Национальная академия наук Беларуси
ГОСУДАРСТВЕННОЕ НАУЧНОЕ УЧРЕЖДЕНИЕ
«ИНСТИТУТ МАТЕМАТИКИ НАН БЕЛАРУСИ»

на правах рукописи
УДК 519.1

Головнёв
Александр Гарникович

Эффективные экспоненциальные алгоритмы решения модельных комбинаторных задач

по специальности 01.01.09 - дискретная математика и математическая кибернетика

Диссертация на соискание ученой степени кандидата физико-математических наук

Научный руководитель
доктор технических наук, профессор
Курбацкий Александр Николаевич

Минск, 2014

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

NP-трудные задачи — это задачи, часто встречающиеся на практике, для которых не известны эффективные алгоритмы решения. Исследование таких задач представляет большой интерес, т.к. существование эффективного алгоритма для любой из таких задач влечет существование эффективных алгоритмов для всех задач из класса NP. И, наоборот, если мы сможем доказать, что хотя бы для одной из таких задач не существует эффективного алгоритма, то это будет означать, что таких алгоритмов не существует для всех NP-трудных задач.

NP-трудная задача оптимизации — это NP-трудная задача, в которой среди всех возможных решений нужно выбрать оптимальное относительно какого-то параметра решение. Например, найти кратчайший цикл, проходящий через каждую вершину заданного взвешенного графа ровно один раз. Возможно, существует множество способов обойти граф, проходя по каждой вершине ровно один раз, но нас интересуют только те обходы, которые минимальны по весу. Эта задача называется задачей коммивояжёра. Время работы лучших известных алгоритмов для этой задачи в худшем случае примерно равно 2^n, где n — количество вершин графа. Большой интерес представляет исследование эффективных алгоритмов решения задач такого рода. Например, алгоритм, который решает задачу за время 1.45^n, позволит решать задачу для больших значений n. Интересно заметить, что тысячекратное увеличение вычислительных мощностей не дает такого ускорения во времени выполнения, как новый алгоритм с меньшей экспонентой. Существование полиномиального алгоритма для такой задачи повлечет равенство классов P и NP, а доказательство отсутствия такого алгоритма будет означать, что эти классы не равны.

В вычислительной сложности принято разделять задачи на эффективно разрешимые и сложные, такие как NP-трудные или #P-трудные задачи. Так как, возможно, сложные задачи не могут быть решены точно за полиномиальное время, исследователи начали изучать приближенные алгоритмы для поиска достаточно хороших решений. Однако, для многих задач даже приближенные алгоритмы не могут дать хороших результатов. Например, известно, что в предположении $P \neq NP$, полиномиальный алгоритм не может приблизить задачу MAX-2-SAT с фактором 0.9546.

Значительный рост вычислительных мощностей и доступной памяти повысил интерес к точным алгоритмам решения задач. В последние два десятилетия было разработано множество нетривиальных детерминированных и вероятностных экспоненциальных алгоритмов. Новые алгоритмы значитель-

но увеличивают размеры входных данных, на которых задачи могут быть решены точно. Известными примерами таких алгоритмов являются алгоритмы решения задачи о k-раскраске на графе из n вершин за время $O^*(2^n)$ (запись O^* скрывает полиномиальные факторы от длины входа), и алгоритмы решения задачи о k-выполнимости на формулах с n переменными за время $O^*((\frac{2(k-1)}{k})^n)$.

Задачи выполнимости (такие как SAT, CSP) формулируются на языке булевых или целочисленных формул, являющемся очень удобным для кодирования многих алгоритмических задач (таких, например, как составление расписаний, автоматическое доказательство теорем, задачи на графах). Задача пропозициональной выполнимости была первой задачей, для которой удалось доказать NP-полноту, и она до сих пор остается одной из самых известных NP-трудных задач. Данной задаче посвящена международная ежегодная конференция (The International Conference on Theory and Applications of Satisfiability Testing), проводящаяся уже более пятнадцати лет, а также научный журнал (Journal on Satisfiability, Boolean Modeling and Computation). Поскольку NP-трудные задачи часто возникают в практических приложениях (например, в распознавании изображений и при разработке микросхем), важное место в исследовании задачи выполнимости занимает разработка программ, решающих задачи выполнимости. Соревнования таких программ проводятся ежегодно.

Важным оптимизационным обобщением задачи выполнимости являются задача максимальной выполнимости (MAX-SAT, MAX-CSP). Задача MAX-SAT возникает в задачах комбинаторной оптимизации и искусственного интеллекта, также она входит в список пятнадцати самых популярных задач комбинаторной оптимизации. Многие задачи оптимизации могут быть сформулированы в терминах задач выполнимости. В частности, задачи MAX-2-SAT и MAX-2-CSP обобщают множество задач оптимизации на графах. Тривиальный алгоритм решения этих задач выполняется за время $O^*(2^n)$, а единственный известный алгоритм, улучшающий эту оценку, использует экспоненциальную память. С другой стороны, для многих частных случаев задач известны более эффективные алгоритмы.

Целью диссертационной работы является получение новых эффективных алгоритмов решения NP-трудных задач. С этой целью в работе рассматриваются NP-трудные задачи, обобщающие множество известных задач из класса NP. Таким образом, предложенные алгоритмы и идеи распространяются на решение многих известных задач оптимизации. В диссертационной работе мы рассматриваем три задачи, которые, на наш взгляд, имеют множество теоретических и практических применений в области алгоритмики. Для рассмат-

риваемых задач мы приводим алгоритмы, имеющие экспоненциальный выигрыш во времени выполнения, по сравнению с уже известными алгоритмами. Также мы рассматриваем частные случаи этих задач и приводим возможные эвристики и улучшения алгоритмов решения задач в частных случаях.

Для задач MAX-2-SAT и MAX-2-CSP известно несколько алгоритмов со временем работы $O^*(2^{nf(d)})$, где $f : \mathbb{R}^+ \to (0, 1)$ — медленно растущая функция, а d — средняя степень переменных входной формулы. Лучший известный алгоритм для MAX-2-CSP работает за время $O^*(2^{n(1-\frac{2}{d+1})})$ и использует полиномиальную память. В диссертации мы продолжаем эту линию исследований и предлагаем новые эффективные алгоритмы для задач MAX-2-SAT и MAX-2-CSP. Мы представляем технику получения новых верхних оценок такой формы из верхних оценок относительно количества клозов. С помощью этой техники мы улучшаем известные оценки до $O^*(2^{n(1-\frac{10/3}{d+1})})$ и $O^*(2^{n(1-\frac{3}{d+1})})$. Также мы показываем технику улучшения этих оценок для графов высоких средних степеней. Как следствие мы получаем простое доказательство оценки $O^*(2^{\frac{m}{5.263}})$ для задачи MAX-2-CSP, где m — количество клозов формулы. Эта оценка совпадает с лучшей известной оценкой относительно количества клозов.

Мы получаем асимптотически более сильные оценки в случае не очень плотных графов. Для этого мы доказываем оценку на сбалансированные разделители в графах сублинейной плотности. Этот результат приводит нас к алгоритму решения задачи MAX-2-CSP в случае $d = o(n)$ за время $O^*(2^{c_d n})$, где $c_d = 1 - \frac{2\alpha \ln d}{d}$ для константного $\alpha < 1$.

В диссертации мы рассматриваем и общий случай задачи MAX-SAT. В параметризованной версии задачи спрашивается о возможности выполнения хотя бы k клозов формулы. Задача естественным образом обобщает многие NP-трудные задачи, поэтому эффективные алгоритмы для MAX-SAT являются эффективными алгоритмами для целого класса задач. Лучшая предыдущая верхняя оценка 1.3695^k для MAX-SAT была доказана более 10 лет назад. В диссертации мы приводим новый алгоритм, доказывающий верхнюю оценку 1.3579^k.

Задача коммивояжёра (ЗК) заключается в поиске кратчайшего гамильтогова пути в полном взвешенном ориентированном графе на n вершинах. Задача коммивояжёра естественным образом возникает во многих транспортных и логистических задачах. Также задача имеет множество применений в производстве микросхем и секвенировании ДНК. К сожалению, в предположении $P \neq NP$, для ЗК не может существовать приемлемого приближающего алгоритма, имеющего полиномиальное время выполнения. Точные алгоритмы решения задачи либо используют экспоненциальную память, либо имеют

слишком долгое время выполнения ($O^*(4^n n^{\log n})$), что делает их неприменимыми на практике. Открытым вопросом является существование алгоритма со временем работы $O^*(2^n)$ и полиномиальной памятью. Мы предлагаем алгоритм, который для любого константного ε, находит $(1+\varepsilon)$-приближение общей задачи коммивояжёра за время $O^*(2^n)$, используя лишь полиномиальную память.

В задаче о кратчайшей общей надстроке (SCS) необходимо найти кратчайшую строку, содержащую каждую из n заданных строк $\{s_1, \ldots, s_n\}$ в качестве подстроки. Задача о надстроке является NP-трудной задачей и имеет множество практических применений, в том числе сборка генома, хранение и сжатие информации. По этой причине было разработано множество приближенных алгоритмов решения задачи о кратчайшей общей надстроке. Однако лучшим известным алгоритм точного решения задачи о надстроке является алгоритм со временем работы $O^*(2^n)$, где n — количество входных строк.

Текущая ситуация с точными алгоритмами для задачи о кратчайшей общей надстроке похожа на ситуацию с другими NP-трудными задачами, такими как задача выполнимости SAT, задача максимальной выполнимости MAX-SAT, задача о коммивояжере, задача о раскраске. Например, несмотря на большое количество усилий, лучшие известные алгоритмы для общих случаев этих задач выполняются за время $O^*(2^n)$ (где n — количество переменных/вершин). Но для частных случаев задач известны лучшие верхние оценки:

- SAT: $O(1.308^n)$ для 3-SAT; $O^*((2-2/k)^n)$ для k-SAT;
- MAX-SAT: $O(1.731^n)$ для MAX-2-SAT, $O(1.109^n)$ для $(n,3)$-MAX-2-SAT; $O((2-\varepsilon)^n)$ на формулах константной плотности;
- ЗК: $O(1.2186^n)$ в кубических графах; $O((2-\varepsilon)^n)$ в графах ограниченной максимальной/средней степени;
- Задача о раскраске: $O(1.3289^n)$ для 3-раскраски; $O((2-\varepsilon)^n)$ для графов ограниченной максимальной степени;

Улучшение оценки $O^*(2^n)$ для каждой из этих задач (а также для задачи о надстроке) остается открытым вопросом.

В диссертационной работе мы показываем, что задача о кратчайшей надстроке множества строк длины 3 может быть решена за время $O^*(3^{n/3})$ и полиномиальную память. Также мы показываем, что задача о кратчайшей надстроке множества строк константной длины r может быть решена вероятностным алгоритмом с односторонней ошибкой за время $O^*(2^{(1-c(r))n})$, где $c(r) = (1+2r^2)^{-1}$.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Связь работы с крупными научными программами и темами Тема диссертационной работы утверждена на заседании Учёного Совета факультета прикладной математики и информатики Белорусского государственного университета 14.12.2010 г. (протокол No 3), скорректирована на заседании Учёного Совета факультета прикладной математики и информатики Белорусского государственного университета 27.05.2014 г. (протокол No 8). Работа посвящена построению и анализу алгоритмов решения NP-трудных задач, что соответствует физико-математическим наукам и специальности 01.01.09 — дискретная математика и математическая кибернетика.

