем использование лексикографических приоритетов переменной $x_{q,i}$. Также, с этого момента графики начинают расходиться. Как видно на графиках, использование прямого BFS порядка на вершинах ДКА немного более эффективно, чем обратного. Также видно, что использование эвристик приоритета и активности переменных действительно позволяет уменьшить время работы SAT-решателя.

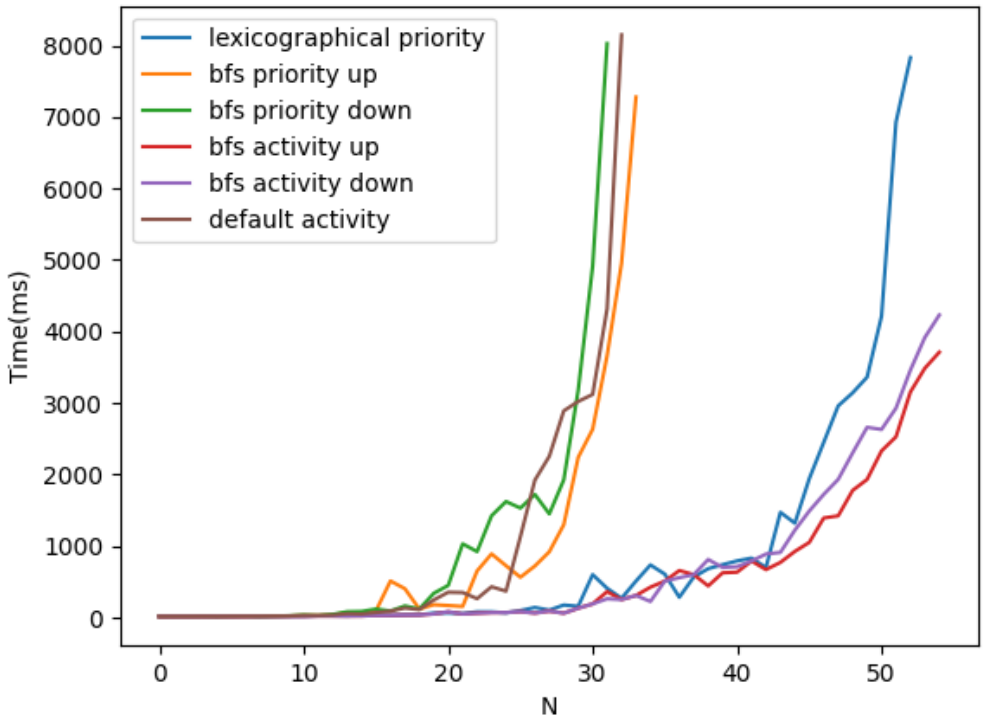


Рисунок 5 – Среднее время работы алгоритма с различными приоритетами и активностями переменных

3.5.1. Использование решателя Glucose 4.1

Так как в [4] используется далеко не самый лучший решатель *MiniSat*, логично было-бы предположить, что в более новых и совершенных SAT-решателях стандартная активность переменных должна быть более эффективной, чем приоритеты, предложенные в [4]. В связи с этим было принято решение сравнить лучшую эвристики приоритетов и активности с стандартными эвристиками активности SAT-решателя *Glucose 4.1*. Как было показано ранее, самым эффективным образом показала себя активность BFS переменных $p_{j,i}$ в порядке возрастания. Результаты тестирования показаны на рисунке 6.

— Стандартная активность переменных реализованная в SAT-решателе *Glucose 4.1* изображена на графике как *default Glucose activity*.

— Лучшая в предыдущем пункте активность переменных, основанная на алгоритме поиска в ширину, в возрастающем порядке изображена на графике *bfs activity*.

Как видно на графике, стандартные эвристики активности решателя требуют значительно больше времени для решения задачи, чем разработанные BFS активности переменных. Начиная с $|Q| = 18$ графики расходятся. Таким

