МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«АДЫГЕЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики, информационных технологий и информационной безопасности

«ДОПУСКАЕТСЯ К ЗАЩИТЕ»
Заведующий кафедрой
_____ М.В. Алиев
« __ » _____ 20__ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Направление подготовки 01.04.01 Прикладная математика и информатика
Магистерская программа «Современная теория игр»

Тема

Оценки хроматического числа двумерной сферы

Научный руководитель	_____ *(подпись)*	к.т.н., доцент *(уч.степень, уч.звание)*	В.А. Воронов *(ФИО)*	__.__.__ *(дата)*
Руководитель программы магистратуры	_____ *(подпись)*	д.ф.-м.н., профессор *(уч.степень, уч.звание)*	А.В. Савватеев *(ФИО)*	__.__.__ *(дата)*

Обучающийся 2ПМ *(группа)* факультета математики и компьютерных наук И.П. Морозов *(ФИО)*

Майкоп 2020

СОДЕРЖАНИЕ

ГЛАВА 2. ЗАДАЧА ВЫПОЛНИМОСТИ БУЛЕВЫХ ФОРМУЛ И SAT-РЕШАТЕЛИ

2.1. Определения

Конъюнктивной нормальной формой (КНФ) называются булева формула вида $\Phi_m(x_1, x_2, \ldots, x_m) = D_1 \wedge D_2 \wedge \cdots \wedge D_k$, где каждый дизъюнкт $D_j = t_{j,1} \vee t_{j,2} \vee \cdots \vee t_{j,n_j}$, и все литералы $t_{j,i}$ – либо переменные, либо их отрицания, причем переменная может встречаться в дизъюнкте не более одного раза. Задача выполнимости КНФ (*ВЫП*, *SAT*) заключается в следующем: для входной формулы Φ_m указанного вида необходимо определить, существует ли набор значений переменных $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ такой, что $\Phi_m(\alpha_1, \alpha_2, \ldots, \alpha_m) = 1$, выполнима ли Φ_m? При этом исследователя часто интересует не только ответ «да» или «нет», но и сам выполняющий набор переменных или доказательство его отсутствия.

Факт принадлежности задачи *ВЫП* к классу $\mathcal{NP}$ является тривиальным. Более содержательное утверждение, знаменитая теорема Кука-Левина, открывшее важность этой задачи для теории сложности вычислений, а впоследствии и для практических приложений, доказал в 1971 году Стивен Кук (тот же результат независимо получило советский математик Леонид Левин). В своей работе [1] он впервые ввел понятие $\mathcal{NP}$-полной задачи и доказал $\mathcal{NP}$-полноту задачи выполнимости КНФ, что сделало ее первой известной $\mathcal{NP}$-полной задачей. Идея доказательства состоит в построении формулы, которая выполнима тогда и только тогда, когда соответствующая недетерминированная машина Тьюринга, решающая выбранную задачу за полиномиальное время, останавливается с положительным ответом.

Этот результат позволил доказать $\mathcal{NP}$-полноту множества других задач путем полиномиального сведения к ним задачи выполнимости КНФ, что стало стандартным способом доказательства $\mathcal{NP}$-полноты. Так, в своей работе [2] 1972 года Ричард Карп доказал $\mathcal{NP}$-полноту 21 задачи (ныне известных как «список Карпа»), среди которых: задача о клике, задача о выполнимости булевых формул с тремя литералами (*3-SAT*, для которой известен рандомизированный алгоритм ($\mathcal{PPSZ}$) со сложностью $\mathcal{O}(1.32216n)$), задача о раскраске графа и другие.

Вокруг исследования задачи выполнимости, ее приложений и обобщений сформировалось международное научное сообщество «*SAT Association*», проводится ежегодная конференция «*International Conference on Theory and Applications of Satisfiability Testing*», публикуется тематический рецензируемый журнал «*Journal on Satisfiability, Boolean Modeling and Computation, JSAT*». Среди перспективных направлений научных исслендований можно отметить изучение задач *CSP* (задача удовлетворения ограничений), *MAX-SAT* (задача поиска максимального количества выполнимых дизъюнктов), $\#SAT$ и *ALL-SAT* (подсчет количества выполняющих наборов и их поиск), *SMT* (задача выполнимости формул в теориях), *QBF* (задача о булевой формуле с кванторной приставкой), а также конструирование приближенных ($\mathcal{PTAS}$) и рандомизированных алгоритмов.

Пока вопрос о равенстве классов $\mathcal{P}$ и $\mathcal{NP}$ остается открытым, $\mathcal{NP}$-полнота задачи *SAT* свидетельствует о том, что она является вычислительно сложной и у нас нет эффективных детерминированных алгоритмов, позволяющих в общем случае решать ее за разумное время. С другой стороны, ряд важных прикладных задач естественным образом (например, с помощью преобразования Цейтина) сводится к *SAT* и ее обобщениям: задачи из области автоматизации проектирования (*EDA*), логического синтеза, автоматического доказательства теорем, криптографии, биоинформатики, проверки моделей, формальной верификации программ и автоматического конструирования тестовых покрытий, планирования, построения расписаний.

Интерес исследователей, практическая необходимость, развитие вычислительной техники и новые теоретически результаты привели к созданию ряда подходов (и программных пакетов на их основе – *SAT*-решателей), позволяющих во многих случаях эффективно решать задачу выполнимости. Все они так или иначе используют модификации метода направленного перебора и в худшем случае имеют экспоненциальную сложность. Их можно условно разделить на несколько семейств по принципу организации перебора:

1. Алгоритмы, основанные на поиске с возвратом, из которых наиболее удачным оказался *DPLL* [4] и его улучшенный вариант: *CDCL* [5] и

другие подходы, основанные на методе резолюций.

2. Вариации генетических алгоритмов, например, *GASAT* [6].

3. Подходы, основанные на применении машинного и глубокого обучения, например, *NeuroSAT* [7], который рассматривает задачу выполнимости как задачу классификации и использует для ее решения рекуррентную нейронную сеть.

4. Методы стохастического локального поиска [14], например, *WalkSAT*, базовая идея которого заключается в итеративном выборе по некоторому правилу ложного дизъюнкта и «переворачивании» одного из входящих в него литералов.

5. Алгоритмы символьных вычислений, основанные на преобразовании бинарной диаграммы решений и ее обобщениях [17].

6. Алгоритмы, основанные на построении набора правил вывода и пошаговом преобразовании формулы в эквивыполнимую [17].

Современные *SAT*-решатели представляют собой многоуровневые комбинации из нескольких подходов, дополненные различного рода эвристиками (например, случайные рестарты), оптимизациями, преобразованиями формулы [19] (например, исключение переменных методом резолюций или с помощью модифицированной процедуры Фурье-Моцкина) с десятками настраиваемых параметров. Объединение различных техник и тонкая настройка параметров позволяют эффективно решать задачи с тысячами переменных и десятками тысяч дизъюнктов. Отдельного внимания заслуживают детали реализации указанных алгоритмов. Так, мемоизация и специализированные структуры данных могут ускорить выполнение некоторых операций в разы. Другие важные аспекты – возможность инкрементального применения, масштабируемость, эффективное распараллеливание, которое может достигаться несколькими способами: портфельный метод (одновременный запуск нескольких экземпляров решателя); метод «разделяй и властвуй», основанный на разбиении формулы на независимые подформулы; параллельный локальный поиск.

Успешность современных алгоритмов при решений индустриальных задач во многом объясняется тем, что они в общем случае имеют регулярную структуру взаимосвязей между переменными. На Рис. 1 приведены визуализации индустриальной и случайно сгенерированной задач, полученные методом Clauset-Newman-Moore [24].

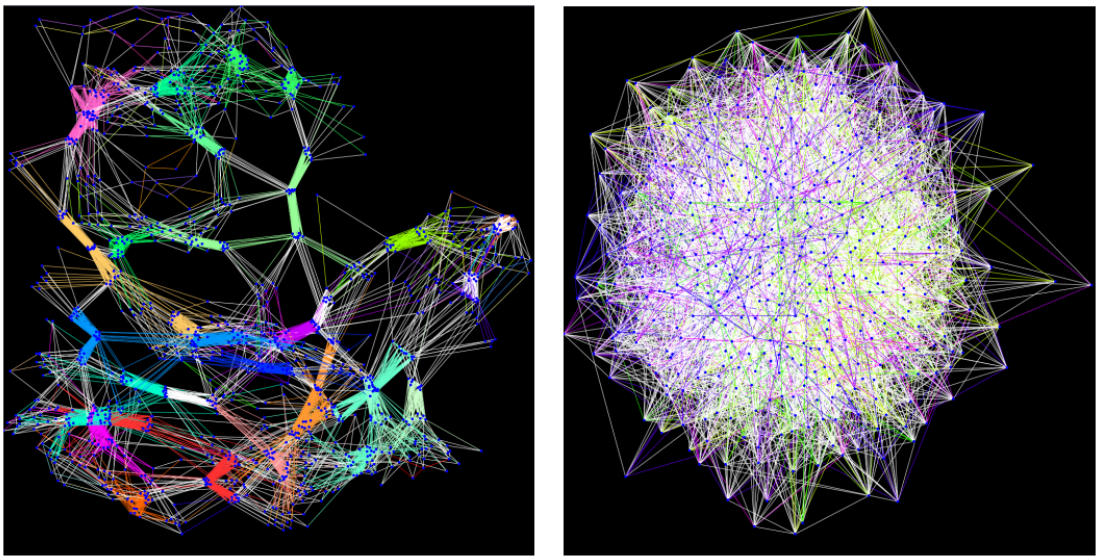

Рис. 1: Сетевая структура индустриальной (слева) и случайно сгенерированной (справа) формул.