Цели и задачи исследования Целью диссертационной работы является получение новых эффективных алгоритмов решения NP-трудных задач. В ходе работы над диссертацией были поставлены следующие задачи:

1. Доказать новые верхние оценки для параметризованной задачи MAX-SAT.
2. Построить алгоритм, улучшающий известные оценки для задач MAX-2-SAT и MAX-2-CSP на формулах ограниченной средней степени.
3. Разработать алгоритм с асимптотически лучшей зависимостью от средней степени формул для задач MAX-2-SAT и MAX-2-CSP.
4. Построить алгоритм, дающий произвольное приближение задачи коммивояжера за время 2^n, используя лишь полиномиальную память.
5. Доказать новую верхнюю оценку для задачи о кратчайшей общей надстроке на строках длины 3.
6. Разработать алгоритм, решающий задачу о кратчайшей общей надстроке на строках константной длины за время, меньшее 2^n.

Объектом исследований являются NP-трудные задачи; *предметом исследований* — алгоритмы решения NP-трудных задач.

Положения, выносимые на защиту

1. Алгоритмы точного решения задачи максимальной выполнимости. Для параметризованной задачи MAX-SAT доказана верхняя оценка 1.3579^k. Построен алгоритм, решающий задачи MAX-2-SAT и MAX-2-CSP за время $O^*(2^{n(1-\frac{10/3}{d+1})})$ и $O^*(2^{n(1-\frac{3}{d+1})})$, соответственно. Также получены улучшенные оценки для графов высоких средних степеней. Для случая, когда граф ограничений не является очень разреженным или очень плотным, разработан алгоритм со временем выполнения $O(2^{c_d n})$, где $c_d = 1 - \frac{2\alpha \ln d}{d}$, где α — константа, меньшая единицы.

2. Алгоритм приближенного решения задачи коммивояжера. Построен алгоритм, дающий произвольное константное приближение задачи коммивояжера за время 2^n, используя лишь полиномиальную память.
3. Алгоритмы точного решения задачи о кратчашей общей надстроке. Для задачи о 3-надстроке доказана верхняя оценка $3^{n/3}$, улучшающая известную оценку 2^n. Разработан алгоритм Монте-Карло с односторонней ошибкой, решающий задачу о r-надстроке за время $2^{n\left(1-\frac{1}{1+2r^2}\right)}$.

Личный вклад соискателя В работе [94] диссертанту принадлежит идея и анализ алгоритма решения общей задачи MAX-SAT. В работе [92] диссертанту принадлежит анализ описанных алгоритмов. В работах [90, 95, 96] диссертантом выполнены анализ приближений на суффиксных графах и анализ нормальных конфигураций, соответственно. В работе [91] диссертанту принадлежит идея и анализ алгоритма решения задачи в случае графов константной плотности. Работы [89, 93, 97] полностью выполнены диссертантом.

Апробация результатов диссертации Основные результаты обсуждались на следующих конференциях и семинарах:

- Международная студенческая школа по информационному поиску и технологиям баз данных (RUSSIR/EDBT'11), Россия, 2011;
- Международный симпозиум International Symposium on Parameterized and Exact Computation (IPEC'11), Германия, 2011;
- Международный конгресс по информатике: информационные системы и технологии (CSIST'11), Беларусь, 2011;
- Международная конференция Computability in Europe: How the World Computes (CiE'12), Великобритания, 2012;
- Международный симпозиум International Symposium on Parameterized and Exact Computation (IPEC'12), Словения, 2012;
- Международный симпозиум Annual Symposium on Combinatorial Pattern Matching (CPM'13), Германия, 2013;
- Международный симпозиум International Symposium on Mathematical Foundations of Computer Science (MFCS'13), Австрия, 2013.

Результаты, лежащие в основе диссертации, также были доложены в

- Санкт-Петербургском Академическом университете,
- Санкт-Петербургском отделении Математического института РАН,
- Московском государственном университете,
- университете штата Пенсильвания (Pennsylvania State University) и
- университете Нью-Йорка (New York University).

Доклад на международном симпозиуме International Symposium on Parameterized and Exact Computation выиграл премию excellent student paper award.

Опубликованность результатов диссертации Основные результаты диссертации опубликованы в 9 научных работах, в том числе в 4 статьях в научных журналах в соответствии с п.18 Положения о присуждении ученых степеней и присвоении ученых званий в Республике Беларусь (общим объемом 2,5 авторского листа), а также в 5 сборниках материалов научных конференций.

Структура и объём диссертации Диссертация состоит из оглавления, введения, общей характеристики работы, 3 глав, заключения, библиографического списка. Полный объём диссертации составляет 84 страницы, включая 12 рисунков на 10 страницах, 4 таблицы. Библиографический список включает 97 наименований (включая собственные публикации соискателя).

ГЛАВА 1
ЗАДАЧА МАКСИМАЛЬНОЙ БУЛЕВОЙ ВЫПОЛНИМОСТИ

1.1 Постановка задачи

Задача MAX-SAT формулируется следующим образом: по заданной формуле в конъюнктивной нормальной форме (КНФ) определить максимальное число одновременно выполнимых клозов. MAX-2-SAT — задача определения максимального числа одновременно выполнимых клозов формулы, заданной в 2-КНФ (конъюнктивная нормальная форма с не более чем двумя литералами в каждом клозе).

Задача Maximum 2-constraint satisfaction (MAX-2-CSP) является обобщением задачи MAX-2-SAT. В терминах графов MAX-2-CSP формулируется следующим образом. Задан граф G на n вершинах и S — множество весовых функций $\{0,1\} \to \mathbb{Z}$ для каждой из вершин, и $\{0,1\}^2 \to \mathbb{Z}$ для каждого ребра (мы будем предполагать, что весовые функции могут быть вычислены и сохранены за полиномиальное время и память). Необходимо найти такое означивание переменных $\phi : V \to \{0,1\}$, которое максимизирует сумму значений S по всем вершинам и рёбрам:

$$\sum_{e=(v_1,v_2)\in E} S_e(\phi(v_1),\phi(v_2)) + \sum_{v\in V} S_v(\phi(v)). \tag{1.1}$$

Заметим, что задача MAX-2-SAT соответствует случаю, когда все функции S_e являются дизъюнкциями, а каждой вершине соответствует вес $S_v = 0$. Также, в графовом представлении задачи MAX-2-SAT могут появиться кратные рёбра. Далее, мы будем описывать задачи MAX-2-CSP графом G и набором весовых функций $S = S_v \cup S_e$, а задачи MAX-2-SAT — булевой формулой F. Через $Opt(G,S)$ мы обозначаем максимальное значение суммы (1.1) для пары (G,S) среди всех возможных означиваний $\phi : V \to \{0,1\}$.

Задачи MAX-SAT и MAX-2-SAT являются NP-трудными задачами оптимизации, обобщающими многие задачи на графах. Известно, что в предположении $P \neq NP$, даже задача MAX-2-SAT не может быть приближена с фактором 0.9546 [43]. Более того, на данный момент неизвестны алгоритмы, решающие задачу MAX-2-SAT за время меньшее $O^*(2^n)$ ($O^*(\cdot)$ скрывает полиномиальные множители от длины входа) с использованием полиномиальной памяти. Сильная гипотеза экспоненциального времени [19, 48] утвер-

ждает, что задача MAX-SAT не может быть решена за время $O(2-\varepsilon)^n$ для константного ε.

Далее мы будем использовать следующие обозначения: n — количество вершин в графе (переменных в формуле), m — количество рёбер (клозов). Под степенью вершины x ($deg(x)$) мы будем понимать количество клозов, в которые входит вершина. За Δ мы обозначаем максимальную степень вершины графа (формулы), $d = 2m/n$ — средняя степень вершины задачи MAX-2-SAT или MAX-2-CSP. Отметим, что при подсчете степени вершины мы учитываем кратные рёбра, то есть степень вершины — это количество 2-клозов, в которые входит соответствующая ей переменная. Мы говорим, что вершина y является соседом вершины x, если в графе есть ребро (x, y). Следуя работе [87], мы будем рассматривать клозы специального вида $\mathcal{T}$, которые выполнимы любым набором.

Под задачами (n, Δ)-MAX-2-SAT и (n, Δ)-MAX-2-CSP мы понимаем задачи, в которых степень каждой вершины не превосходит Δ. В частности, задача $(n, 3)$-MAX-2-SAT — это задача определения максимального количества одновременно выполнимых клозов формулы в 2-КНФ, если известно, что каждая переменная встречается не более, чем в трёх клозах входной формулы.

Пусть (G, S) — экземпляр задачи MAX-2-CSP (MAX-2-SAT, MAX-SAT), а v — вершина (переменная) G. $(G, S)[v]$ (или просто $G[v]$) обозначает экземпляр соответствующей задачи, полученный из (G, S) заменой v на 1. Аналогично, в $G[\neg v]$ мы заменяем v на 0. $F[x = \bar{y}]$ обозначает формулу, полученную заменой x и $\bar{x}$ на $\bar{y}$ и y, соответственно.

Вершинное покрытие графа $G = (V, E)$ — это такое подмножество вершин $C \subseteq V$, что для любого ребра графа $(u, v) \in E$, либо $u \in C$, либо $v \in C$. Дополнение вершинного покрытия $V \backslash C$ называется независимым множеством графа G. Два множества вершин $U \subset V, W \subset V$ называются попарно независимыми, если $U \cap W = \emptyset$ и в графе нет ребер $(u, w) \in E$ между ними $u \in U, w \in W$. Мы будем называть множество вершин $S \subseteq V$ сбалансированным разделителем мощности k, если $V \backslash S = (U, W)$ и

- U и W — попарно независимы,
- $|U| = \lfloor \frac{n-k}{2} \rfloor$ и $|W| = \lceil \frac{n-k}{2} \rceil$.

1.2 Известные результаты

В этом разделе мы перечислим известные алгоритмы решения рассматриваемых задач, также перечисленные алгоритмы собраны в Таблицах 1.1 и 1.2.

Williams [84] разработал алгоритм, решающий задачи MAX-2-SAT и MAX-2-CSP за время $O^*(2^{\frac{\omega n}{3}})$, где ω – экспонента перемножения матриц. Текущей лучшей оценкой является $\omega < 2.3727$ [85]. Но алгоритм [84] использует память $\Theta(2^{\frac{2n}{3}})$, что делает его неприменимым на практике. Вопрос о том, можно ли решить эти задачи за время меньшее $O^*(2^n)$, используя лишь полиномиальную память, до сих пор остается открытым. Однако, тривиальная оценка 2^n была улучшена для некоторых частных случаев рассматриваемых задач. Dantsin и Wolpert [25] показали, что MAX-SAT на формулах константной плотности (т.е. на формулах, в которых отношение количества клозов к количеству переменных ограничено константой) может быть решена быстрее, чем $O(2^n)$, но с экспоненциальной памятью. Kulikov и Kutzkov [57] разработали полиномиальный по памяти алгоритм для MAX-SAT со временем работы $O^*(2^{c_\rho n})$ для формул константной плотности ρ, где $c_\rho < 1$ — константа. Далее мы будем рассматривать только полиномиальные по памяти алгоритмы.

Fürer и Kasiviswanathan в статье [33] предлагают алгоритм решения задачи MAX-2-SAT за время $O^*(2^{n(1-\frac{1}{d-1})})$. Алгоритм, предложенный Scott и Sorkin в [77], решает задачу за время $O^*(2^{n(1-\frac{2}{d+1})})$.