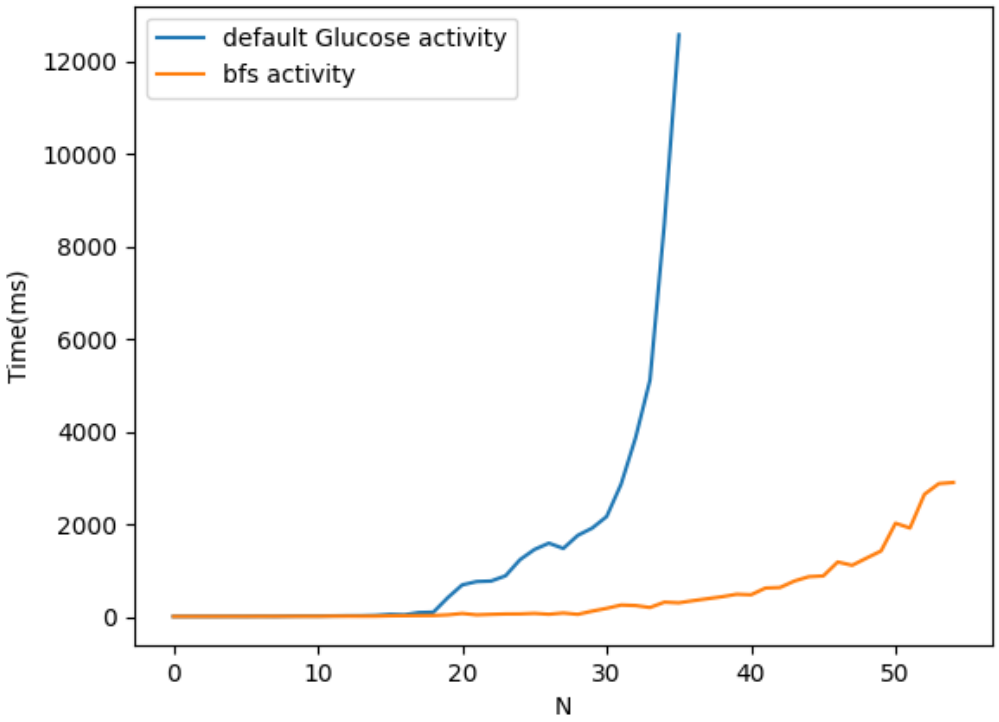


Рисунок 6 – Среднее время работы алгоритма со стандартной активностью и BFS активностью на решателе Glucose 4.1

образом можно убедиться, что предложенные в данном исследовании эвристики активности BFS предикатов нарушения симметрии работают также и на других SAT-решателях.

3.6. Эвристика инкрементального решения задачи

В данном пункте проводится сравнение эвристик инкрементального решения задачи, предложенных в [4], а также разработанных в настоящем исследовании. Сравнение проходит в двух областях: выбор стартового набора слов и выбор нового слова для добавления в решатель.

3.6.1. Эвристики стартового набора

Было проведено тестирование и сравнение эвристик стартового набора слов решателя. Сравнивались пустой набор, который используется в [4], случайный набор слов, а также $t = 3|Q|$ лексикографически первых слов. Число t выло выбрано опираясь на среднее число слов, которые алгоритм добавляет в решатель прежде чем получит искомый ДКА. Результаты тестирования показаны на рисунке 7.

— Оригинальная эвристика без стартового набора, предложенная в [4], изображена на графике как *lexicographical incremental*.
— Эвристика использующая в качестве стартового набора случайный набор слов изображена на графике как *random starting*.
— Эвристика использующая в качестве стартового набора лексикографически первые слова изображена на графике как *lexicographical starting*.