В качестве примера удачной реализации стратегии «разделяй и властвуй» можно привести проект построения распределенного *SAT*-решателя *SAT@home* [10], выполненного на платформе для GRID-вычислений *BOINC*, реализованный лабораторией Дискретного анализа и прикладной логики Института динамики систем и теории управления СО РАН, позволивший, в числе прочего, найти несколько ортогональных пар диагональных латинских квадратов порядка 10.

Не все алгоиртмы являются полными в том смысле, что не гарантируют при любом входе завершиться с корректным ответом. Часто отсутствие полноты становится платой за введение дополнительных эвристик, которые могут значительно ускорить решение задач некоторого класса. Многие реализации алгоритма *CDCL* в случае невыполнимости КНФ позволяют построить «сертификат невыполнимости» в одном из общепринятых форматов (*TraceCheck*, *DRUP* [8]), которое можно верифицировать специальной утилитой, человеку это сделать обычно не под силу. Так, группе

ученых во главе с Марином Хойле удалось с помощью *SAT*-решателя и суперкомпьютера *Stampede* (800 ядер) построить доказательство в формате *DRAT* отсутствия такой двухцветной раскраски множества $\{1, \ldots, 7825\}$, при которой ни одна пифагорова тройка из этого множества не является одноцветной (булева проблема пифагоровых троек) [9]. Размер файла с доказательством достиг 200 терабайт. Такой метод доказательства утверждений используется все чаще, хотя и не приветствуется математическим сообществом.

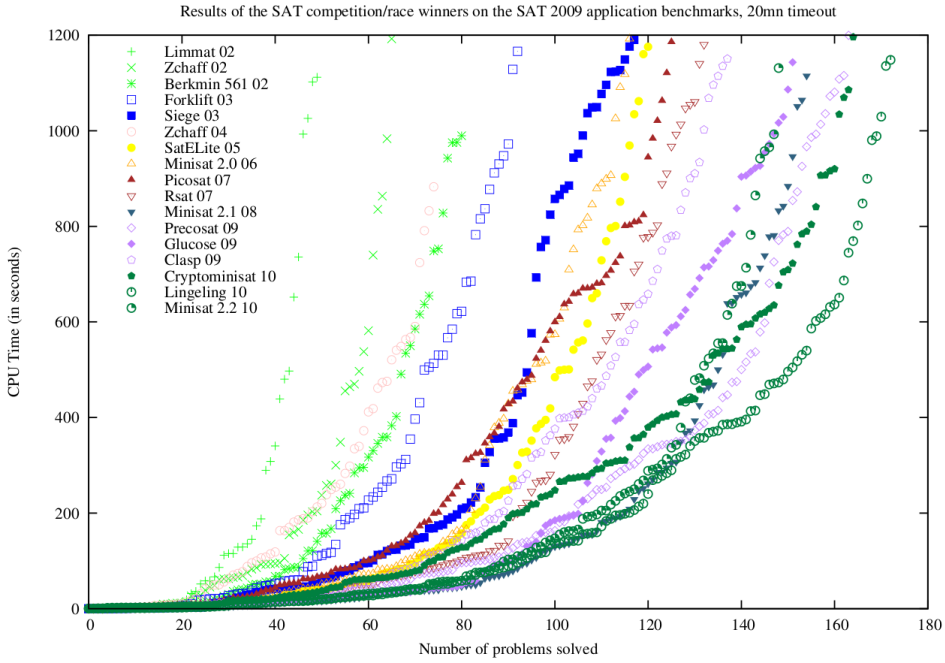

Рис. 2: Результаты The International SAT Competition 2009 года.

На протяжении двух десятков лет проводятся международные состязания по решению задачи *SAT* [11, 12], на которых участники соревнуются в скорости решения специально подобранных задач, записанных в стандартном формате *DIMACS*, при различных условиях (последовательные, параллельные вычисления) и ограничениях. Построение коротких и «сложных» конкурсных задач, а также формализация свойств формулы, которые делают ее сложной для «SAT»-решателя – отдельная интересная проблема. Результаты соревнования (Рис. 2) публикуются на сайте `http://www.satcompetition.org/`. Победителями этого соревнования в разные годы становились *MiniSAT*, *Glucose*, *Lingeling*, *CryptoMiniSat*, *YalSAT*, *MapleSAT*, *abcdSAT*, *RISS*. Все эти проекты имеют открытый исходный код и доступны

для свободного использования. Группа ученых из Университета Британской Колумбии поддерживает коллекцию SAT-задач различного уровня сложности, известную как бенчмарк $SATLIB$ [13]. На протяжении многих лет наилучшие результаты на таких соревнованиях, а также и при решении практических задач, показывают алгоритмы, базирующиеся на идее $CDCL$, о который и пойдет речь далее в этой главе.

2.2. Алгоритм DPLL

Алгоритма $DPLL$ [4] – это полный и высокоэффективный алгоритм решения задачи выполнимости, основанный на классическом алгоритме решения задач комбинаторной оптимизации: поиск в глубину с возвратом. Он назван в честь своих авторов: Дэвиса, Патнема, Логемана, Лавленда, впервые опубликован в 1962 году и является усовершенствованной версией DP, предыдущего алгоритма Дэвиса и Патнема, основанного на методу резолюций.

Далее введем несколько общепринятых в литературе обозначений. Поскольку порядок элементов не важен, здесь и далее формулу Φ удобно представлять как множество дизъюнктов $\{D_1, \ldots, D_k\}$, каждый из которых является множеством литералов $\{t_{j,1}, \ldots, t_{j,n_j}\}$ над переменными из множества X. Если множество дизъюнктов пусто, формула считается тривиально выполнимой, если один из дизъюнктов пуст - не выполнимой. В контексте алгоритмов поиска выполняющих наборов каждая переменная $x \in X$ может находиться в разных состояниях. Переменной может быть присвоено значение $\nu(x)$, $\nu : X \mapsto \{0, 1, ?\}$, где знаком «?» обозначается, что значение переменной не определено. Если $\forall x \in X\ \nu(x) \in \{0, 1\}$, то присваивание называется *полным*, иначе - *частичным*. Присваивание ν позволяет вычислить значения литерала l^ν, дизъюнкта D^ν и всей формулы Φ^ν, в этом случае говорят, что им присвоено соответствующее значение. Переменная называется *чистой*, если она входит в формулу либо только с отрицанием, либо только без отрицания. *Чистую* переменную (и все ее дизъюнкты) можно удалить из формулы, не нарушив ее выполнимость, такая операция называется *удаление чистых переменных*.

В ходе вычислений каждый дизъюнкт, в зависимости от функции присваивания, можно охарактеризовать одним из четырех состояний: *невыпол-*

ненный, *выполненный*, *единичный*, *неопределенный*. Дизюнкт называется невыполненным, если всем его литералам присвоен 0, выполненным, если хотя бы одному из его литералов присвоено значение 1, единичным, если всем литералам, кроме одного, значение которого не определено, присвоено значение 1, в остальных случаях дизюнкт считается неопределенным. Конечная цель алгоритма - сделать все дизъюнкты выполненными путем присваивания переменным значений.

Ключевой процедурой алгоритма *DPLL* является *разрешение булевых ограничений*. Если на каком-то этапе вычислений в формуле появился единичный дизъюнкт, то сделать его выполненным можно только одним способом: выбрать подходящее значение неопределеной переменной, которое называется *предполагаемым*. На этом этапе возможен *конфликт*: два разных дизъюнкта могут «потребовать» одновременно противоположных значений переменной. В ситуации конфликта присваивания, которые послужили причиной конфликта (антецедент, $\alpha(x)$), отменяются, алгоритм возвращается на шаг назад. Базовый алгоритм является рекурсивным, поэтому можно ввести понятие *уровня присваивания* переменной $\delta(x)$, который равень уровню рекурсии, на котором было выполнено присваивание. Для неопределенных переменных $\delta(x) = -1$, для предполагаемых $\delta(x)$ равно максимальному из уровней присваиваний антецедентов. На практике разрешение булевых ограничений приводит к каскадному сокращению формулы. Обозначения $x = v@d$, $d/x = v$ эквивалентны $\delta(x) = d$ и $\nu(x) = v$.