Также известны эффективные алгоритмы решения частных случаев задачи MAX-2-SAT. Kojevnikov и Kulikov в работе [55] предлагают эффективный алгоритм решения задачи $(n, 3)$-MAX-2-SAT, время работы алгоритма составляет $O^*(2^{\frac{n}{6}})$. В работе [57] Kulikov и Kutzkov улучшили алгоритм и оценили время работы как $O^*(2^{\frac{n}{6.7}})$. Задача MAX-CUT также является частным случаем задачи MAX-2-CSP. Della Croce, Kaminski и Paschos в [23] привели алгоритм решения задачи MAX-CUT с оценкой на время работы $O^*(2^{n(1-\frac{2}{\Delta})})$

Отметим лучшие известные алгоритмы для рассматриваемых задач относительно количества клозов исходной формулы. Chen, Kanj в [21] предложили решение общей задачи MAX-SAT за время $O^*(2^{\frac{m}{2.465}})$. Для задач MAX-2-SAT и MAX-2-CSP лучшим из множества предложенных ([46, 87, 55, 77, 57, 14, 37]) алгоритмов относительно количества клозов является алгоритм [37]. Gaspers и Sorkin в [37] показывают верхние оценки $O^*(2^{\frac{m}{6.321}})$ и $O^*(2^{\frac{m}{5.263}})$ на задачи MAX-2-SAT и MAX-2-CSP, соответственно.

В параметризованной версии общей задачи MAX-SAT необходимо определить возможно ли одновременно выполнить k клозов. Сложность алгоритмов параметризованной версии задачи оценивается относительно k. Лучшая известная верхняя оценка 1.3695^k на время работы такого алгоритма была доказана Chen и Kanj [20]. Предыдущие результаты перечислены в Таблице 1.2.

Таблица 1.1 – Известные верхние оценки для задач MAX-2-SAT и MAX-2-CSP

время выполнения	задача	авторы	год
относительно n			
$2^{\frac{\omega n}{3}}$, экс. память	MAX-2-CSP	Williams [84]	2005
относительно n, формулы константной плотности			
$c^n, c < 2$, exp. space	MAX-SAT	Dantsin, Wolpert [25]	2006
$c^n, c < 2$	MAX-SAT	Kulikov, Kutzkov [57]	2009
относительно n и d			
$2^{n(1-\frac{1}{d-1})}$	MAX-2-CSP	Fürer, Kasiviswanathan [33]	2007
$2^{n(1-\frac{2}{d+1})}$	MAX-2-CSP	Scott, Sorkin [77]	2007
$2^{n(1-\frac{2}{\Delta})}$	MAX-CUT	Della Croce, Kaminski, Paschos [23]	2007
$2^{\frac{n}{6}}$	$(n,3)$-MAX-2-SAT	Kojevnikov, Kulikov [55]	2006
$2^{\frac{n}{6.7}}$	$(n,3)$-MAX-2-SAT	Kulikov, Kutzkov [57]	2009
относительно m			
$2^{\frac{m}{2.465}}$	MAX-SAT	Chen, Kanj [21]	2004
$2^{\frac{m}{6.321}}$	MAX-2-SAT	Gaspers, Sorkin [37]	2012
$2^{\frac{m}{5.263}}$	MAX-2-CSP	Gaspers, Sorkin [37]	2012

1.2.1 Новые результаты

В этой главе мы предложим три полиномиальных по памяти алгоритма, улучшающих верхние оценки на задачи максимальной выполнимости.

В разделе 1.3 мы докажем новую верхнюю оценку 1.3579^k на параметризованную задачу MAX-SAT. Этот результат является первым улучшением оценки 1.3695^k, доказанной Chen и Kanj более 10 лет назад.

В разделе 1.4 мы предложим алгоритм, решающий задачи MAX-2-SAT и MAX-2-CSP за время $O^*(2^{n(1-\frac{10/3}{d+1})})$ и $O^*(2^{n(1-\frac{3}{d+1})})$, соответственно. Предло-

Таблица 1.2 – Известные верхние оценки для параметризованной задачи MAX-SAT

Bound	Authors	Year
1.618^k	Mahajan, Raman [63]	1999
1.3995^k	Niedermeier, Rossmanith [69]	1999
1.3803^k	Bansal, Raman [8]	1999
1.3695^k	Chen, Kanj [20]	2002

женный алгоритм улучшает известную оценку $O^*(2^{n(1-\frac{2}{d+1})})$ [77]. Также мы покажем как улучшить оценку для графов высокой средней степени. Например, предложенный алгоритм решает задачи MAX-2-SAT и MAX-2-CSP за время $O^*(2^{n(1-\frac{3.45}{d+1})})$ и $O^*(2^{n(1-\frac{3.20}{d+1})})$ при $d \geq 8$. Т.к. задача MAX-CUT является частным случаем задачи MAX-2-CSP, алгоритм также дает рекордную верхнюю оценку $O^*(2^{n(1-\frac{3}{d+1})})$ для задачи о максимальном разрезе.

Алгоритм из раздела 1.5 доказывает асимптотически лучшую верхнюю оценку для задач MAX-2-SAT и MAX-2-CSP, чем все известные оценки, но только для случая, когда граф ограничений не является очень разреженным или очень плотным. Алгоритм использует графы-разделители ограниченного размера, существование которых мы доказываем, основываясь на недавнем результате Feige и Kogan [30]. Итоговое время выполнения алгоритма составляет $O(2^{c_d n})$, где $c_d = 1 - \frac{2\alpha \ln d}{d}$, $\alpha < 1$ — константа.

1.3 Параметризованная задача булевой выполнимости

1.3.1 Описание алгоритма

В этом разделе мы покажем, как по формуле в КНФ определить выполнимы ли в ней одновременно хотя бы k клозов за время 1.3579^k. Предложенный алгоритм основан на стандартных техниках расщепления, но использует новые правила упрощения, которые часто помогают избежать разбора случаев. Новая верхняя оценка основана на простом алгоритме для частного случая задачи MAX-SAT, где каждая переменная встречается не более 3 раз.

Через $\#_F(l)$ мы обозначаем количество вхождений в формулу литерала l. Клоз длины 1 мы называем единичным клозом. Мы будем говорить, что x является переменной типа (a, b), если литерал x встречается a раз, а литерал $\bar{x}$ — b раз. Также мы назваем переменную x $(k, 1)$-одиночкой ($(k, 1)$-неодиночной переменной), если x — переменная типа $(k, 1)$, а ее единственное отрицание содержится (не содержится) в одиночном клозе. Литерал l называется чистым, если в формуле ни разу не встречается литерал $\bar{l}$. Мы будем говорить, что литерал y доминирует литерал x, если все клозы, содержащие x, также содержат y. Два литерала называются противоречащими, если один из них является отрицанием второго. Мы используем "...", чтобы показать, что в клозе могут встречаться другие литералы. Например, клоз $(x \vee \bar{y} \vee \dots)$ содержит литералы $x, \bar{y}$ и, возможно, некоторые другие литералы.

Экземпляром параметризованной задачи MAX-SAT является пара (F, k). Мы говорим, что существует расщепление $(a_1, \dots, a_q)$, если мы можем эф-

фективно построить такие формулы $F_1, F_2, \ldots, F_q$, что ответ к исходной задаче может быть найден за полиномиальное время из ответов к задачам $(F_1, k-a_1), (F_2, k-a_2), \ldots, (F_q, k-a_q)$. Если l — литерал F, то простым примером расщепления может служить пара $(F[l], k - \#_F(l)), (F[\bar{l}], k - \#_F(\bar{l}))$. Известно, что алгоритм, использующий лишь расщепления из множества

$$(a_{1,1}, \ldots, a_{1,q_1}), (a_{2,1}, \ldots, a_{2,q_2}), \ldots, (a_{t,1}, \ldots, a_{t,q_t}),$$

где $a_{i,1} \leq a_{i,2} \leq \cdots \leq a_{i,q_i}$, $1 \leq i \leq t$, выполняется за время $O^*(c^k)$, где c — максимальный положительный корень многочлена

$$p(X) = \prod_{j=1}^{t}(X^{a_{j,q_j}} - \sum_{i=1}^{q_j} X^{a_{j,q_j}-a_{j,i}}).$$

Пусть $(a_1, \ldots, a_q)$, $a_1 \leq a_2 \leq \cdots \leq a_q$. Через $\tau(a_1, a_2, \ldots, a_q)$ (число расщепления) мы обозначаем единственный положительный корень многочлена $X^{a_q} - (X^{a_q-a_1} + X^{a_q-a_2} + \cdots + X^{a_q-a_q})$.

Простейшее расщепление по переменной высокой степени дает хорошее число расщепления. Поэтому тяжелым случаем для алгоритма является формула, состоящая из переменных низких степеней. Легко видеть, что переменные степени 2 могут быть удалены из формулы. Пусть x — переменная степени 3: $(x \vee A)(x \vee B)(\bar{x} \vee C)$, где A, B, C — дизъюнкции литералов. Если A и B состоят из одного литерала, то $(x \vee A)(x \vee B)(\bar{x} \vee C)$ может быть заменена формулой $(\bar{A} \vee B \vee C)(A \vee C)$. Но если A и B достаточно длинные, то мы можем произвести расщепление на две следующие ветки:

- заменить $(x \vee A)(x \vee B)(\bar{x} \vee C)$ на $(A \vee C)(B \vee C)$;
- установить все литералы из A, B, C на 0.

Корректность этих правил будет показана в следующем разделе.

Для решения задачи MAX-SAT, состоящей лишь из $(3,1)$- и $(4,1)$-одиночных переменных, мы вызываем эффективный алгоритм решения задачи покрытия множества, разработанный van Rooij и Bodlaender [81].

1.3.2 Правила упрощения и расщепления

Первое правило упрощения очевидно и не нуждается в доказательстве.

Правило упрощения 1. *Если l является чистым литералом или количество единичных клозов (l) не меньше количества клозов, содержащих $\bar{l}$, то l может быть присвоено значение* 1.

Правило упрощения 2. *Переменная степени ≤ 2 может быть исключена.*

Корректность: Если l — чистый литерал, то мы можем означить $l = 1$. Иначе, $F = G \wedge (l \vee A) \wedge (\bar{l} \vee B)$. Легко видеть, что MAX-SAT(F, k) = MAX-SAT$(F \wedge (A \vee B), k - 1)$. □

Правило упрощения 3. *Пары клозов* (x) *и* $(\bar{x})$ *могут быть удалены.*

Корректность: Очевидно, MAX-SAT$(F \vee (x) \vee (\bar{x}), k)$ = MAX-SAT$(F, k-1)$. Здесь не имеет значения встречается ли переменная x в формуле F. □

Правило упрощения 4. *Если две переменные* x *и* y *степени* 3 *встречаются вместе в* 3 *клозах, то все эти* 3 *клоза могут быть выполнены означиванием переменных* x *и* y*.*

Корректность: Можно выполнить 2 клоза означиванием только переменной x, последний клоз можно выполнить верным означиванием переменной y. □

Правило упрощения 5. *Пусть* x *— переменная степени* 3*:* $F = G \wedge (x \vee A) \wedge (x \vee B) \wedge (\bar{x} \vee C)$*. Если длина* A *или* B *меньше* 2*, то мы можем свести задачу к задаче с меньшим параметром.*

Корректность: Не умаляя общности, предположим, что длина A меньше 2. Если длина A равна 1, то A является литералом. Тогда

$$\text{MAX-SAT}(F, k) = \text{MAX-SAT}(G \wedge (\bar{A} \vee B \vee C) \wedge (A \vee C), k - 1).$$

Если A пусто, то мы можем означить $x = 1$, уменьшив параметр задачи на 2. □

Замечание 1.3.1. Все правила упрощения могут быть применены за полиномиальное время и уменьшают значение параметра k хотя бы на 1. Заметим, что некоторые правила упрощения выполняют несколько клозов формулы, в то время как другие правила заменяют существующие клозы на новые, уменьшая значения параметра (например, правила упрощения 2 и 5).