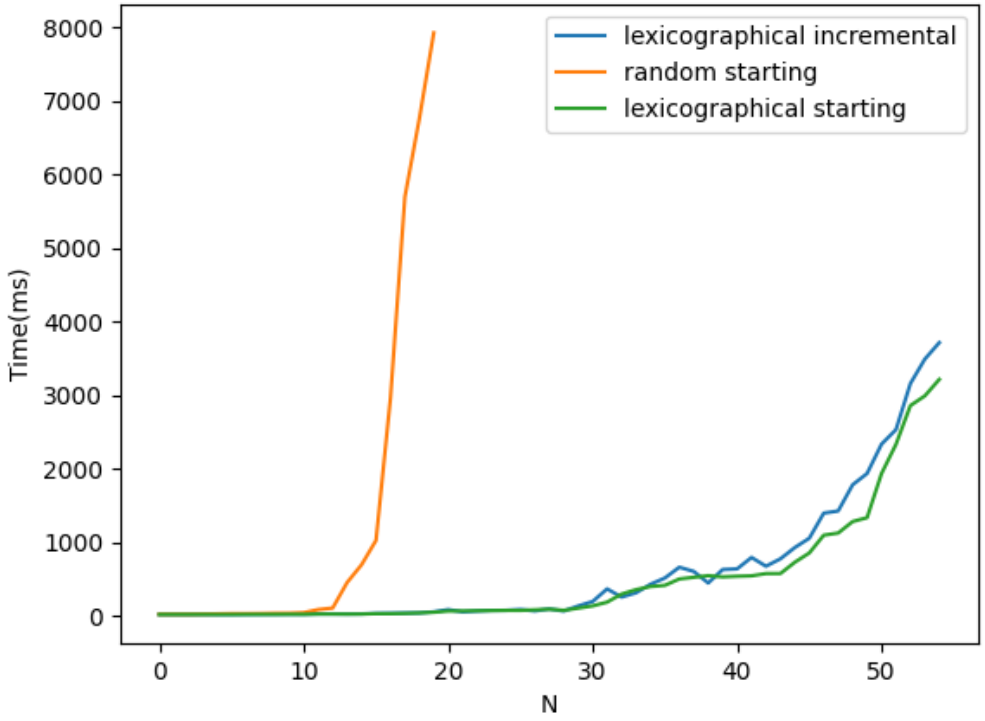


Рисунок 7 – Среднее время работы алгоритма с различными стартовыми набора слов

Результаты показали, что добавление случайных слов в решатель является крайне неэффективным подходом, время работы которого стремительно растет с $|Q| = 15$. В связи с тем, что размер словарей достаточно велик, по сравнению с размером ДКА, при случайном выборе высока вероятность выбрать длинное слово, которое внесет мало пользы в процесс построения ДКА, но при этом увеличит время решения задачи, а при большом числе таких слов время решения возрастает значительно. Также видно, что выбор $t = 3|Q|$ лексикографически первых слов и добавление их в решатель незначительно уменьшают время решения задачи. Это обуславливается тем, что при достаточно большом размере словарей высока вероятность выбрать удачный набор,

который поможет решить задачу частично, но этого будет недостаточно и придется далее добавлять слова по одному.

3.6.2. Эвристики добавления слов в SAT-решатель

Было проведено тестирование и сравнение эвристик, направленных на то, чтобы на каждой итерации алгоритма добавлять в решатель не одно, а несколько слов. Проводилось сравнение эвристики, которая вместе с лексикографически следующим словом добавляют случайно выбранное, эвристики, добавляющей поддерево размера $t = |Q|$ лексикографически следующего слова, а также решения из [4] просто добавляющего лексикографически следующее слово. Константа t была выбрана, опираясь на среднее число слов, которые алгоритм добавляет в решатель прежде чем получит искомый ДКА. Результаты тестирования показаны на рисунке 8.

— Оригинальная эвристика, добавляющая лексикографически наименьшее слово, предложенная в [4], изображена на графике как *lexicographical incremental*.
— Эвристика добавляющая в решатель случайное слово изображена на графике как *additional random*.
— Эвристика добавляющая в решатель поддерево лексикографически наименьшего слова изображена на графике как *additional tree*.

По графику видно, что добавление случайного слова к лексикографически следующему практически не влияет на время решения задачи. Это происходит из-за того, что при достаточно большом размере словарей, выбор лексикографически следующего слова с большой вероятностью является удачным и строит автомат близкий к искомому. Таким образом добавление случайного слова не оказывает влияния на результат, так как уже соответствует промежуточному ДКА. Добавление в решатель лексикографически наименьшего слова вместе с его поддеревом позволяет уменьшить время решения задачи по сравнению с добавлением только лексикографически наименьшего слова. Это происходит по причине того, что высока вероятность вместе с наименьшим словом добавить несколько следующих и обработать их за одну попытку решения SAT.