Основная схема алгоритма: по некоторому правилу выбрать из множества неопределенных переменных *переменную ветвления*, присвоить ей некоторое значение, сохранить его в *стеке присваиваний*, преобразовать формулу. Затем рекурсивно проверяется выполнимость новой формулы: если она выполнима, то и исходная формула была выполнимой, алгоритм завершает работу с результатом *SAT*, в противном случае (обнаружен конфликт) запустить ту же процедуру, используя противоположное значение переменной. Если оба значения выбранной переменной привели к конфликту, алгоритм возвращается на шаг назад, выталкивая одно присваивание из стека. Если возвращаться «некуда», алгоритм возвращает *UNSAT*. В общем случае алгоритм завершает работу с результатом *UNSAT*, если был

выполнен полный перебор всевозможных комбинаций значений переменных.

Преобразование формулы состоит из следующих шагов:

1. Из формулы удаляются все дизъюнкты, которые стали выполненными после присваивания переменной, все остальные вхождения этой переменной удаляются.

2. Выполняется разрешение булевых ограничений.

3. Выполняется удаление чистых переменных.

Псевдокод алгоритма можно записать следующим образом:

```
def DPLL(Φ):
  Φ = preprocess(Φ)
  Φ = pure_literal_elimination(Φ)
  Φ = unit_propagation(Φ)
  if Φ = {}:
    return SAT
  if {} ∈ Φ:
    return UNSAT
  x = choose_literal(Φ)
  return DPLL(Φ ∧ {x}) or DPLL(Φ ∧ {x̄})
```

Доработка этого алгоритма ведется в нескольких направлениях:

1. Построение различных эвристических правил выбора переменной ветвления и соответствующего литерала.

2. Построение ленивых структур данных, позволяющих ускорить отдельные шаги вычисления и сократить объем используемой памяти.

3. Использование «нехронологических» возвратов и «запоминание» конфликтных дизъюнктов.

Последняя идея привела к созданию алгоритма *CDCL*, который является ядром практически всех современных *SAT*-решателей.

2.3. Алгоритм CDCL

Текст

2.4. Детали реализации современных SAT-решателей

В этом разделе рассатриваются некоторые технические аспекты реализации SAT-решателей, такие как экристики ветвления, случайные рестарты, наблюдаемые литералы, структура данных, методы подбора параметров, которые сыграли ключевую роль [22] в успешности современных алгоритмов. Стоит отметить, что это далеко не исчерпывающий список приемов и их всестороннее исследование может стать темой отдельной работы.

Эвристики ветвления

Эвристикой ветвления называется алгоритм выбора переменной ветвления. Простейший способ, в данном случае, - случайный выбор. Еще один возможный вариант - выбирать ту переменную, присваивание которой порождает как можно больше единичных дизъюнктов. Парадоксально, но случайный выбор нередко позволяет получить результат быстрее прочих, поэтому все эвристики так или иначе включают элемент случайности. С другой стороны, в ходе вычислений решатель накапливает определенную информацию, которую можно использовать при выборе переменной ветвления для того, чтобы ускорить вычислительный процесс, сделать его более направленным и контролируемым, а не полагаться на волю случая.

В литературе предложен ряд эффективных методов, которые используют динамическую информацию о ходе вычислений, структуре конфликтов [18], имеют некоторые теоретические обоснования и широко используются на практике. Ключевая идея: каким-либо образом оценить «важность» переменной, используя имеющиеся данные. Стоит отметить, что вычисление такой характеристики может быть достаточно «дорогим», поэтому нередко умозрительно удачные, но «тяжелые» подходы проигрывают на практике. Далее рассматривается несколько наиболее популярных методов.

На случайно сгенерированных задачах лучшие результаты показывает эвристика, предложенная **Bohm**: на каждом шаге из множества неопределенных переменных выбирается переменная с максимальным значением вектора $H_i(x_i) = (H_1(x_i), \ldots, H_m(x_i))$ (в смысле лексикографического порядка), где $H(x) = \alpha \max\{h_i(x), h_i(\overline{x})\} + \beta \min\{h_i(x), h_i(\overline{x})\}$ и $h_i(x)$ – коли-

чество неопределенных дизъюнктов с i литералами, содержащих x. Такая эвристика стремится сделать истинными короткие дизъюнкты (при $x = 1$) либо их уменьшить.

Метод **MOM** (*Maximum Occurrences on Minimum sized clauses*) предлагает выбирать переменную x, которая максимизирует функцию $S(x) = (f^*(x) + f^*(\overline{x})) \cdot 2^k + f^*(x) \cdot f^*(\overline{x})$, где $f^*(l)$ - это количество вхождений литерала l в невыполненные дизъюнкты минимального размера. Этот метод выделяет переменные, которые входят в большое количество коротких дизъюнкций с отрицанием или без (при достаточно большом k) и одновременно.

Эвристика **Jeroslow-Wang** устроена следующим образом. Для литерала l вычисляется $J(l) = \sum_{D \in \Phi} 2^{-|D|}$ Односторонний вариант **JW-OS** предполагает выбор литерала l с наибольшим значением $J(l)$. Двусторонний **JW-TS** – поиск переменной x с наибольшей суммой $J(x) + J(\overline{x})$ и присваивание ей 1, если $J(x) \geq J(\overline{x})$. Такой метод стремится выбирать переменные, которые часто встречаются в коротких дизъюнктах.

Эвристики подсчета литералов устроены значительно проще предыдущих и имеют очевидный интуитивный смысл. Пусть $C_p(x)$ - количество неопределенных дизъюнкций, в которые x входит без отрицания, $C_n(x)$ - с отрицанием. Характеристики $C_p(x)$ и $C_n(x)$ можно учитывать в сумме или по отдельности:

1. Выбрать x, для которого $C_p(x) + C_n(x)$ максимальна (**DLCS**) и присваиванить ей 1, если $C_p(x) \geq C_n(x)$.

2. Выбрать x, для которого $C_p(x)$ ($C_n(x)$) максимальна (**DLIS**) и присвоить ей заналогично предыдущему пункту (**DLIS**) или случайно (**RDLIS**).

Характеристика **VSIDS** (*Variable State Independent Decaying Sum*) основана на анализе конфликтов и вычисляется инкрементально:

1. На старте каждым литералом ассоциируется счетчик с нулевым значением.

2. При добавлении очередной «выученной» дизъюнкции счетчик, ассоциированный с каждым литералом из этой дизъюнкции, увеличивается на 1.

3. С некоторым периодом все счетчики делятся на константу.

На шаге ветвления выбирается литерал с наибольшим значением счетчика (случайно в случае ничьей). Такая эвристика стремится удовлетворить последние конфликтные дизъюнкты и направляет процесс поиска в сторону их разрешения, что может быть особенно эффективно при решении сложных задач (например, задач с функциональными зависимостями между переменными, в которых конфликты возникают часто). Ее подсчет достаточно прост с вычислительной точки зрения: счетчики обновляются только в случае конфликта.

Метод **LEFV** (*Last Encountered Free Variable*) очень быстр и хорошо подходит для невыполнимых формул: запоминается неопределенная переменная, которую алгоритм «встретил» последней на этапе разрешения булевых ограничений. На этапе ветвления выбирается отмеченная переменная, если ее значение все еще не определено, иначе - случайная.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Cook S. A. The complexity of theorem-proving procedures //Proceedings of the third annual ACM symposium on Theory of computing. – 1971. – С. 151-158.

2. Karp R. M. Reducibility among combinatorial problems //Complexity of computer computations. – Springer, Boston, MA, 1972. – С. 85-103.

3. Rolf D. Improved bound for the PPSZ/Schöning-algorithm for 3-SAT //Journal on Satisfiability, Boolean Modeling and Computation. – 2006. – Т. 1. – №. 2. – С. 111-122.

4. Robinson J. A. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. Communications of the ACM, vol. 5 (1962), pp. 394–397 //The Journal of Symbolic Logic. – 1967. – Т. 32. – №. 1. – С. 118-118.

5. Marques-Silva J., Malik S. Propositional SAT solving //Handbook of Model Checking. – Springer, Cham, 2018. – С. 247-275.

6. Lardeux F., Saubion F., Hao J. K. GASAT: a genetic local search algorithm for the satisfiability problem //Evolutionary Computation. – 2006. – Т. 14. – №. 2. – С. 223-253.

7. Selsam D. et al. Learning a SAT solver from single-bit supervision //arXiv preprint arXiv:1802.03685. – 2018.

8. Heule M., Biere A. Proofs for satisfiability problems //All about Proofs, Proofs for all. – 2015. – Т. 55. – №. 1. – С. 1-22.

9. Heule M. J. H., Kullmann O., Marek V. W. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer //International Conference on Theory and Applications of Satisfiability Testing. – Springer, Cham, 2016. – С. 228-245.