Правило расщепления 1. *Для любого литерала* l*, расщепление* $(F[l], k - \#_F(l)), (F[\bar{l}], k - \#_F(\bar{l}))$ *корректно.*

Правило расщепления 2. *Пусть* x *— переменная степени* 3*:* $F = G \wedge (x \vee A) \wedge (x \vee B) \wedge (\bar{x} \vee C)$*. Тогда следующее расщепление корректно:*

- $(G \wedge (A \vee C) \wedge (B \vee C), k - 1)$
- $(G', k-2)$*, где* G' *получена из* G *заменой всех литералов из* A, B, C *на* 0*.*

Корректность: Пусть $R = (A \vee C) \wedge (B \vee C)$. Если оптимальное означивание выполняет s клозов из R, где $s = 1, 2$, то в $F - G$ может быть выполнен $s + 1$ клоз, но не $s + 2$. Однако, если оптимальное означивание не выполняет ни одного клоза из R, то в $F - G$ два клоза могут быть выполнены означиванием $x = 1$. □

Следствие 1.3.1. *Если $A \cup B \cup C$ содержит противоречащие литералы, тогда можно рассматривать лишь первую ветку расщепления. Это означает, что задача (F, k) может быть сведена к задаче $(G \wedge (A \vee C) \wedge (B \vee C), k-1)$.*

Замечание 1.3.2. Под расщеплением по переменной мы будем понимать применение правила расщепления 1. Примение правила расщепления 2 к переменной x мы будем обозначать $BR2(x)$. Под $SRi(x)$, где $1 \leq i \leq 5$, мы понимаем применение правила упрощения i к переменной x.

Лемма 1.3.1 (Kulikov, Kutzkov [56])**.** *Если литерал y доминирует литерал x, то следующее расщепление корректно:*

- $x = 1, y = 0$;
- $x = 0$.

Доказательство. Рассмотрим означивание, в котором $x = y = 1$. Изменение значения переменной x не уменьшает количества выполненных клозов. Действительно, все клозы, которые могут быть выполнены означиванием $x = 1$, также выполняются означиванием $y = 1$. □

Лемма 1.3.2. *Если x — $(t, 1)$-неодиночная переменная, то расщепление по x является $(t, 2)$-расщеплением.*

Доказательство. Пусть y — сосед $\bar{x}$. Тогда в ветке $x = 1$ мы выполним хотя бы t клозов. В ветке $x = 0$ мы можем означить $y = 0$ и выполнить хотя бы 2 клоза. Теперь лемма следует из Леммы 1.3.1 с литералами $y, \bar{x}$ вместо y, x. □

Лемма 1.3.3. *Если x — переменная степени ≥ 6, то число расщепления по x не превосходит $\tau(1, 5)$.*

Доказательство. Лемма следует из факта, что $\tau(1, 5) > \tau(2, 4) > \tau(3, 3)$. □

1.3.3 Решение $(n, 3)$-MAX-SAT за время 1.273^k

Задача $(n, 3)$-MAX-SAT — частный случай задачи MAX-SAT, в котором каждая переменная встречается не более, чем в 3 клозах. В этом разделе мы предлаегаем простой алгоритм решения задачи $(n, 3)$-MAX-SAT за время 1.2721^k. Лучший известный алгоритм для этой задачи [20] имеет сложность 1.3247^k. Мы будем предполагать, что F является $(n, 3)$-MAX-SAT формулой.

Лемма 1.3.4. *Пусть x — переменная степени 3: $F = G \wedge (x \vee A) \wedge (x \vee B) \wedge (\bar{x} \vee C)$, и правила упрощения 1-4 не применимы к F. Если длина A меньше 2, то существует такое $(2, 4)$-расщепление, что полученные формулы также относятся к классу $(n, 3)$-MAX-SAT.*

Доказательство. Если A пусто, то мы можем означить $x = 1$. Иначе, применяя правило упрощения 5, мы удалим один клоз и получим новую формулу $F' = G \wedge (\bar{A} \vee B \vee C) \wedge (A \vee C)$. Переменные F' степени 4 могут встречаться только в A и C. Расщепление по переменной A произведет $(n, 3)$-MAX-SAT-формулы в обоих ветках. A — переменная степени 4, следовательно, расщепление по ней даст как минимум $(1, 3)$-расщепление ($\tau(2, 2) < \tau(1, 3)$). Т.к. один клоз уже был выполнен, итоговое расщепление будет хотя бы $(2, 4)$. □

Лемма 1.3.5 (Bliznets [18])**.** *Если каждая переменная формулы F встречается один раз отрицательно, два раза положительно, а все отрицательные литералы встречаются лишь в одиночных клозах, то MAX-SAT(F) может быть вычислено за полиномиальное время.*

Доказательство. Построим вспомогательный граф $G_F = (V, E)$. Для каждого клоза из положительных переменных введем вершину графа, соединим две вершины ребром, если соответствующие клозы имеют общую переменную. Тогда MAX-SAT(F) $= n + \nu(G_F)$, где $\nu(G_F)$ — размер максимального паросочетания графа G_F. □

Алгоритм 1 $(n, 3)$-MAX-SAT-Alg — решение задачи $(n, 3)$-MAX-SAT за время 1.2721^k.

Вход: F — экземпляр задачи $(n, 3)$-MAX-SAT.
Параметр: k — количество клозов, которые необходимо выполнить.
Выход: 1, если k клозов могут быть выполнены; иначе — 0.

1: применить правила упрощения 1–4
2: **if** если все отрицания — клозы одиночки **then**
3: вернуть результат (по Лемме 1.3.5).
4: **end if**
5: выбрать x, что $\bar{x}$ — не одиночка: $(x \vee A)(x \vee B)(\bar{x} \vee C), |C| > 0, |A| \leq |B|$
6: **if** $|A| \leq 1$ **then**
7: расщепиться в соответствии с Леммой 1.3.4
8: **end if**
9: **if** $|A| \geq 2$ **then**
10: расщепиться BR $1(x)$
11: **end if**

Теорема 1.3.1. *Алгоритм* $(n, 3)$-MAX-SAT-Alg *решает задачу* $(n, 3)$-*MAX-SAT за время* 1.2721^k.

Доказательство. По Лемме 1.3.5, время выполнения шага 3 полиномиально. Лемма 1.3.4 гарантирует $(2,4)$-расщепление на шаге 7. Расщепление на шаге 10 дает хотя бы $(2,4)$-расщепление. Действительно, из $|C| > 0$ и Леммы 1.3.1 следует, что в случае $x = 0$ мы можем выполнить хотя бы 2 клоза $(\bar{x} \vee C)$ и клоз с литералом $\bar{t}$, где t — литерал C. По правилу упрощения 4, в формуле есть 4 клоза, содержащих переменные из A. В ветке $x = 1$ два клоза выполнены означиванием x, и существуют переменные y, z, которые встречаются в формуле один или два раза. Значит, хотя бы два клоза содержат переменные y или z. Но тогда правило упрощения 2, примененное к переменным y, z из A, выполнит оба новых клоза. Следовательно, расщепление по x даст $(4,2)$-расщепление, и время работы алгоритма составляет $\max(\tau(2,4), \tau(3,3))^k = \tau(2,4)^k < 1.2721^k$. □

1.3.4 Удаление переменных степени 3

В этом разделе мы покажем, что формула, содержащая переменную степени 3, может быть упрощена так, что либо мы найдем достаточно хорошее расщепление, либо уменьши параметр k. Предположим, что x встречается в F три раза и правила упрощения не применимы к F. Пусть F содержит клозы $(x \vee A), (x \vee B), (\bar{x} \vee C)$, где A, B, C — дизъюнкции литералов. Мы будем рассматривать лишь случай, где $|A|, |B| \geq 2$, т.к. иначе мы либо можем присвоить $x = 1$, либо применить правило упрощения 5.

Определение 1.3.1. Под $LN(!A_1, \ldots, !A_k)$ мы будем понимать множество клозов F, содержащих отрицание какого-либо литерала из $A_1 \cup \cdots \cup A_k$

Также мы будем считать, что $A \cup B \cup C$ не содержит противоречащих литералов, иначе, по Следствию 1.3.1, формула может быть упрощена.

Лемма 1.3.6. *Пусть литералы $y, z \in A \cup B \cup C$, отрицание $\bar{y}$ встречается не менее двух раз и доминирует $\bar{z}$. Тогда в формуле существует $(2,4)$-расщепление.*

Доказательство. Напомним, что $\bar{y}, \bar{z}$ встречаются в клозах из $LN(!A, !B, !C)$. Рассмотрим расщепление $F[y], F[\bar{y}]$. Во втором случае два клоза выполнены $\bar{y}$, также z может быть означен 1, т.к. z стал чистым литералом. $z = 1$ выполняет хотя бы один из клозов $(x \vee A), (x \vee B), (\bar{x} \vee C)$. После этого мы можем применить правило упрощения $SR1(x)$, т.к. степень x не превосходит 2. Это правило выполнит не менее 4 клозов. В формуле $F[y]$ правило упрощения $SR2(x)$ уменьшит параметр хотя бы на 2. □

Лемма 1.3.7. *Пусть y, z — литералы из $A \cup B \cup C$, $\bar{y}$ встречается в формуле один раз и доминирует $\bar{z}$. Тогда в формуле существует $(3,3)$-расщепление.*

Доказательство. По аналогии с предыдущей леммой, в ветке $F[\bar{y}]$ мы можем означить $z = 1$. Литерал z встречается в формуле хотя бы два раза, т.к. $\bar{z}$ встречается только один раз. Следовательно, $z = 1$ выполняет хотя бы два клоза (напомним, что z и $\bar{z}$ не встречаются в одном клозе). Эти подстановки выполняют хотя бы 3 клоза. В ветке $F[y]$ мы выполним хотя бы 2 клоза, содержащих y, и один правилом упрощения $SR2(x)$, т.к. после означивания $y = 1$, один или два клоза содержат переменную x. Таким образом мы получаем $(3, 3)$-расщепление. □

Лемма 1.3.8. *Если* $|LN(!A, !B, !C)| < 3$*, то мы либо можем применить одну из двух предыдущих лемм, либо* $A = B = y \vee z$*. В первом случае мы имеем расщепление* $(2, 4)$, $(3, 3)$, $(1, 6)$ *или лучше, во втором случае значение параметра может быть уменьшено.*

Доказательство. $|A|, |B| \geq 2$. Либо $A \cup B$ содержит больше двух литералов, либо равно $y \vee z$. В первом случае три литерала должны встретиться в двух клозах, поэтому мы можем применить одну из двух предыдущих лемм. Во втором случае мы можем заменить клозы с переменной x на клоз $(y \vee z \vee C)$, уменьшив при этом значение параметра на 2. □

Теперь мы можем рассматривать только случай формул, в которых $|LN(!A, !B, !C)| > 2$. Если $|LN(!A, !B, !C)| > 3$, то упрощение $\mathrm{BR2}(x)$ сразу же дает $(1, 6)$-расщепление. Поэтому мы сосредоточимся на случае $|LN(!A, !B, !C)| = 3$.