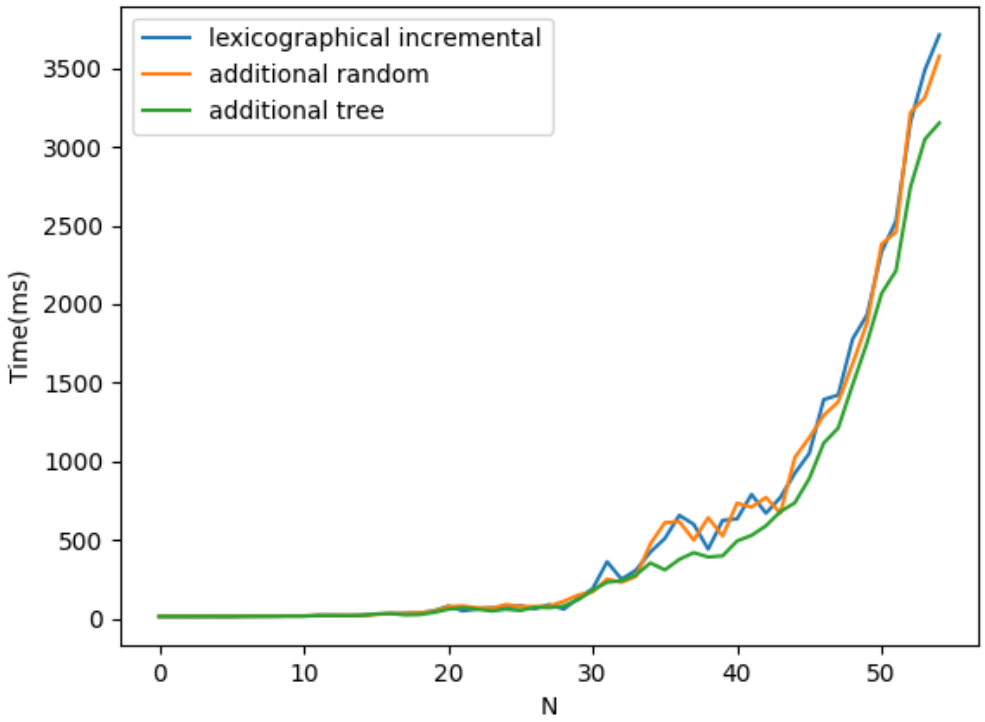


Рисунок 8 – Среднее время работы алгоритма с различными эвристиками добавления слов в решатель

3.7. Эвристика инкрементального решения задачи при неизвестном размере автомата

Было проведено тестирование и сравнение эвристик решения задачи генерации ДКА по примерам поведения при неизвестном размере автомата. Рассматривалась эвристика запоминания набора слов, который приводил к удачному ДКА на предыдущей итерации и сравнивалась со стандартным алгоритмом полного перезапуска решателя. В силу того, что алгоритму приходится решать много задач, когда искомого ДКА не существует, значительно возрастает время решения задачи и уменьшается число рассмотренных размеров ДКА соответственно. Результаты тестирования показаны на рисунке 9.

— Подход предполагающий полный перезапуск решателя изображен на графике как *full restart*.
— Подход предполагающий запоминание последнего успешного набора слов изображен на графике как *memorising*.

Как видно по графику, эвристика запоминания предыдущего набора слов показывает лучшие результаты, чем полный перезапуск начиная с $|Q| = 18$. Далее графики начинают расходиться, при $|Q| = 28$ имея разницу вдвое.