10. Zaikin O., Kochemazov S., Semenov A. SAT-based search for systems of diagonal latin squares in volunteer computing project sat@ home //2016

39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). – IEEE, 2016. – С. 277-281.

11. Järvisalo M. et al. The international SAT solver competitions //Ai Magazine. – 2012. – Т. 33. – №. 1. – С. 89-92.

12. Biere A. Cadical, Lingeling, Plingeling, Treengeling and YalSAT entering the sat competition 2018 //Proc. of SAT Competition. – 2018. – С. 13-14.

13. Hoos H. H., Stützle T. SATLIB: An online resource for research on SAT //Sat. – 2000. – Т. 2000. – С. 283-292.

14. Hoos H. H., Stützle T. Local search algorithms for SAT: An empirical evaluation //Journal of Automated Reasoning. – 2000. – Т. 24. – №. 4. – С. 421-481.

15. Sorensson N., Een N. Minisat v1. 13-a sat solver with conflict-clause minimization //SAT. – 2005. – Т. 2005. – №. 53. – С. 1-2.

16. Biere A., Heule M., van Maaren H. (ed.). Handbook of satisfiability. – IOS press, 2009. – Т. 185.

17. Puri R., Gu J. A BDD SAT solver for satisfiability testing: An industrial case study //Annals of Mathematics and Artificial Intelligence. – 1996. – Т. 17. – №. 2. – С. 315-337.

18. Marques-Silva J. The impact of branching heuristics in propositional satisfiability algorithms //Portuguese Conference on Artificial Intelligence. – Springer, Berlin, Heidelberg, 1999. – С. 62-74.

19. Eén N., Biere A. Effective preprocessing in SAT through variable and clause elimination //International conference on theory and applications of satisfiability testing. – Springer, Berlin, Heidelberg, 2005. – С. 61-75.

20. Moskewicz M. W. et al. Chaff: Engineering an efficient SAT solver //Proceedings of the 38th annual Design Automation Conference. – 2001. – С. 530-535.

21. Kullmann O. Theory and applications of satisfiability testing //SAT. – 2009. – С. 147.

22. Katebi H., Sakallah K. A., Marques-Silva J. P. Empirical study of the anatomy of modern sat solvers //International Conference on Theory and Applications of Satisfiability Testing. – Springer, Berlin, Heidelberg, 2011. – C. 343-356.

23. Nadel A. Backtrack search algorithms for propositional logic satisfiability: Review and innovations. – Hebrew University of Jerusalem, 2002.

24. Newsham Z. et al. SATGraf: Visualizing the evolution of SAT formula structure in solvers //International Conference on Theory and Applications of Satisfiability Testing. – Springer, Cham, 2015. – C. 62-70.

ПРИЛОЖЕНИЕ 1. ДИАПАЗОНЫ ЗНАЧЕНИЙ РАДИУСА ДЛЯ РЕШЕНИЙ ЗАДАЧИ ТОМСОНА

n	k	r_{min}	r_{max}
12	7	0.6123724360498286	0.8660254030566606
14	7	0.7012189637303037	0.866624518733132
15	8	0.7793808400826431	0.8656389953148873
16	8	0.7684263761279637	0.9255571736431185
23	8	0.9266065297020001	1.0609578149116226
28	8	1.0193198671994046	1.1904870421859908
29	8	1.1118911931137054	1.1928382470918413
30	8	1.103828172201918	1.2182041997693482
32	8	0.99533136788154	1.3033066587870559
111	8	2.212357881832403	2.3346135438916327
122	8	2.1392950159792288	2.449501775328924
136	8	2.447587588495632	2.6020398401935063
195	8	3.0058363930722134	3.0753771547084856
20	9	0.9552322210490912	0.9684019184222455
22	9	0.9076990380095793	1.049810069186102
27	9	1.0046137417910734	1.1581231511243772
34	9	1.143590646093195	1.304930527120207
36	9	1.2798797747112989	1.3102016113892045
38	9	1.2570414397707683	1.3801714129853129
39	9	1.1374150070398186	1.3796828971803947
40	9	1.158566580485515	1.4020729430044754
41	9	1.2090537112274389	1.4174223482729418
42	9	1.2662387614856856	1.4345586174481404
45	9	1.447374234588995	1.4852378256091217
46	9	1.346631801514362	1.495266938914718
50	9	1.3896440935146956	1.5960840162398628
51	9	1.573281611556351	1.5772692987432342
56	9	1.595652535666857	1.6608424224821425
58	9	1.6302428305546681	1.6857859722500796

n	k	r_{min}	r_{max}
62	9	1.6606746964615353	1.7525995750841878
67	9	1.6828172987995853	1.82256677170873
68	9	1.705482381950591	1.8220306955646062
69	9	1.8204826102695895	1.820678259874453
72	9	1.6620564896383414	1.9191332930682803
75	9	1.8814218812821752	1.9049130498982851
77	9	1.878385308179857	1.9449207721566744
78	9	1.7730755729164913	1.9621470622525556
88	9	2.0756380532383494	2.0825696270197906
92	9	2.0954026638173175	2.1399873411539225
107	9	2.221137165656961	2.272252748957184
112	9	2.1840614573093817	2.3558945465936105
113	9	2.187158802364712	2.3595276881801657
115	9	2.2793434035282023	2.3524530209661334
127	9	2.3789716301202626	2.5114023425105523
128	9	2.4933247257758806	2.4990791682828184
132	9	2.2977315371836533	2.5862275269271904
135	9	2.5171094352124648	2.5877451534827087
137	9	2.4592235602983035	2.5855965185070424
146	9	2.5697820123477357	2.6719021087062877
160	9	2.6854624025078295	2.775392626525233
161	9	2.7450629229193	2.7717942991901636
162	9	2.736805681367747	2.8113962147535827
164	9	2.7502302986365557	2.8450497579658958
177	9	2.7411063865141485	2.949599139846946
182	9	2.827393724885202	2.9995678593229447
184	9	2.84287715104243	3.0157139478058337
187	9	2.861928349097998	3.0560627362196784
188	9	2.9453318480010737	3.016831277347484
192	9	2.7249661450014657	3.102751142557401
197	9	2.8976233113839487	3.1153470029840915
200	9	3.115751501406568	3.1285125569081824

n	k	r_{min}	r_{max}
202	9	3.00056508912289	3.135728972792108
204	9	2.901518149752298	3.178982675181515
209	9	3.1166508955872056	3.1906647171577056
212	9	2.9642824789475735	3.2656300632780146
214	9	3.047120652177433	3.2627704558154718
216	9	3.112953637337046	3.275995598668503
217	9	3.095319709660984	3.265621591192845
220	9	3.176622111808861	3.279539766998457
222	9	3.2415374276276974	3.2867469170401122
223	9	3.179989790277018	3.2935142969213684
224	9	3.2257135319019765	3.294997114114236
233	9	3.3082648993572685	3.346269481218835
236	9	3.333360609458009	3.379669049745145
242	9	3.4041744299077563	3.4307298966462403
243	9	3.329201645482653	3.4354532681874144
244	9	3.361701422031611	3.4308545982500145
250	9	3.324399194402528	3.5093548308361684
255	9	3.2840960211932506	3.546997679177647
256	9	3.3059727735358138	3.5529537342201105
257	9	3.2828353509437154	3.573897570495309
258	9	3.360064110241416	3.5665641294387487
259	9	3.356534084563132	3.576180156153748
263	9	3.3574674048060174	3.5988119922922652
264	9	3.5303622446074416	3.613529580131228
272	9	3.2479524300443976	3.6920279105437777
273	9	3.471103777455965	3.662315088916946
274	9	3.5098241951640623	3.6583403249876576
275	9	3.5985687355642995	3.649710877908703
276	9	3.4626711347373873	3.6982579099058204
277	9	3.6471563235851163	3.6524196159972626
278	9	3.5869150615879164	3.6731535804441853
279	9	3.5959131199981478	3.6804256458959355

n	k	r_{min}	r_{max}
282	9	3.330095988029239	3.7562436984942464
287	9	3.7211497733684866	3.721424249430957
288	9	3.6243345761291996	3.7513569556452264
291	9	3.589989960494795	3.785220806657392
292	9	3.5549133441619825	3.786237548829822
293	9	3.557209395382303	3.808151184595572
295	9	3.7228788034921636	3.786847535818198
299	9	3.7282035049944264	3.8019039222467694
304	9	3.713236032410276	3.8484715564187724
305	9	3.7395550485483353	3.8506719173411037
306	9	3.6586960278980984	3.8911190055267184
308	9	3.686198820686234	3.897642998973312
311	9	3.7593650380637214	3.8926817082821366
314	9	3.7371303147614583	3.93462175844112
315	9	3.6916241962952916	3.947070262499115
316	9	3.7774173750757263	3.943345674962319
317	9	3.726422432591502	3.9428016618278385
318	9	3.780896519239086	3.955933141393098
320	9	3.8329705356293515	3.9500458689396263
322	9	3.81217777387446	3.972016673210202
324	9	3.8473829200425427	3.969229913818113
327	9	3.9416975926750353	3.987110181065034
328	9	3.808033857771553	4.007346238741135
329	9	3.8589052406508273	4.010903315769737
330	9	3.8669654421498647	4.029802976479282
331	9	3.8612215197125908	4.009510505471454
332	9	3.8956870683259006	4.0319750383225985
333	9	3.8715004679486476	4.028419743285473
339	9	3.94563396667274	4.055849145883436
340	9	4.0324475926749015	4.035034582166366
344	9	3.9758764453660054	4.088398714023686
345	9	4.07364039366915	4.085135336245742