Лемма 1.3.9. *Если* $|A \cup B \cup C| > 3$*, то применима одна из лемм 1.3.6,1.3.7. Следовательно, существует* $(2, 4)$- *или* $(3, 3)$*-расщепление.*

Доказательство. Хотя бы 4 отрицания литералов из $|A \cup B \cup C|$ должны встретиться в 3 клозах. □

Поэтому теперь мы можем рассматривать только случай $|A \cup B \cup C| \leq 3$.

Лемма 1.3.10. *Если* $min\{|A|, |B|\} \geq 3$*, то либо применима одна из Лемм 1.3.6,1.3.7, либо параметр задачи может быть уменьшен.*

Доказательство. Если $|A \cup B \cup C| > 3$, то по применима Лемма 1.3.9. Иначе, из $|A \cup B \cup C| \leq 3$ и $min\{|A|, |B|\} \geq 3$ следует, что $|A| = |B| = 3$ и $x \vee A = x \vee B = x \vee y_1 \vee y_2 \vee y_3$. Поэтому мы можем заменить $x \vee A, x \vee B, \bar{x} \vee C$ на $A \vee C$, уменьшив параметр на 2. □

Не умаляя общности, будем считать, что $x \vee A = x \vee y \vee z$.

Лемма 1.3.11. *Если для всех переменных* x *задачи* (F, k)*, которые встречаются в формуле три раза, выполняются следующие условия :*

- x *встречается в клозах* $x \vee A_x, x \vee B_x, \bar{x} \vee C_x$*,*

- $LN(!A_x, !B_x, !C_x) = 3$,

то мы либо можем уменьшить параметр задачи, либо получить одно из следующих расщеплений: $(3,3), (2,4)$.

Доказательство. Если применима хотя бы одна из предыдущих лемм, то следствие теоремы очевидно следует. Значит, мы можем выбрать такую переменную x степени 3, что соответствующий клоз $A_x = y \vee z$. Заметим, что если $\bar{y}$ встречается 3 раза, то применима одна из Лемм 1.3.6,1.3.7. Рассмотрим два случая: $\bar{y}$ встречается в формуле один или два раза.

Случай 1: $\bar{y}$ ***встречается*** 1 ***раз.***

Случай 1.1: У $\bar{y}$ ***есть сосед.***

F содержит следующие клозы:

$$(x \vee y \vee z), \quad (x \vee \dots), \quad (\bar{x} \vee \dots), \quad (\bar{y} \vee w \vee \dots).$$

В ветке $F[\bar{y}]$, по Лемме 1.3.1, w может быть присвоено значение 0, а после применимо правило SR5(x) или правило упрощения $SR2(x)$. Так мы уменьшаем параметр задачи на 3. В ветке $F[y]$ мы выполним хотя бы два клоза и, используя правило упрощения $SR2(x)$, уменьшим параметр на 1. Таким образом мы получаем $(3,3)$-расщепление. Следовательно, во всех оставшихся случаях должен присутствовать клоз $(\bar{y})$.

Случай 1.2: y ***встречается два раза и хотя бы один из них не с переменной*** x***.***

$F[y]$ и правило упрощения $SR2(x)$ выполняют хотя бы 4 клоза. В $F[\bar{y}]$, используя SR5(x), параметр задачи уменьшается на 2.

Случай 1.3: литерал y ***всегда встречается с переменной*** x***.***

Формула содержит следующие клозы:

$$(x \vee y \vee z), \quad (x \vee y \vee \dots), \quad (\bar{x} \vee y \vee \dots), \quad (\bar{y}).$$

Переменная y не встречается в оставшейся части формулы, иначе бы мы могли это рассматривать как случай 1.2. Достаточно рассмотреть случай $y = 0$, т.к. означивание $x = 1, y = 0$ не хуже означиваний $x = y = 1$ и $x = 0, y = 1$. Следовательно, мы можем выполнить один клоз и избавиться от одной переменной без расщеплений.

Случай 1.4: y ***встречается ровно дважды.***

Заметим, что клоз, содержащий $\bar{x}$, не содержит других литералов (иначе это был бы случай 1.1). Аналогичными рассуждениями мы можем получить, что либо $\bar{z}$ встречается дважды, либо $\bar{z}$ встречается один раз, z — дважды, и формула содержит клоз $(\bar{z})$. Рассмотрим два этих случая.

Случай 1.4.1: $\bar{z}$ ***встречаеися один раз,*** z ***— дважды.***

$$(x \vee y \vee z), \quad (x \vee \dots), \quad (\bar{x}), \quad (\bar{y}), \quad (\bar{z}).$$

$F[x]$, SR2(y), SR2(z) уменьшают параметр на 4. В ветке $F[\bar{x}]$ мы можем считать, что $y = \bar{z}$. То есть, мы получили $(4, 4)$-расщепление.

Случай 1.4.2: $\bar{z}$ ***встречается дважды.***

В этом случае формула содержит следующие клозы:

$$(x \vee y \vee z), \quad (x \vee \dots), \quad (\bar{x}), \quad (\bar{y}), \quad (\bar{z} \vee \dots), \quad (\bar{z} \vee \dots).$$

$F[z]$, SR2(x), SR2(y) уменьшают параметр на 3. В ветке $F[\bar{z}]$ мы можем считать, что $x = \bar{y}$. Так мы получаем $(3, 4)$-расщепление.

Случай 2: $\bar{y}$ ***встречается ровно два раза.***

Аналогичными рассуждениями мы можем получить, что $\bar{z}$ также встречается в формуле ровно два раза (иначе мы можем рассмотреть случай 1 с переменной z вместо переменной y). Следовательно, формула содержит следующие клозы:

$$(x \vee y \vee z), \quad (x \vee B), \quad (\bar{x} \vee \dots), \quad (\bar{y} \vee \dots), \quad (\bar{y} \vee \dots).$$

Предположим, что $y \in B$ (случай $z \in B$ аналогичен). В ветке $F[y]$ $x = 0$ выполнит 3 клоза. В ветке $F[\bar{y}]$ мы используем правило упрощения SR5(x), удалив при этом 3 клоза. Если y, z не встречаются в B, то $|B| < 2$ или $|A \cup B| \geq 4$, следовательно, мы можем применить одну из Лемм 1.3.6,1.3.7

□

Теорема 1.3.2. *Если x встречается в формуле ровно 3 раза, то либо существует $(1, 6)$-, $(2, 4)$- или $(3, 3)$-расщепление, либо параметр задачи может быть уменьшен.*

1.3.5 Решение MAX-SAT за время 1.3579^k

В этом разделе мы предлагаем простой алгоритм MAX-SAT-Alg, улучшающий верхнюю оценку на сложность параметризованной задачи MAX-SAT. Тяжелым случаем алгоритма является случай, когда все переменные формулы — это $(1,3)$- или $(1,4)$-одиночки. Мы сведем этот частный случай к задаче о минимальном покрытии множества. В задаче о минимальном покрытии множества дано универсальное множество U и семейство подмножеств $\mathcal{S}$, требуется найти минимальное количество множеств $S' \subset \mathcal{S}$, покрывающих U: $\bigcup_{S_i \in S'} S_i = U$. Под частотой элемента $e \in U$ $f(e)$ будем понимать количество множеств из $\mathcal{S}$, содержащих e.

Заметим, что алгоритм для задачи минимального доминирующего множества, предложенный van Rooij и Bodlaender [81], решает задачу о минимальном покрытии множества за время $1.28759^{k(U,\mathcal{S})}$, где $k(U,\mathcal{S}) = \sum_{e \in U} v(f(e)) + \sum_{S_i \in \mathcal{S}} w(|S_i|)$, v, w — весовые функции. Максимальные значения v и w равны 0.595723 и 1, соответственно. Также заметим, что для множеств размера ≤ 4, максимальное значение w равно 0.866888 (окончание раздела 3 в [81]). Тогда, согласно van Rooij and Bodlaender [81], верна следующая теорема.

Теорема 1.3.3 (van Rooij, Bodlaender [81]). *Если мощность множеств из S не превосходит 4, то алгоритм* MSC *решает задачу о минимальном покрытии множества за время* $O^*(1.28759^{0.595723|U|+0.866888|\mathcal{S}|}) = O^*(1.29^{0.6|U|+0.9|\mathcal{S}|})$.

Также мы будем использовать теорему Lieberherr and Specker [60], позже передоказанную Yannakakis [88].

Теорема 1.3.4 (Lieberherr, Specker [60]; Yannakakis [88]). *Если любые три клоза формулы F выполнимы, то хотя бы $\frac{2m}{3}$ выполнимы одновременно.*

Мы будем использовать Теорему 1.3.3 для задач с $m < 1.5k$, а Теорему 1.3.4 — для задач, где $m \geq 1.5k$.

Теорема 1.3.5. *Алгоритм* MAX-SAT-Alg *решает задачу MAX-SAT за время* $O^*(1.3579^k)$.

Доказательство. Ниже мы показываем, что в каждом случае алгоритм имеет число расщепления не больше, чем $\tau(5,10,1) < 1.3579$, из чего следует оценка $O^*(1.3579^k)$ на время выполнения алгоритма.

- Шаг 3. Если $deg(x) \geq 6$, то из Леммы 1.3.3 следует $(1,5)$-расщепление. $\tau(1,5) \approx 1.3248 < 1.3579$.
- Шаг 6. Если $deg(x) = 3$, то из Теоремы 1.3.2 следует $(1,6)$-расщепление. $\tau(1,6) \approx 1.2852 < 1.3579$.

Алгоритм 2 MAX-SAT-Alg — решение задачи MAX-SAT за время 1.3579^k.

Вход: F — формула MAX-SAT.

Параметр: k — количество клозов, которые необходимо выполнить.

Выход: 1, если k клозов могут быть выполнены одновременно; иначе — 0.

применить правила упрощения 1–5.
if есть x степени не менее 6 **then**
  расщепиться по вершине x
end if
if есть x степени 3 **then**
  расщепиться по x в соответствии с Теоремой 1.3.2
end if{%%Остались только переменные степени 4 и 5.}
if F содержит переменную x типа $(3,2)$, $(3,1)$-неодиночная или $(4,1)$-неодиночная **then**
  расщепиться по x
end if{%%Остались только одиночки и $(2,2)$-переменные.}
if F содержит $(2,2)$-переменную x **then**
  if у x есть сосед $(4,1)$-одиночка y, и хотя бы один из $x, \bar{x}$ не доминируется y **then**
    расщепиться по y
  else
    расщепиться по x
  end if
end if{%%Остались $(3,1)$-одиночки и $(4,1)$-одиночки.}
if $k \leq n$ **then**
  return 1
end if
if $m < 1.5k$ **then**
  return $k \leq \text{MSC}(F)$
end if
if остался клоз длины 2: $(x \vee y)$ **then**
  расщепиться $F[x, y]; F[x = \bar{y}]$.
else
  return 1
end if

- Шаг 9. $(3,2)$-переменная дает $\tau(3,2) \approx 1.3248 < 1.3579$. По Следствию 1.3.2, расщепление по $(4,1)$- или $(3,1)$-неодиночной ведет к рас-

щеплению хотя бы $\tau(3,2) \approx 1.3248 < 1.3579$.