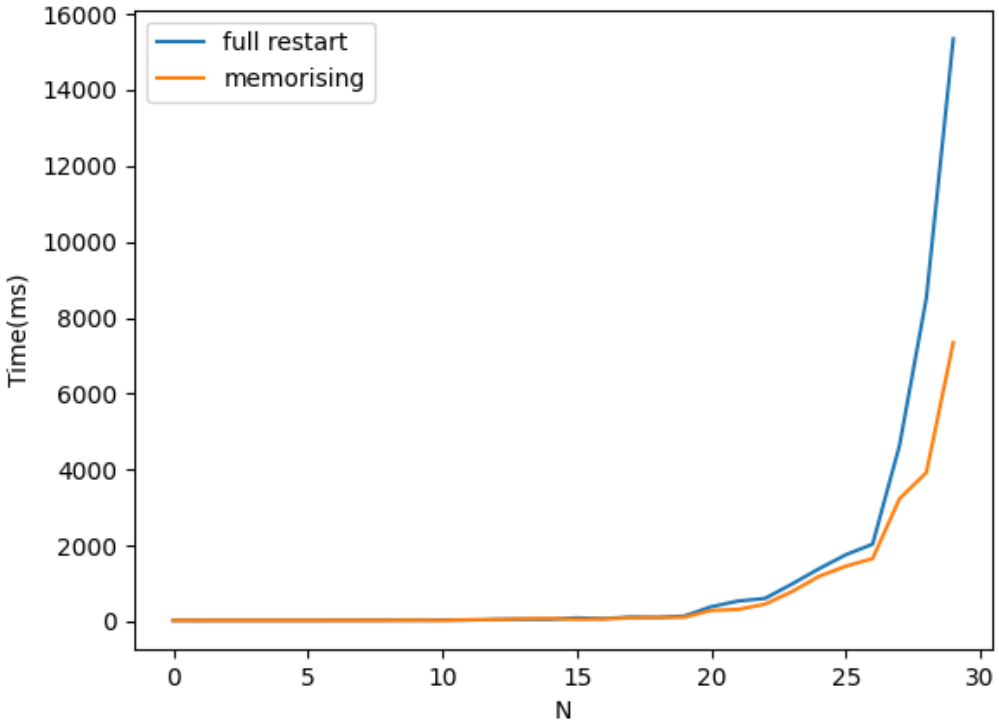


Рисунок 9 – Среднее время работы алгоритма с различными стартовыми набора слов

ЗАКЛЮЧЕНИЕ

В данной работе были изучены эвристики инкрементального решения задачи и предметной области, предложенные в [4]. Эксперименты показали, что они успешно сочетаются с предикатами нарушения симметрии на основе алгоритма поиска в ширину [16] и дают значительный прирост к производительности алгоритма, позволяя обработать более чем на 10 размеров ДКА за одну сессию тестирования, из чего делается вывод, что различные эвристики задачи построения ДКА по примерам поведения можно и нужно сочетать вместе в единые решения.

Также на основе эвристик из [4] были разработаны различные эвристики решения задачи. Эвристики приоритета и активности переменных установили порядок на переменных BFS предикатов нарушения симметрии, который передавался в SAT-решатель в качестве активности или приоритета. В экспериментальной части показано, что использование прямого порядка активности BFS предикатов позволяет улучшить время работы оригинального алгоритма из [4], использующего предикаты нарушения симметрии на основе алгоритма поиска в ширину примерно на 50%. На экспериментах с *Glucose 4.1* была показана эффективность данного подхода на более совершенных SAT-решателях.

Были рассмотрены различные эвристики выбора стартового набора слов при решени задачи инкрементальным методом. Также были разработаны эвристики добавления слов в решатель на каждой итерации. Эксперименты показали, что при решении задачи с использованием инкрементального метода выбор стартового набора слов не требуется, так как он либо не дает существенного преимущества, либо значительно замедляет алгоритм. В то-же время добавление поддерева лексикографически наименьшего слова примерно на 10% улучшает время работы алгоритма.

Также была рассмотрена задача построения ДКА по примерам поведения при неизвестном размере автомата. Была разработана эвристика запоминания предыдущего набора слов на каждой итерации. Эксперименты показали эффективность данной эвристики и уменьшение времени работы примерно на 50% при $|Q| = 28$.

Данная работа показала, что эвристики решения задачи построения ДКА по примерам поведения возможны не только путем добавления новых предикатов решатель. Модификация самого алгоритма, использование возможно-

стей SAT-решателя, его модификация, а также сочетание различных эвристик способны значительно улучшить скорость работы алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Audemard G.*, *Simon L.* The Glucose SAT Solver. — URL: `https://www.labri.fr/perso/lsimon/glucose`.

2 *Eén N.*, *Sörensson N.* The MiniSat SAT Solver. — URL: `http://minisat.se`.

3 *Aloul F.*, *Sakallah K.*, *Markov I.* Effcient symmetry breaking for boolean satisability // IEEE transactions on Computers. — 2006.

4 *Avellaneda F.*, *Petrenko A.* Learning Minimal DFA: Taking Inspiration from RPNI to Improve SAT Approach. — 2019.

5 *Biere A.*, *Heule M.*, *Maaren H. van.* Handbook of satisfiability. — Amsterdam: IOS Press, 2009.

6 *Biermann A. W.*, *Feldman J. A.* On the synthesis of finite-state machines from samples of their behavior // IEEE transactions on Computers. — 1972.

7 *Brown C. A.*, *Finkelstein L.*, *Jr. P. W. P.* Backtrack searching in the presence of symmetry. — Springer, 1988.

8 Chaff: Engineering an effcient sat solver / M. W. Moskewicz [et al.]. — 2001.

9 *Freeman J. W.* Improvements to propositional satisability search algorithms. — Citeseer, 1995.

10 *Gold E. M.* Complexity of automaton identification from given data. — Elsevier, 1978. — P. 302–320.

11 *Marijn J. H.*, *Heule M.*, *Verwer S.* Exact DFA Identification Using SAT Solvers. — Springer Berlin Heidelberg, 2010. — P. 66–79.

12 *Marques-Silva J.* The impact of branching heuristics in propositional satisability algorithms. — Springer, 1999.

13 *Oncina J.*, *Garcia P.* Inferring regular languages in polynomial updated time // World Scientist. — 1992.

14 *Pitt L.*, *Warmuth M. K.* The minimum consistent DFA problem cannot be approximated within any polynomial // Journal of the ACM. — 1993.

15 Solving difficult SAT instances in the presence of symmetry / F. Aloul [et al.]. — 2002.

16 *Ulyantsev V.*, *Zakirzyanov I.*, *Shalyto A.* FS-Based Symmetry Breaking Predicates for DFA Identification. — 2015.