n	k	r_{min}	r_{max}
346	9	4.025451688033153	4.082797800082753
348	9	3.8175778625851056	4.15211907479343
349	9	4.078584000628203	4.1024770247652445
356	9	3.907906368339495	4.197006792711948
357	9	3.8602933165289515	4.207005210997042
358	9	3.9076988119905454	4.209311084069152
367	9	3.99190579971022	4.249084714640408
372	9	3.841095568367235	4.323232416190199
375	9	4.054259242071652	4.290091782561422
390	9	4.0487617046216595	4.409909682102864
392	9	3.9511356594065847	4.429186846675495
400	9	4.178863159337794	4.449448136318128

ПРИЛОЖЕНИЕ 2. КОДИРОВАНИЕ ЗАДАЧИ РАСКРАСКИ ГРАФА

```
import math
import numpy as np
from numpy import linalg as LA
import networkx as nx

#import matplotlib.pylab as plt

#from google.colab import files
#files.upload()
#files.download("file.txt")

#https://pysathq.github.io/
#!pip install python-sat
import pysat
from pysat.formula import CNF
from pysat.solvers import *

import os
import sys
import random
from zipfile import ZipFile
from tqdm import tqdm_notebook as tqdm
from itertools import combinations, permutations

out_dir = ""

class Utils:

    def read_dimacs_graph(file = 'graph.col'):

        if not (os.path.exists(file) and os.path.getsize(file) > 0):
            raise Exception("File " + file + " not found")

        nodes = []
        edges = []
        labels = []
        got_labels = False
        nnodes = nedges = 0

        with open(file, 'r') as f:
            for line in f:
                line = [l.strip() for l in line.split(' ')]
                if line[0] == 'c': #comment
                    continue
                elif line[0] == 'p': #problem
                    nnodes = int(line[2])
                    nedges = int(line[3])
                    nodes = list(range(1, nnodes + 1))
                    labels = [0] * nnodes
                elif line[0] == 'e': #edge
                    edges.append((int(line[1]), int(line[2])))
                elif line[0] == 'l':
                    labels[int(line[1]) - 1] = int(line[2])
                    got_labels = True

        if got_labels:
            nodes = [(n, {'c' : l}) for n, l in zip(nodes, labels)]

        g = nx.Graph()
        g.add_nodes_from(nodes)
        g.add_edges_from(edges)
        return g

    def write_dimacs_graph(file = 'graph.col', g = nx.Graph(), comments =
    []):
        with open(file, 'w') as f:
            for c in comments:
                f.write("c " + c + "\n")
            f.write("p EDGE {} {}\n".format(g.number_of_nodes(), g.
    number_of_edges()))
            for u, v in g.edges():
                f.write("e {} {}\n".format(u, v))
            for node in g.nodes():
```

```
                if 'c' in g.node[node]:
                    f.write("l {} {}\n".format(node, g.node[node]['c']))

    def draw_with_colors(g = nx.Graph()):
        color_map = []
        for node in g.nodes():
            if 'c' in g.node[node]:
                color_map.append(g.node[node]['c'] * 10)
        nx.draw(g, pos = nx.spring_layout(g, scale=2), node_color=
    color_map, with_labels=True, cmap = plt.cm.jet)

    def write_proof(file = "proof.txt", proof = []):
        with open(file, 'w') as f:
            for p in proof:
                f.write("%s\n" % str(p))

    def zip_files(file = "archive.zip", files = []):
        with ZipFile(file, 'w') as archive:
            for f in set(files):
                if not (os.path.exists(file) and os.path.getsize(file) >
    0):
                    raise Exception("File " + file + " not found")
                archive.write(f)

def find_triangle(g = nx.Graph()):
    for a in g:
        for b, c in combinations(g[a], 2):
            if b in g[c]:
                return [a, b, c]
    return []

    #return set(frozenset([a, b, c]) for a in g for b, c in combinations(g
    [a], 2) if b in g[c])


def find_isolates(g = nx.Graph()):
    isolates = []
    for n in g:
        if g.degree(n) == 0:
            isolates.append(n)
    return isolates

class ColMap:

    def __init__(self, g = nx.Graph(), ncolors = 40):

        self.ncolors = ncolors
        self.cmap = dict()
        self.cunmap = dict()

        i = 1
        for node in g.nodes():
            for color in range(1, ncolors + 1):
                self.cmap[(node, color)] = i
                self.cunmap[i] = (node, color)
                i += 1

    def enc(self, node, color):
        return self.cmap[(node, color)]

    def dec(self, node_color):
        return self.cunmap[node_color]

class ColSAT:

    def __init__(self, g = nx.Graph(), ncolors = 10):

        self.ncolors = ncolors
        self.g = g.copy()
        self.cmap = ColMap(g, ncolors)

    def check_coloring(self):
        for n1, n2 in self.g.edges():
            if 'c' not in self.g.node[n1] or 'c' not in self.g.node[n2]:
                return False
            if self.g.node[n1]['c'] == self.g.node[n2]['c']:
                return False
        return True
```

```
147
148     def apply_model(self):
149
150         check = set()
151         for var in self.model[self.model > 0]:
152             node, color = self.cmap.dec(var)
153             self.g.node[node]['c'] = color
154             if (node, color) in check:
155                 raise Exception("Two colors for one node???")
156             else:
157                 check.add((node, color))
158
159         self.colored = self.check_coloring()
160
161         if self.colored != self.solved:
162             raise Exception("Something went wrong!")
163
164         return self.colored
165
166     def build_cnf(self):
167
168         self.formula = CNF()
169         colors = list(range(1, self.ncolors + 1))
170
171         for n1, n2 in self.g.edges():
172             for c in colors:
173                 self.formula.append([-self.cmap.enc(n1, c), -self.cmap.enc
    (n2, c)])
174
175         #specials = [28, 194, 242, 355, 387, 397, 468]
176         #ii = 1
177         #for n in specials:
178         #    self.formula.append([self.cmap.enc(n, ii)])
179         #   ii += 1
180
181
182         for n in self.g.nodes():
183             #if not n in specials:
184             self.formula.append([self.cmap.enc(n, c) for c in colors])
185             for c1 in colors:
186                 for c2 in colors:
187                     if c1 < c2:
188                         self.formula.append([-self.cmap.enc(n, c1), -self.
    cmap.enc(n, c2)])
189
190         return self.formula
191
192     def solve_cnf(self, solver = ''):
193
194         triangle = find_triangle(self.g)
195         assumptions = []
196         if len(triangle) > 0:
197             assumptions = [self.cmap.enc(triangle[0], 1), self.cmap.enc(
    triangle[1], 2), self.cmap.enc(triangle[2], 3)]
198
199         #Glucose3, Glucose4, Lingeling, MapleChrono, MapleCM, Maplesat,
    Minisat22, MinisatGH
200         #with Glucose4(bootstrap_with=self.formula.clauses, with_proof=
    True) as ms:
201         with Lingeling(bootstrap_with=self.formula.clauses) as ms:
202             self.solved = ms.solve(assumptions=assumptions)
203             if self.solved:
204                 self.model = np.array(ms.get_model())
205                 self.apply_model()
206             else:
207                 self.proof = []#ms.get_proof()
208                 self.colored = False
209
210         return self.solved
211
212 if __name__ == "__main__":
213
214     if len(sys.argv) < 4:
215         raise "I need in_file out_file ncolors"
216
217     infile = sys.argv[1]
218     outfile = sys.argv[2]
219     ncolors = int(sys.argv[3])
220     g = Utils.read_dimacs_graph(infile)
```

```
221     problem = ColSAT(g, ncolors)
222     problem.build_cnf().to_file(outfile)
```