- Шаг 13. x — $(2,2)$-переменная, y — $(1,4)$-одиночка, сосед x и y не доминируют одновременно x и $\bar{x}$. Расщепление по y даст $\tau(4,1)$, а следующая итерация в ветке $y=1$ будет содержать переменную степени 3 или меньше. Следовательно, итоговое число расщепления не превосходит $\tau(4+1,4+6,1) = \tau(5,10,1) < 1.3579$.
- Шаг 15. x — $(2,2)$-переменная. Соседи x — переменные степени 4, либо $x, \bar{x}$ доминируются y. Тогда обе ветки $F[x], F[\bar{x}]$ содержат переменную степени 3. По Теореме 1.3.2, переменная степени 3 дает $(1,6)$,- $(2,4)$- или $(3,3)$-расщепление. Поэтому возможны расщепления $\tau(2+1,2+6,2+1,2+6), \tau(2+2,2+3,2+2,2+3), \tau(2+3,2+3,2+3,2+3)$, а худшее из них — $\tau(3,8,3,8) \approx 1.3480 < 1.3579$.

Далее мы предполагаем, что все переменные это $(3,1)$- или $(4,1)$-одиночки.

- Шаг 19. Все переменные — одиночки, значит, мы можем выполнить n клозов означив все переменные 0. Если $k \leq n$, то это решает задачу.
- Шаг 22. Мы предполагаем, что все переменные встречаются 3 или 4 раза положительно и один раз отрицательно в единичном клозе. Заметим, что каждый клоз сейчас либо отрицательный одиночка, либо состоит только из положительных переменных. В этом случае существует оптимальное означивание переменных, которое выполняет все положительные клозы. Действительно, если какой-то положительный клоз не выполнен, то изменив значение любой его переменной, мы не уменьшим количество выполненных клозов. Это значит, что нам необходимо присвоить минимальное количество единиц переменным так, чтобы выполнить все положительные клозы. Эта задача может быть сведена к задаче о минимальном покрытии множества. Пусть U — множество положительных клозов ($|U| = m-n$), $\mathcal{S}$ содержит n множеств. Множество $S_i \in \mathcal{S}$ состоит из положительных клозов, содержащих переменную x_i. Сейчас мы хотим покрыть множество U минимальным количеством множеств из $\mathcal{S}$. Если t — решение задачи о минимальном покрытии множества, то максимальное количество одновременно выполнимых клозов равно $m-t$. Далее мы просто должны сравнить это значение с k и вернуть результат. По Теорме 1.3.3, алгоритм дял задачи о минимальном покрытии множества на множествах размера ≤ 4 работает за время $T(F) = O^*(1.29^{(0.6|U|+0.9|\mathcal{S}|)}) = O^*(1.29^{(0.6(m-n)+0.9n)})$. Также мы знаем, что $k > n$ и $m < 1.5k$. Поэтому $T(F) \approx 1.3574^k < 1.3579^k$.
- Шаг 25. Мы знаем, что формула содержит клозы $(\bar{x})$ и $(\bar{y})$. Если в формуле есть клоз $(x \vee y)$, то существует оптимальное означивание, вы-

полняющее клоз $(x \vee y)$. Действительно, иначе мы могли бы означить $x = 1$ и увеличить количество выполненных клозов. Значит, мы можем расщепиться на ветки $x = y = 1$ и $x = \bar{y}$. В первом случае мы выполним как минимум 3 клоза, т.к. x — это $(3,1)-$ или $(4,1)-$переменная. Во второй ветке мы выполняем клоз $(x \vee y)$, далее, по правилу упрощения 3, мы выполняем один из клозов (x) и $(\bar{x})$. Итак, мы получили $(2,3)$-расщепление.

- Шаг 27. Сейчас формула содержит $m \geq 1.5k$ клозов. В F нет клозов длины 2. Следовательно, любая тройка клозов выполнима. Из Теоремы 1.3.4 мы знаем о существовании означивания, выполняющего хотя бы $\frac{2m}{3} \geq k$ клозов.

□

Замечание 1.3.3. Чтобы найти одно из означиваний, которое выполняет k клозов, мы можем использовать свойство самосводимости задачи SAT. Пусть $\#_x$ — количество клозов, содержащих литерал x. Для каждой переменной x_i мы пробуем означивание $x_i = 1$. Если алгоритм сообщает, что мы можем выполнить $k - \#_x$ клозов новой формулы, то мы оставляем $x_i = 1$, иначе мы устанавливаем $x_i = 0$. Для поиска такого означивания нам потребуется лишь $O(n)$ вызовов алгоритма.

Рисунок 1.1 – Случаи выполнения алгоритма MAX-SAT-ALG

1.4 Задача выполнимости в графах ограниченной степени

1.4.1 Описание алгоритма

В этом разделе мы предлагаем алгоритм решения задач MAX-2-SAT и MAX-2-CSP с оценками на время работы $O^*(2^{n(1-\frac{10/3}{d+1})})$ и $O^*(2^{n(1-\frac{3}{d+1})})$, соответственно. Существует простое сведение задачи MAX-CUT к задаче MAX-2-CSP без увеличения количества и средней степени вершин. Используя это сведение, предложенный алгоритм дает новую верхнюю оценку $O^*(2^{n(1-\frac{3}{d+1})})$ и для задачи MAX-CUT. Также, мы приводим улучшенные оценки алгоритма для графов высоких средних степеней. Например, рассматриваемый алгоритм решает задачу MAX-2-SAT (MAX-2-CSP) за время $O^*(2^{n(1-\frac{3.45}{d+1})})$ ($O^*(2^{n(1-\frac{3.20}{d+1})})$) при $d \geq 8$.

Для начала рассмотрим алгоритм, предложенный Scott и Sorkin [77] для задачи MAX-2-CSP. Основываясь на результате Alon, Kahn и Seymour [2] о размере индуцированного дерева в графе средней степени $d \geq 2$, авторы используют тот факт, что задача MAX-2-CSP разрешима за полиномиальное время в случае, когда соответствующий граф условий является деревом. Это наблюдение следует из того, что в MAX-2-CSP вершины степени 2 могут быть удалены (см. Лемму 1.4.1). Это правило упрощения играет ключевую роль в эффективных алгоритмах для задач MAX-2-SAT и MAX-2-CSP. Однако, удаление вершин степени 2 — это не единственный прием в построении алгоритмов для MAX-2-SAT. Лучшие известные оценки получены сложными алгоритмами, использующими множество правил упрощения, и оцененными с помощью нестандартных мер сложности. Некоторые из этих правил упрощений, например, запоминание клозов [57], подходят только для задачи MAX-2-SAT, но не MAX-2-CSP. Основная идея предложенного в этом разделе алгоритма заключается в комбинировании всех этих техник, но используя их лишь неявно. Алгоритм будет производить расщепления по переменным высоких степеней, удаляя все смежные ребра. Мы покажем, что таким образом граф будет становиться более разреженным. Мы будем продолжать расщепдения до тех пор, пока граф не станет достаточно разреженным, чтобы запуск известных алгоритмов относительно количества клозов на нем был эффективней алгоритма полного перебора. Таким образом, мы предлагаем простой алгоритм, использующий сложные техники анализа алгоритмов лишь как черный ящик. Т.к. для задачи MAX-2-SAT известны более эффективные алгоритмы, чем для задачи MAX-2-CSP, мы получим лучшие оценки для задачи MAX-2-SAT.

Рассмотрим задачу MAX-2-SAT на 2-КНФ формуле F с m 2-клозами и n переменными. Стандартная мера сложности (количество клозов) формулы F может быть записана следующим образом:

$$m(F) = \sum_{i=1} \frac{in_i}{2},$$

где n_i равно количеству переменных степени i.

Kojevnikov и Kulikov [55] показали, что альтернативная мера сложности для задачи MAX-2-SAT:

$$\mu(F) = \sum_{i=1} \alpha_i n_i,$$

где $\alpha_i \leq i/2$, лучше подходит для анализа времени выполнения некоторых алгоритмов расщепления. Из-за большого количества разнообразных правил упрощения переменных низких степеней, эти переменные больше влияют на эффективность алгоритма, чем переменные высоких степеней. В новом алгоритме для задач MAX-2-SAT и MAX-2-CSP мы также будем использовать похожую меру для доказательства новых верхних оценок.

1.4.2 Алгоритм решения задач MAX-2-SAT и MAX-2-CSP

Напомним, что задача (n, Δ)-MAX-2-SAT ((n, Δ)-MAX-2-CSP) — это частный случай задачи MAX-2-SAT (MAX-2-CSP), в котором степень каждой переменной (максимальное количество 2-клозов, в которые входит переменная) не превосходит Δ. В этом разделе мы предложим простой алгоритм решения задач (n, Δ)-MAX-2-SAT и (n, Δ)-MAX-2-CSP, и оценим время его работы относительно Δ. Оценку на время работы алгоритма для задач (n, Δ)-MAX-2-SAT и (n, Δ)-MAX-2-CSP мы получим из оценок на время работы алгоритмов для задач $(n, \Delta - 1)$-MAX-2-SAT и $(n, \Delta - 1)$-MAX-2-CSP. Следствием будет простой алгоритм решения задач MAX-2-SAT и MAX-2-CSP с верхними оценками на время работы $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ и $O^*(2^{n(1-\frac{3}{\Delta+1})})$, соответственно.

Лемма 1.4.1. *Пусть F – формула задачи MAX-2-SAT или задачи MAX-2-CSP, содержащая вершину и степени не более 2. Тогда F за полиномиальное время может быть преобразована в формулу F', что*

1. $deg_{F'}(u) = 0$,
2. *для любой вершины v $deg_{F'}(v) \leq deg_F(v)$,*
3. *$Opt(F)$ может быть вычислено из $Opt(F')$ за полиномиальное время.*

Для задачи MAX-2-SAT доказательство леммы можно найти в статье [55], для задачи MAX-2-CSP — в статье [37]. Этот результат позволяет на каждой итерации алгоритма считать, что степень каждой вершины графа не менее 3.

Алгоритм будет производить расщепление по вершинам максимальной степени Δ до тех пор, пока не получит граф с максимальной степенью вершин меньше Δ. Так можно продолжать, пока степень графа не станет равной 3. Граф степени три соответствует задаче $(n,3)$-MAX-2-SAT или задаче $(n,3)$-MAX-2-CSP. Для решения этих задач известны эффективные алгоритмы.

Пусть n_i – число вершин степени i для $i \in \{3,\dots,\Delta\}$. Рассмотрим задачу (n,Δ)-MAX-2-SAT ((n,Δ)-MAX-2-CSP). Введём различные веса для вершин с различными степенями. Так для вершины степени i вес будет равен α_i. Отметим, что благодаря правилам упрощения вершины степени 2, на каждом шаге алгоритма степень каждой вершины не менее 3. Мы хотим найти такие $\alpha_i, i = 3,\dots,\Delta$, что время работы алгоритма на формуле F будет равно $poly(|F|)\cdot 2^{\alpha_3 n_3+\dots+\alpha_\Delta n_\Delta}$.