ПРИЛОЖЕНИЕ 3. ВЫЧИСЛЕНИЕ ГРАНИЦ ДИАПАЗОНОВ ЗНАЧЕНИЙ РАДИУСА

```
import os
import numpy as np
import itertools
import networkx as nx
import scipy.optimize as opt
from numpy.linalg import norm
from scipy.spatial import SphericalVoronoi

def read_points2(filename):
    points = []
    f = open(filename, 'r')
    k = int(f.readline())
    f.readline()
    for s in f:
        s = ' '.join(s.split())
        l = s.split(' ')
        points.append(np.array([float(l[1]), float(l[2]), float(l[3])], dtype=np.float64))
    f.close()
    return np.array(points)

def read_triang2(filename):
    points = []
    f = open(filename, 'r')
    k = int(f.readline())
    for s in f:
        s = ' '.join(s.split())
        l = s.split(' ')
        points.append(np.array([int(l[0]), int(l[1]), int(l[2])], dtype=np.int32))
    f.close()
    return np.array(points)

def write_dimacs(g, filename):
    g1 = g #nx.convert_node_labels_to_integers(g, first_label=1)
    f = open(filename, 'w')
    f.write('p edges ' + str(g.number_of_nodes()) + ' ' + str(g.number_of_edges()) + '\n')
    for e in g1.edges():
        f.write('e ' + str(e[0]) + ' ' + str(e[1]) + '\n')
    f.close()

def write_faces(faces, filename):
    f = open(filename, 'w')
    for face in faces:
        for p in face:
            f.write(' '.join(map(str, p)) + '\n')
        f.write(' \n')
    f.close()

'''
def build_g2(g):
    paths = dict(nx.all_pairs_shortest_path(g, 2))

    G2 = G.copy()
    keys = list(paths.keys())
    for key in keys:
        for nn in list(paths.get(key).keys()):
            G2.add_edge(key, nn)

    return G2
'''

def connect_dist2(g):

    gg = g.copy()

    to_connect = []
    for i in gg.nodes():
        for j in gg.nodes():
            if nx.shortest_path_length(gg, i, j) == 2:
```

```
                to_connect.append([i,j])
    for e in to_connect:
        gg.add_edge(e[0], e[1])
    return gg

def dist(u, v):
    return norm(u - v)

# angle between radius vectors
def angle(p1 ,p2):
    c = np.dot(p1, p2) / norm(p1) / norm(p2)
    return np.arccos(np.clip(c, -1, 1))

# mid arc between  x, y
def middle(x, y):
    z = (x + y) / 2
    return z / norm(z)

# area diameter
def get_diam(faces):
    diam = 0.0
    for f in faces:
        ij = itertools.combinations(range(len(f)), 2)
        dists = np.array([dist(f[i], f[j]) for i, j in ij])
        diam = np.max([diam, np.max(dists)])
    return diam

def plane_equation(x, y, z):
    a1 = x[1] - x[0]
    b1 = y[1] - y[0]
    c1 = z[1] - z[0]
    a2 = x[2] - x[0]
    b2 = y[2] - y[0]
    c2 = z[2] - z[0]
    a = b1 * c2 - b2 * c1
    b = a2 * c1 - a1 * c2
    c = a1 * b2 - b1 * a2
    d = (- a * x[0] - b * y[0] - c * z[0])
    return np.array([a, b, c, d])

def foot(x,y,z,v):
    p = plane_equation(x, y, z)
    n = np.array([p[0], p[1], p[2]])
    l = norm(n)
    n = n / l #[n[0]/l,n[1]/l,n[2]/l]
    h = p[0]*v[0] + p[1]*v[1] + p[2]*v[2] + p[3]
    n1 = n * h / l #[n[0]*h/l,n[1]*h/l,n[2]*h/l]
    return v - n1 #[v[0]-n1[0],v[1]-n1[1],v[2]-n1[2]]

def circ_dist(x,y,v,eps=1E-8):
    z = np.array([0.0, 0.0, 0.0])
    f = foot(x,y,z,v)

    l = norm(f)
    f = f / l #[f[0]/l,f[1]/l,f[2]/l]
    a1 = angle(x,f)
    a2 = angle(f,y)
    a3 = angle(x,y)
    d = min(dist(x,v), dist(y,v))
    if abs(a1+a2-a3)<eps:
        d = min(d,dist(f,v))
    return d

# distance between areas
def faces_dist(face1, face2):
    f1 = face1
    f2 = face2
    d = 2.0
    for i in range(len(f1)):
        for j in range(len(f2)):
            v = f1[i]
            x = f2[j]
            if j==len(f2)-1:
                y = f2[0]
            else:
                y = f2[j+1]
            d = min(d,circ_dist(x,y,v))
    for i in range(len(f2)):
```

```
147         for j in range(len(f1)):
148             v = f2[i]
149             x = f1[j]
150             if j==len(f1)-1:
151                 y = f1[0]
152             else:
153                 y = f1[j+1]
154             d = min(d,circ_dist(x,y,v))
155     return d
156 
157 # center of the circumscribed circle of a spherical triangle, vertex of
    the Voronoi diagram
158 def center(x, y, z):
159     c = np.vstack([x, y, z])
160     x0 = np.mean(c, axis=0)
161     x0 = x0 / norm(x0)
162     f = lambda v: np.square(np.dot(v, x - y)) + np.square(np.dot(v, x - z)
    ) + np.square(np.dot(v, v) - 1)
163     sol = opt.minimize(f, x0, method='nelder-mead')
164     return sol.x
165 
166 def get_dual(g, points):
167     faces = []
168     for v in g.nodes():
169         neigh = g.neighbors(v)
170 
171         try:
172             n_cyc = nx.cycle_basis(g.subgraph(neigh))[0]   # adjacent
    vertex cycle
173             n_cyc.append(n_cyc[0])
174         except IndexError:
175             pass
176             #print(nx.cycle_basis(g.subgraph(neigh)))
177 
178         face = []   # points of the area
179         for i in range(len(n_cyc) - 1):
180             c = center(points[v - 1], points[n_cyc[i] - 1], points[n_cyc[i
    +1] - 1])
181             face.append(c)
182 
183         faces.append(face)
184     return faces
185 
186 def get_dual2(g, points):
187     sv = SphericalVoronoi(points)
188     sv.sort_vertices_of_regions()
189     faces = []
190     for region in sv.regions:
191         face = sv.vertices[region]
192         faces.append(face)
193     return np.array(faces)
194 
195 # minimum of distances between regions at a distance> 3
196 def faces_d3_dist2(g, faces):
197     n = g.number_of_nodes()
198     d = 2.0
199     for i in g.nodes():
200         for j in g.nodes():
201             if nx.shortest_path_length(g, i, j) == 3:
202                 d = np.min([d, faces_dist(faces[i - 1], faces[j - 1])])
203     return d
204 
205 def faces_d3_dist(g, faces):
206 
207     d = 2.0
208     paths = dict(nx.all_pairs_shortest_path_length(g, 3))
209 
210     for i in g.nodes():
211         for j in g.nodes():
212             if i in paths:
213                 if j in paths[i]:
214                     if paths[i][j] == 3:
215                         d = np.min([d, faces_dist(faces[i - 1], faces[j -
    1])])
216 
217     return d
218 
219 
```

```
thomsons = [f.strip() for f in open('thomson/list_of_files.txt')]

thomsons.sort(key=lambda x: int(x.replace('.xyz', '')))

for thomson in thomsons:
    try:
        print(thomson)

        n_thomson = int(thomson.replace('.xyz', ''))
        points = read_points2('thomson/' + thomson)

        f = open('tmp', 'w')
        f.write('3 \n')
        f.write(str(len(points)) + '\n')
        for p in points:
            f.write(' '.join(map(str, p)) + '\n')
        f.close()

        cmd = 'C:\\cpp\\qhull-2019.1\\bin\\qconvex.exe i < tmp > ' + 'thomson1/' + str(n_thomson) + '.g'
        os.system(cmd)

        triangles = read_triang2('thomson1/' + str(n_thomson) + '.g')

        G = nx.Graph()

        for t in triangles:
            t = t + 1
            G.add_edge(t[0], t[1])
            G.add_edge(t[1], t[2])
            G.add_edge(t[0], t[2])

        write_dimacs(G, 'g/' + str(n_thomson) + '.g')

        good = True

        for v,d in G.degree:
            if (d != 5) and (d != 6):
                good = False
                break

        if good:
            write_dimacs(connect_dist2(G), 'g2/' + str(n_thomson) + '.g2')

            faces = get_dual2(G, points)
            d0 = get_diam(faces)
            d1 = faces_d3_dist(G, faces)

            #if d1/d0 > 1:
            print(thomson + ' ' + str(d0) + ' ' + str(d1) + ' ' + str(d1/d0))
            write_faces(faces, 'vor/' + str(n_thomson) + '.vor')
    except Exception as e:
        print('err' + str(e))
```

ПРИЛОЖЕНИЕ 4. ВИЗУАЛИЗАЦИЯ РАСКРАСОК

```
1
2  #include "Utils.hpp"
3  #include <gl2ps-1.4.0-source/gl2ps.h>
4  #include <freeglut/include/GL/glut.h>
5  #include <glm/vec3.hpp>
6
7  Mouse mouse = {0, 0};
8  double cameraDistance = 3.5;
9  std::pair<double, double> cameraAngleXY(0, 0), cameraTransXY(0, 0);
10 std::pair<int, int> screenWH;
11
12 std::string fileName;
13 std::string nColors;
14 std::vector<Vec3d> points;
15 std::map<int, int> coloring;
16 std::set<std::pair<int, int>> sedges, sedges2, sedges3;
17 std::vector<std::pair<int, int>> edges, edges2;
18 std::vector<std::vector<Vec3d>> faces, faces2;
19 std::vector<GLfloat> facesTriangles;
20
21 std::string vshader =
22 "#version 330 core"
23 "layout(location = 0) in vec3 pos;"
24 "void main() {"
25     "gl_Position.xyz = pos;"
26 "}";
27
28 std::vector<Vec3d> palette =
29 {
30     { 0.0, 0.0, 0.0 },
31     { 1.0, 0.0, 0.0 },
32     { 0.0, 1.0, 0.0 },
33     { 1.0, 1.0, 0.0 },
34     { 0.0, 0.0, 1.0 },
35     { 0.60, 0.40, 0.12 },
36     { 1.0, 0.0, 1.0 },
37     { 0.75, 0.75, 0.75 },
38     { 0.0, 1.0, 1.0 },
39     { 0.25, 0.25, 0.25 },
40     { 0.98, 0.625, 0.12 },
41     { 0.98, 0.04, 0.7 },
42     { 0.60, 0.40, 0.70 },
43     { 1.0, 1.0, 1.0 },
44 };
45
46 bool drawPoints = false;
47 bool drawSphere = false;
48 bool drawG = false;
49 bool drawG2 = false;
50 bool drawFaces = true;
51 bool drawColors = true;
52 bool drawAxes = false;
53 bool drawFacesSkeletons = false;
54
55
56 void DisplayCallback();
57 void DoDraw();
58
59 void save(const std::string& fileName, int type = GL2PS_PDF)
60 {
61     FILE *fp;
62     GLint buffsize = 0, state = GL2PS_OVERFLOW;
63     fp = fopen(fileName.c_str(), "wb");
64     printf("Saving ... \n");
65     while (state == GL2PS_OVERFLOW) {
66         buffsize += 1024 * 1024;
67         gl2psBeginPage("test", "gl2psTestSimple", NULL,
68             GL2PS_PDF, GL2PS_SIMPLE_SORT,
69             GL2PS_DRAW_BACKGROUND | GL2PS_USE_CURRENT_VIEWPORT,
70             GL_RGBA, 0, NULL, 100, 100, 100, buffsize, fp, fileName.c_str
    ());
71         DisplayCallback();
72         state = gl2psEndPage();
73     }
74     fclose(fp);
```

```
75      printf("Done \n");
76  }
77
78  void DisplayCallback()
79  {
80      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
        GL_STENCIL_BUFFER_BIT);
81
82      /*
83      glPushMatrix();
84      glTranslatef(-2, -2, 0);
85      DoDraw();
86      glPopMatrix();
87
88      glPushMatrix();
89      glRotatef(180, 0, 1, 0);
90      glTranslatef(-2, 2, 0);
91      DoDraw();
92      glPopMatrix();
93
94      glPushMatrix();
95      glTranslatef(2, 2, 0);
96      DoDraw();
97      glPopMatrix();
98
99      glPushMatrix();
100     glTranslatef(2, -2, 0);
101     DoDraw();
102     glPopMatrix();
103     */
104
105     DoDraw();
106     glFlush();
107     glutSwapBuffers();
108 }
109
110 void DoDraw()
111 {
112     glPushMatrix();
113
114     //camera
115     {
116         glTranslatef(cameraTransXY.first, cameraTransXY.second, -
        cameraDistance);
117         glRotatef(cameraAngleXY.first, 1, 0, 0);
118         glRotatef(cameraAngleXY.second, 0, 1, 0);
119     }
120
121     if (drawSphere)
122     {
123         glColor4f(0.3f, 0.3f, 0.3f, 1.f);
124         //glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
125         glutSolidSphere(0.99f, 100, 100);
126     }
127
128     if (drawPoints)
129     {
130         int i = 0;
131         //auto p = points[15];
132         for (auto& p : points)
133         {
134             if (drawColors)
135             {
136                 const auto& clr = palette[coloring[i + 1]];
137                 glColor4f(clr.x, clr.y, clr.z, 1.f);
138             }
139             else
140             {
141                 glColor4f(0.0f, 0.0f, 1.0f, 1.f);
142             }
143
144             i++;
145             glPushMatrix();
146             glTranslatef(p.x, p.y, p.z);
147             glutSolidSphere(0.03f, 10, 10);
148             glPopMatrix();
149         }
```

```
150     }
151
152     if (drawG)
153     {
154         glColor4f(1.0f, 0.0f, 0.0f, 1.5f);
155
156         for (auto& e : edges)
157         {
158             //if (e.first - 1 != 15)
159             //{
160             //    continue;
161             //}
162             auto& p1 = points[e.first - 1];
163             auto& p2 = points[e.second - 1];
164             glBegin(GL_LINES);
165             glVertex3f(p1.x, p1.y, p1.z);
166             glVertex3f(p2.x, p2.y, p2.z);
167             glEnd();
168         }
169     }
170
171     if (drawG2)
172     {
173         glColor4f(0.0f, 0.3f, 0.0f, 1.2f);
174
175         for (auto& e : sedges3)
176         {
177             //if (e.first - 1 != 15)
178             //{
179             //    continue;
180             //}
181             auto& p1 = points[e.first - 1];
182             auto& p2 = points[e.second - 1];
183             glBegin(GL_LINES);
184             glVertex3f(p1.x, p1.y, p1.z);
185             glVertex3f(p2.x, p2.y, p2.z);
186             glEnd();
187         }
188     }
189
190     if (drawFacesSkeletons)
191     {
192         glPushMatrix();
193
194         for (auto& face : faces)
195         {
196             glColor4f(0.6f, 0.3f, 0.2f, 1.f);
197             for (size_t i = 0; i < face.size(); i++)
198             {
199                 auto& p1 = face[i];
200                 auto& p2 = face[(i + 1) % face.size()];
201
202                 glBegin(GL_LINES);
203                 glVertex3f(p1.x, p1.y, p1.z);
204                 glVertex3f(p2.x, p2.y, p2.z);
205                 glEnd();
206             }
207         }
208
209         glPopMatrix();
210     }
211
212     if (drawFaces)
213     {
214         glPushMatrix();
215
216         glLineWidth(2.f);
217
218         int i = 0;
219         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
220         //auto& face = faces[15];
221         for (auto& face : faces2)
222         {
223             auto l = face.size();
224
225             if (drawColors)
226             {
```

```
227                 const auto& clr = palette[coloring[i + 1]];
228                 glColor4f(clr.x, clr.y, clr.z, 1.f);
229             }
230             else
231             {
232                 glColor4f(0.6f, 0.3f, 0.2f, 1.f);
233             }
234 
235             auto& c = points[i];
236             std::vector<GLfloat> triangles;
237             for (size_t j = 0; j < l; j++)
238             {
239                 auto& p1 = face[j];
240                 auto& p2 = face[(j + 1) % l];
241 
242                 //glBegin(GL_TRIANGLES);
243                 //glVertex3f(p1.x, p1.y, p1.z);
244                 //glVertex3f(p2.x, p2.y, p2.z);
245                 //glVertex3f(c.x, c.y, c.z);
246                 //glEnd();
247 
248                 triangles.push_back(p1.x);
249                 triangles.push_back(p1.y);
250                 triangles.push_back(p1.z);
251 
252                 triangles.push_back(p2.x);
253                 triangles.push_back(p2.y);
254                 triangles.push_back(p2.z);
255 
256                 triangles.push_back(c.x);
257                 triangles.push_back(c.y);
258                 triangles.push_back(c.z);
259             }
260             glEnableClientState(GL_VERTEX_ARRAY);
261             glVertexPointer(3, GL_FLOAT, 0, triangles.data());
262             glDrawArrays(GL_TRIANGLES, 0, triangles.size() / 3);
263             glDisableClientState(GL_VERTEX_ARRAY);
264 
265 
266             /*
267             glBegin(GL_POLYGON);
268             for (auto& p : face)
269             {
270                 glVertex3f(p.x, p.y, p.z);
271             }
272             glEnd();
273             */
274 
275             i++;
276         }
277         glPopMatrix();
278     }
279 
280     if (drawAxes)
281     {
282         glPushMatrix();
283 
284         glutSolidSphere(.01f, 100, 100);
285 
286         //x
287         glColor4f(0.0f, 0.3f, 0.0f, 1.5f);
288         glBegin(GL_LINES);
289         glVertex3f(-2, 0, 0);
290         glVertex3f(2, 0, 0);
291 
292         //arrow
293         glVertex3f(2.0, 0.0f, 0.0f);
294         glVertex3f(1.8, 0.1f, 0.0f);
295         glVertex3f(2.0, 0.0f, 0.0f);
296         glVertex3f(1.8, -0.1f, 0.0f);
297 
298         glEnd();
299 
300         //y
301         glColor4f(0.3f, 0.0f, 0.0f, 1.5f);
302         glBegin(GL_LINES);
303         glVertex3f(0, -2, 0);