Пусть существуют α_3, α_4, …, $\alpha_{\Delta-1}$ и алгоритм A решения задачи $(n,\Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP) с оценкой на время работы $2^{\alpha_3 n_3+\dots+\alpha_{\Delta-1} n_{\Delta-1}}$. Рассмотрим следующий алгоритм решения задачи (n,Δ)-MAX-2-SAT ((n,Δ)-MAX-2-CSP). Входом алгоритма является формула F, параметром — алгоритм A, решающий задачу $(n,\Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP).

Алгоритм 3 MetaAlg — решение задачи (n,Δ)-MAX-2-SAT.

Вход: F — экземпляр задачи (n,Δ)-MAX-2-SAT или (n,Δ)-MAX-2-CSP.
Параметр: Алгоритм A, решающий задачу $(n,\Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP).
Выход: $Opt(F)$.

```
применить правила упрощения ко всем переменным степени < 3
if F не содержит клозов, кроме 𝒯 then
    return количество таких клозов
end if
if максимальная степень переменной в F меньше Δ then
    return A(F)
end if
выбрать вершину x максимальной степени Δ
return max(MetaAlg(A, F[x]), MetaAlg(A, F[¬x]))
```

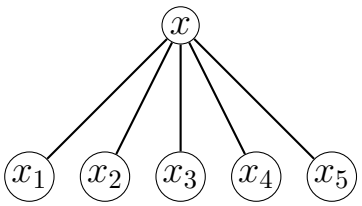

Рисунок 1.2 – Шаг 8 работы алгоритма MetaAlg

Лемма 1.4.2. *Пусть* $\Delta > 3$, $\alpha_i < 1$ *и*

$$\delta = \min(\alpha_\Delta - \alpha_{\Delta-1}, \alpha_{\Delta-1} - \alpha_{\Delta-2}, \ldots, \alpha_4 - \alpha_3, \alpha_3).$$

$$\delta \geq \frac{1 - \alpha_\Delta}{\Delta}, \tag{1.2}$$

то время работы алгоритма MetaAlg *на задаче* (n, Δ)*-MAX-2-SAT* ((n, Δ)*-MAX-2-CSP) не превосходит* $O^*(2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta})$.

Доказательство. Введем меру сложности формул $\mu = \alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta$. Обозначим через $T(n_3, \ldots, n_\Delta)$ время работы алгоритма на формуле, в которой n_3 вершин степени 3, n_4 вершин степени 4, $\ldots$, n_Δ вершин степени Δ. Если в графе нет вершин степени Δ (т.е. $n_\Delta = 0$), то алгоритм MetaAlg вызовет алгоритм A. Тогда по условию:

$$T(n_3, \ldots, n_\Delta) \leq 2^{\alpha_3 n_3 + \ldots + \alpha_{\Delta-1} n_{\Delta-1}} = 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta}.$$

Пусть в графе существует вершина степени Δ. Тогда алгоритм MetaAlg на шаге 8 выберет вершину степени Δ и произведёт расщепление по ней. Алгоритм вызовет себя рекурсивно два раза. Мы хотим показать, что в обоих вызовах мера μ уменьшится хотя бы на 1.

Действительно, мера уменьшится на α_Δ, т.к. мы расщепили одну вершину степени Δ. Для каждой из Δ соседних вершин $x_1, \ldots, x_\Delta$ степень вершины уменьшилась хотя бы на 1. Следовательно, мера каждой из этих вершин уменьшилась хотя бы на δ, т.к. δ — это минимальное изменение меры вершины при уменьшении ее степени. Заметим, что при упрощении вершины степени 2, степени остальных вершин не возрастают. Следовательно, не возрастает и мера сложности.
Из (1.2), $\delta\Delta + \alpha_\Delta \geq 1$. Следовательно, мера уменьшилась хотя бы на 1.
Тогда

$$T(n_3, \ldots, n_\Delta) \leq 2 \cdot 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta - 1} + poly(|F|) \leq 2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta} + poly(|F|),$$

где $|F|$ – длина исходной формулы. Таким образом, время работы алгоритма MetaAlg на задаче (n, Δ)-MAX-2-SAT ($(n, \Delta - 1)$-MAX-2-CSP) не превосходит $O^*(2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta})$. □

Теперь мы можем оценить время работы алгоритма MetaAlg на задаче (n, Δ)-MAX-2-SAT ((n, Δ)-MAX-2-CSP) как $O^*(2^{\alpha n})$, где $\alpha = \max(\alpha_\Delta, \dots \alpha_3)$. Но из условий $\alpha_i < 1$ и (1.2) следует, что $\alpha_i > \alpha_j$ при $i > j$. То есть $\alpha = \alpha_\Delta$.

Из работы [57] мы знаем о существовании алгоритма для решения задачи $(n, 3)$-MAX-2-SAT с верхней оценкой на время выполнения $O^*(2^{n/6.7})$.
Из правил упрощения для вершин степени 2 следует алгоритм для задачи $(n, 3)$-MAX-2-CSP с верхней оценкой на время выполнения $O^*(2^{n/4})$. Действительно, производя на каждом шаге алгоритма расщепление по вершине степени 3, мы будем уменьшать общее количество вершин в графе хотя бы на 4.

Следствие 1.4.1. *Следующий алгоритм решает задачу MAX-2-SAT (MAX-2-CSP) за время* $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ *(*$O^*(2^{n(1-\frac{3}{\Delta+1})})$*).*

Алгоритм 4 SimpleAlg — решение задачи MAX-2-SAT (MAX-2-CSP).

Вход: F — экземпляр задачи MAX-2-SAT или MAX-2-CSP.
Выход: $Opt(F)$.

```
1: применить правила упрощения ко всем переменным степени меньше 3
2: if F не содержит клозов, кроме 𝒯 then
3:    return количество таких клозов
4: end if
5: if максимальная степень переменных в F равна 3 then
6:    return результат работы известного алгоритма на задаче (n, 3)-MAX-
      2-SAT или (n, 3)-MAX-2-CSP, соответственно
7: end if
8: выбрать вершину x максимальной степени Δ
9: return max(SimpleAlg(A, F[x]), SimpleAlg(A, F[¬x]))
```

Доказательство. Алгоритм SimpleAlg получен из алгоритма MetaAlg. В качестве параметра A SimpleAlg принимает сам себя при $i > 3$ и описанные алгоритмы при $i = 3$. Для оценки времени работы алгоритма выберем α_i, удовлетворяющие (1.2).
Выше было сказано, что задача $(n, 3)$-MAX-2-SAT ($(n, 3)$-MAX-2-CSP) решается алгоритмом SimpleAlg за время $O^*(2^{n/6.7})$ ($O^*(2^{n/4})$). Следовательно, $\alpha_3 \geq \frac{1}{6.7}$ ($\alpha_3 \geq \frac{1}{4}$).
Пусть $\Delta = 4$. Выберем α_4 так, чтобы $\max(\alpha_3, \alpha_4)$ был минимален и выполнялось (1.2).
Для задачи $(n, 4)$-MAX-2-SAT получим $\alpha_3 = 1/6$, $\alpha_4 = 1/3$.
Для задачи $(n, 4)$-MAX-2-CSP получим $\alpha_3 = 1/4$, $\alpha_4 = 2/5$.

Таким образом, верны оценки $O^*(2^{n/3})$ и $O^*(2^{2n/5})$ на время работы SimpleAlg на задачах $(n,4)$-MAX-2-SAT и $(n,4)$-MAX-2-CSP, соответственно.

Теперь, пусть $\Delta > 4$. Положим

$$\alpha_i = \frac{1 + i \cdot \alpha_{i-1}}{i+1}. \tag{1.3}$$

Сформулируем некоторые свойства последовательности α_i:

- $\alpha_i = 1 - 4\frac{1-\alpha_3}{i+1}$.
  Вычислим α_i явно.

$$\alpha_i = \frac{1 + i \cdot \alpha_{i-1}}{i+1} = \frac{1 + i \cdot \frac{1+(i-1)\alpha_{i-2}}{i}}{i+1} = \frac{2 + (i-1)\alpha_{i-2}}{i+1} = \ldots = \frac{i - 3 + 4\alpha_3}{i+1} = 1 - 4\frac{1-\alpha_3}{i+1}. \tag{1.4}$$

- $\alpha_i < 1$.
  Для $i = 3$ это верно. Для $i > 3$ из (1.4) следует, что $\alpha_i < 1$.
- α_i возрастает с ростом i.
  Действительно, из (1.3):

$$\alpha_i = \frac{1 + i\alpha_{i-1}}{i+1} > \frac{\alpha_{i-1} + i\alpha_{i-1}}{i+1} = \alpha_{i-1}$$

- $\alpha_i - \alpha_{i-1} = \frac{1-\alpha_i}{i}$.
  Из (1.3) $(i+1)\alpha_i - i\alpha_{i-1} = 1$. Следовательно, $i(\alpha_i - \alpha_{i-1}) = 1 - \alpha_i$.
  Откуда и получеам требуемое равенство.

Для задачи MAX-2-SAT получим: $\alpha_\Delta = 1 - \frac{10/3}{\Delta+1}$.

Для задачи MAX-2-CSP: $\alpha_\Delta = 1 - \frac{3}{\Delta+1}$.

Покажем, что выбранные α_i удовлетворяют условию леммы.

Первое, SimpleAlg решает задачу $(n, \Delta-1)$-MAX-2-SAT ($(n,\Delta-1)$-MAX-2-CSP) за время $2^{\alpha_3 n_3 + \ldots + \alpha_{\Delta-1} n_{\Delta-1}}$ по построению.

Осталось показать, что выполняется (1.2). Сначала покажем, что $\alpha_3 \geq \frac{1-\alpha_\Delta}{\Delta}$ для $\Delta \geq 4$.

$$\frac{1-\alpha_\Delta}{\Delta} \leq \frac{1 - 1 + \frac{10/3}{\Delta+1}}{\Delta} \leq \frac{10}{3\Delta(\Delta+1)} \leq \frac{10}{3 \cdot 4 \cdot (4+1)} \leq \frac{1}{6} \leq \alpha_3.$$

Теперь покажем, что $\alpha_i - \alpha_{i-1} \geq \frac{1-\alpha_\Delta}{\Delta}$ для $i \leq \Delta$.

Из свойств последовательности α_i мы знаем, что $\alpha_i - \alpha_{i-1} = \frac{1-\alpha_{i-1}}{i+1}$. Т.к. α_i возрастает, то при $i < \Delta$ $\frac{1-\alpha_{i-1}}{i+1} > \frac{1-\alpha_\Delta}{\Delta}$. Следовательно, $\alpha_i - \alpha_{i-1} = \frac{1-\alpha_{i-1}}{i+1} > \frac{1-\alpha_\Delta}{\Delta}$ при $i < \Delta$.