```

```
304             glVertex3f(0, 2, 0);
305 
306             //arrow
307             glVertex3f(0.0f, 2.0f, 0.0f);
308             glVertex3f(0.1f, 1.8f, 0.0f);
309             glVertex3f(0.0f, 2.0f, 0.0f);
310             glVertex3f(-0.1f, 1.8f, 0.0f);
311 
312             glEnd();
313 
314             //z
315             glColor4f(0.0f, 0.0f, 0.3f, 1.5f);
316             glBegin(GL_LINES);
317             glVertex3f(0, 0, -2);
318             glVertex3f(0, 0, 2);
319 
320             //arrow
321             glVertex3f(0.0, 0.0f, -2.0f);
322             glVertex3f(0.0, 0.1f, -1.8f);
323             glVertex3f(0.0, 0.0f, -2.0f);
324             glVertex3f(0.0, -0.1f, -1.8f);
325 
326             glEnd();
327 
328             glPopMatrix();
329         }
330 
331         glPopMatrix();
332     }
333 
334     void MouseCallback(int button, int state, int x, int y)
335     {
336         mouse = { x, y, button, state };
337     }
338 
339     void MouseMotionCallback(int x, int y)
340     {
341         if ((mouse.button == GLUT_LEFT_BUTTON) && (mouse.state == GLUT_DOWN))
342         {
343             cameraAngleXY.first += (y - mouse.y) * 0.3f;
344             cameraAngleXY.second += (x - mouse.x) * 0.3f;
345             mouse.x = x;
346             mouse.y = y;
347         }
348         else if ((mouse.button == GLUT_MIDDLE_BUTTON) && (mouse.state ==
        GLUT_DOWN))
349         {
350             cameraDistance -= (y - mouse.y) * 0.1f;
351             mouse.y = y;
352         }
353 
354         glutPostRedisplay();
355     }
356 
357     void KbCallBack(unsigned char key, int x, int y)
358     {
359         switch (key)
360         {
361         case 's':
362             save("test1.pdf");
363             glPushMatrix();
364             glRotatef(180, 0, 1, 0);
365             save("test2.pdf");
366             glPopMatrix();
367             break;
368         }
369         glutPostRedisplay();
370         std::cout << key;
371     }
372 
373     void SpCallBack(int key, int x, int y)
374     {
375         const auto mods = glutGetModifiers();
376 
377         if (GLUT_ACTIVE_CTRL & mods)
378         {
379             switch (key)
```

```
380         {
381         case GLUT_KEY_LEFT:
382             cameraTransXY.first -= 0.5;
383             break;
384         case GLUT_KEY_RIGHT:
385             cameraTransXY.first += 0.5;
386             break;
387         case GLUT_KEY_DOWN:
388             cameraTransXY.second += 0.5;
389             break;
390         case GLUT_KEY_UP:
391             cameraTransXY.second -= 0.5;
392             break;
393         case GLUT_KEY_PAGE_UP:
394             break;
395 
396         }
397     }
398     else
399     {
400 
401         switch (key)
402         {
403         case GLUT_KEY_LEFT:
404             cameraAngleXY.second -= 0.5;
405             break;
406         case GLUT_KEY_RIGHT:
407             cameraAngleXY.second += 0.5;
408             break;
409         case GLUT_KEY_DOWN:
410             cameraAngleXY.first += 0.5;
411             break;
412         case GLUT_KEY_UP:
413             cameraAngleXY.first -= 0.5;
414             break;
415         case GLUT_KEY_PAGE_UP:
416             break;
417 
418         }
419     }
420     glutPostRedisplay();
421     std::cout << key;
422 }
423 
424 void ReshapeCallback(int w, int h)
425 {
426     screenWH = { w, h };
427 
428     glViewport(0, 0, (GLsizei)screenWH.first, (GLsizei)screenWH.second);
429     glMatrixMode(GL_PROJECTION);
430     glLoadIdentity();
431 
432     gluPerspective(45.0f, (float)(screenWH.first) / screenWH.second, 1.0f,
    100.0f); // FOV, AspectRatio, NearClip, FarClip
433 
434     // switch to modelview matrix in order to set scene
435     glMatrixMode(GL_MODELVIEW);
436     glLoadIdentity();
437 }
438 
439 void TimerCallback(int millisec)
440 {
441     glutTimerFunc(millisec, TimerCallback, millisec);
442     glutPostRedisplay();
443 }
444 
445 void MenuCallback(int item)
446 {
447     switch (item)
448     {
449     case 1:
450         drawG = !drawG;
451         break;
452     case 2:
453         drawG2 = !drawG2;
454         break;
455     case 3:
456         drawSphere = !drawSphere;
457         break;
458     case 4:
```

```
459             drawFaces = !drawFaces;
460             break;
461         case 5:
462             drawColors = !drawColors;
463             break;
464         case 6:
465             drawAxes = !drawAxes;
466             break;
467         case 7:
468             drawPoints = !drawPoints;
469             break;
470         case 8:
471             drawFacesSkeletons = !drawFacesSkeletons;
472             break;
473         default:
474             break;
475         }
476 
477         glutPostRedisplay();
478 }
479 
480 int main(int argc, char *argv[])
481 {
482         std::cout << "xyz: ";  std::cin >> fileName;
483         std::cout << "colors: ";  std::cin >> nColors;
484 
485         points = readXYZ(fileName + ".xyz");
486         edges = readTriangEdges(fileName + ".g");
487         edges2 = readTriangEdges(fileName + ".g2");
488         faces = readFaces(fileName + ".vor");
489         coloring = readColoring(fileName + "." + nColors + "c");
490 
491         for (auto& f : faces)
492         {
493             faces2.push_back(upgrade_face(f));
494         }
495 
496         sedges = std::set<std::pair<int, int>>(edges.begin(), edges.end());
497         sedges2 = std::set<std::pair<int, int>>(edges2.begin(), edges2.end());
498         std::set_difference(sedges2.begin(), sedges2.end(), sedges.begin(),
        sedges.end(),
499             std::inserter(sedges3, sedges3.end()));
500 
501         glutInit(&argc, argv);
502         glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH |
        GLUT_STENCIL);
503         glutInitWindowSize(1000, 800);
504         glutInitWindowPosition(0, 0);
505         glutCreateWindow((fileName + ".xyz").c_str());
506 
507         {
508             glutDisplayFunc(DisplayCallback);
509             glutMouseFunc(MouseCallback);
510             glutMotionFunc(MouseMotionCallback);
511             glutTimerFunc(100, TimerCallback, 100);
512             glutSpecialFunc(SpCallBack);
513             glutKeyboardUpFunc(KbCallBack);
514             glutReshapeFunc(ReshapeCallback);
515         }
516 
517         {
518             glClearColor(0, 0, 0, 0);
519             glShadeModel(GL_SMOOTH);
520             glEnable(GL_DEPTH_TEST);
521             glEnable(GL_POINT_SMOOTH);
522             glEnable(GL_LINE_SMOOTH);
523             glEnable(GL_POLYGON_SMOOTH);
524             //glEnable(GL_LIGHTING);
525             glEnable(GL_BLEND);
526             glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
527             glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
528             glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
529             glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
530             glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);
531             glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);
532 
533             //glPointSize(point_size);
534             //glLineWidth(line_width);
```

```
535     }
536
537     //light
538     {
539         GLfloat lightKa[] = { .3f, .3f, .3f, .9f };  // ambient light
540         GLfloat lightKd[] = { .7f, .7f, .7f, .9f };  // diffuse light
541         GLfloat lightKs[] = { .9f, .9f, .9f, .9f };   // specular light
542         glLightfv(GL_LIGHT0, GL_AMBIENT, lightKa);
543         glLightfv(GL_LIGHT0, GL_DIFFUSE, lightKd);
544         glLightfv(GL_LIGHT0, GL_SPECULAR, lightKs);
545
546         float lightPos[4] = { 0, 0, 1, 0 }; // directional light
547         glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
548         glEnable(GL_LIGHT0);
549     }
550
551     //menu
552     {
553         glutCreateMenu(MenuCallback);
554         glutAddMenuEntry("Show G", 1);
555         glutAddMenuEntry("Show G2", 2);
556         glutAddMenuEntry("Show sphere", 3);
557         glutAddMenuEntry("Show faces", 4);
558         glutAddMenuEntry("Show colors", 5);
559         glutAddMenuEntry("Show axes", 6);
560         glutAddMenuEntry("Show points", 7);
561         glutAddMenuEntry("Show faces skeletpons", 8);
562         glutAttachMenu(GLUT_RIGHT_BUTTON);
563     }
564
565     glutMainLoop();
566
567     return 0;
568 }
```