При $i = \Delta$, $\alpha_i - \alpha_{i-1} = \frac{1-\alpha_\Delta}{\Delta}$. □

Следовательно, мы построили алгоритм со временем работы $O^*(2^{n(1-\frac{10/3}{\Delta+1})})$ для задачи MAX-2-SAT, и $O^*(2^{n(1-\frac{3}{\Delta+1})})$ — для MAX-2-CSP. Отметим, что для графа степени 3 нам достаточно вызывать простой алгоритм с верхней оценкой $2^{n/6}$ из работы [55], а не лучший известный из работы [57]. От использования лучшего алгоритма в данном случае мы не получили бы выигрыша, т.к. в условии леммы (1.2) заложено ограничение на $\alpha_3 \geq \frac{1-\alpha_4}{4}$. Отсюда при $\alpha_3 < \frac{1}{6}$ растет α_4 и, следовательно, все последующие α_i.
Интересно отметить, что построенный алгоритм можно улучшить, если для малых Δ использовать алгоритмы из [37] со временем работы $O^*(2^{\frac{m}{6.321}})$ для MAX-2-SAT и $O^*(2^{\frac{m}{5.263}})$ для MAX-2-CSP. Отметим, что построенный алгоритм уже улучшает известные на данный момент алгоритмы относительно Δ из [33] и [77] с оценками $O^*(2^{n(1-\frac{1}{\Delta-1})})$ и $O^*(2^{n(1-\frac{2}{\Delta+1})})$.

1.4.3 Анализ алгоритма

Отличие результатов работ [33] и [77] от построенного алгоритма в том, что они сохраняют те же оценки при переходе от максимальной степени к средней степени графа. Мы же пока построили алгоритм только для максимальной степени Δ. Данный раздел посвящен переходу от оценки сложности относительно Δ к оценке сложности относительно d. Мы покажем, что можно перейти от оценок относительно Δ к оценкам относительно d, не изменив верхнюю оценку на время работы алгоритма. Как следствие мы получим оценки для SimpleAlg алгоритма. Неформально, теорема говорит о том, что если максимальная сложность алгоритма достигается при степенях вершин близких к средней степени, то сложность алгоритма не увеличится при переходе от оценки относительно Δ к оценке относительно d. То есть, если худший случай для алгоритма – это случай, когда все степени вершин одинаковы, то оценки сложности относительно Δ и d совпадают. В следствии мы заметим, что это естественное свойство выполняется для алгоритма SimpleAlg.

Мы будем использовать неравенство Йенсена в следующей форме:
Теорема 1.4.1. *Если f — вогнутая функция, $x_1, x_2, \ldots, x_n$ из области определения f, n_i — положительные числа, то:*

$$\frac{\sum n_i f(x_i)}{\sum n_i} \leq f(\frac{\sum n_i x_i}{\sum n_i}).$$

Следствие 1.4.2. *Для произвольных $c, n_i > 0$, из $\alpha_i \leq 1 - \frac{c}{i+1}$ и $\sum n_i = n$ следует*

$$n_3\alpha_3 + \ldots + n_\Delta\alpha_\Delta \leq n(1 - \frac{c}{d+1}),$$

где $d = \frac{\sum i n_i}{n}$.

Доказательство. Легко видеть, что для положительного c, $f(x) = 1 - \frac{c}{x+1}$ является вогнутой функцией при $x \geq 3$. По Теореме 1.4.1,

$$n_3\alpha_3 + \ldots + n_\Delta\alpha_\Delta \leq (\sum n_i) \cdot f(\frac{\sum n_i i}{\sum n_i}) = nf(d) = n(1 - \frac{c}{d+1}).$$

□

Следствие 1.4.3. *Алгоритм* SimpleAlg *решает задачу MAX-2-SAT (MAX-2-CSP) за время* $O^*(2^{n(1-\frac{10/3}{d+1})})$ $(O^*(2^{n(1-\frac{3}{d+1})}))$.

1.4.4 Улучшения алгоритма

Построенный нами в разделе 1.4.2 алгоритм на каждой итерации удалял вершину максимальной степени Δ. Поэтому мера сложности убывала хотя бы на $\alpha_\Delta + \Delta\delta$, где δ — минимальное возможное изменение меры при уменьшении степени вершины на 1. То есть, $\delta = \min(\alpha_\Delta - \alpha_{\Delta-1}, \alpha_{\Delta-1} - \alpha_{\Delta-2}, \ldots, \alpha_4 - \alpha_3, \alpha_3)$. В (1.4.1) мы показали, что $\delta = \alpha_\Delta - \alpha_{\Delta-1}$, и в соответствии с этим выбирали α так, чтобы

$$\alpha_\Delta + \Delta \cdot \delta = \alpha_\Delta + \Delta(\alpha_\Delta - \alpha_{\Delta-1}) = 1.$$

Поэтому мы выбирали $\alpha_\Delta = \frac{1+\Delta\alpha_{\Delta-1}}{\Delta+1}$.
Теперь мы хотим улучшить полученные оценки. Для этого сделаем так, чтобы на каждой итерации у удаляемой вершины была хотя бы одна соседняя вершина степени меньше Δ. Тогда мы сможем выбирать α_Δ исходя из следующего равенства:

$$\alpha_\Delta + (\Delta - 1)(\alpha_\Delta - \alpha_{\Delta-1}) + (\alpha_{\Delta-1} - \alpha_{\Delta-2}) = 1.$$

Тогда:

$$\alpha_\Delta = \frac{1 + \alpha_{\Delta-2} + (\Delta - 2)\alpha_{\Delta-1}}{\Delta}. \tag{1.5}$$

Приведем алгоритм, для которого будут верны такие оценки.

Алгоритм Max2Alg отличается от алгоритма SimpleAlg в следующем:

- На шаге 5 Max2Alg обрабатывает несвязные графы.
- На шаге 11 Max2Alg пытается выбрать вершину, у которой есть сосед степени меньше Δ.

Лемма 1.4.3. *Если*

1. *Алгоритм, вызываемый на шаге 9, решает задачу* $(n, 3)$*-MAX-2-SAT* $((n, 3)$*-MAX-2-CSP) за время* $O^*(2^{\alpha_3 n})$,

Алгоритм 5 Max2Alg — решение задачи MAX-2-SAT (MAX-2-CSP).

Вход: F — экземпляр задачи MAX-2-SAT или MAX-2-CSP.
Выход: $Opt(F)$.

1: применить правила упрощения ко всем переменным степени меньше 3
2: **if** F не содержит клозов, кроме $\mathcal{T}$ **then**
3: **return** количество таких клозов
4: **end if**
5: **if** формуле F соответствует несвязный граф, с компонентами связности F_1 и F_2 **then**
6: **return** Max2Alg(F_1) + Max2Alg(F_2)
7: **end if**
8: **if** максимальная степень переменных в F равна 3 **then**
9: **return** результат работы известного алгоритма на задаче $(n, 3)$-MAX-2-SAT или $(n, 3)$-MAX-2-CSP, соответственно
10: **end if**
11: выбрать вершину x максимальной степени Δ **такую, что хотя бы один из ее соседей имеет степень меньше Δ.** Если таких вершин не существует, то выбрать вершину x степени Δ
12: **return** $\max(\text{Max2Alg}(A, F[x]), \text{Max2Alg}(A, F[\neg x]))$

2. $\delta_i = \min(\alpha_i - \alpha_{i-1}, \ldots, \alpha_4 - \alpha_3, \alpha_3)$,
для любого i*:*

$$\alpha_i + (i-1)\delta_i + \delta_{i-1} \geq 1,$$

то алгоритм Max2Alg *решает задачу MAX-2-SAT (MAX-2-CSP) за время* $O^*(2^{\alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta + \tau})$, *где* $\tau = \sum_i (1 - \tau_i)$,

$$\tau_i = \alpha_i + i\delta_i.$$

Доказательство. Докажем это утверждение индукцией по числу вершин графа. Снова введем меру сложности функции $\mu = \alpha_3 n_3 + \ldots + \alpha_\Delta n_\Delta$. Обозначим через $T(n_3, \ldots, n_\Delta)$ время работы алгоритма на формуле, в которой n_3 вершин степени 3, n_4 вершин степени 4, …, n_Δ вершин степени Δ.
Если в одной компоненте связности есть как вершины степени Δ, так и вершины степени меньше Δ, то при рекурсивном вызове мера сложности функции уменьшится хотя бы на 1. Действительно, расщепляя вершину степени Δ, мы уменьшаем меру сложности на α_Δ. Так как у вершины есть хотя бы одна соседняя вершина степени меньше Δ, то мера уменьшилась хотя бы на $\delta_{\Delta-1}$. Ещё $\Delta - 1$ соседних вершин, каждая из которых уменьшила меру сложности хотя бы на δ_Δ. Из условия мы знаем, что $\alpha_\Delta + (i-1)\delta_\Delta + \delta_{\Delta-1} \geq 1$.

Следовательно,

$$T(n_3,\dots,n_\Delta) \le 2\cdot 2^{\alpha_3 n_3+\dots+\alpha_\Delta n_\Delta+\tau-1} + poly(|F|) \le 2^{\alpha_3 n_3+\dots+\alpha_\Delta n_\Delta+\tau} + poly(|F|),$$

Теперь рассмотрим случай, когда не существует компоненты связности, в которой бы находились вершины степени Δ и вершины степени меньше Δ. Тогда алгоритм выбирает вершину степени Δ, все соседи которой имеют степень Δ. Это значит, что в рассматриваемой компоненте связности больше никогда не случится ситуации, когда в компоненте связности будут лишь вершины степени Δ. Значит, до тех пор пока максимальная степень в компоненте связности не станет меньше Δ, мера сложности будет убывать на каждой итерации хотя бы на 1. Рассматриваемая итерация алгоритма производит 2 рекурсивных вызова, формульная сложность в каждом из которых меньше хотя бы на τ_i:

$$\begin{aligned} T(n_3,\dots,n_\Delta) &\le 2\cdot 2^k \cdot 2^{\alpha_3 n_3+\dots+\alpha_\Delta n_\Delta-k-\tau_i+\sum_{i=3}^{\Delta-1}(1-\tau_i)} + poly(|F|) \\ &\le 2^{\alpha_3 n_3+\dots+\alpha_\Delta n_\Delta+\sum_{i=3}^{\Delta}(1-\tau_i)} + poly(|F|). \end{aligned}$$

□

Следствие 1.4.4. *Если для любого i выполняются неравенства:*

$$2\alpha_i \ge \alpha_{i+1} + \alpha_{i-1},$$

$$\alpha_i - \alpha_{i-1} \le \alpha_3,$$

то время работы алгоритма Max2Alg

$$O^*(2^{\alpha_D+\varepsilon(\alpha_{D+1}-\alpha_D)+\Delta(1-\alpha_\Delta)}),$$

где $D = \lfloor d \rfloor$, $d = D+\varepsilon$, $d = \frac{2m}{n}$ – средняя степень вершин графа.

Доказательство. Условие можно переписать в виде $\alpha_{i+1} - \alpha_i \le \alpha_i - \alpha_{i-1}$ Значит, разность $\alpha_i - \alpha_{i-1}$ убывает с ростом i, следовательно $\delta_i = \alpha_i - \alpha_{i-1}$. Тогда

$$\tau = \sum_i (1-\tau_i) = \sum (1-\alpha_i - i\delta_i) = \sum (1-\alpha_i - i\delta_i) =$$

$$\sum (1-\alpha_i - i\alpha_i + i\alpha_{i-1}) = \Delta - 3 - (\Delta-1)\alpha_\Delta + 4\alpha_3 = \Delta(1-\alpha_\Delta) + const$$

Для окончания доказательства нам достаточно воспользоваться Леммой 1.4.3.

□

Таблица 1.3 – Значения α_k для $3 \leq k \leq 10$.

d	(n, Δ)-MAX-2-SAT	(n, Δ)-MAX-2-CSP
3	1/6	1/4
4	1/3	3/8
5	13/30	19/40
6	23/45	131/240
7	359/630	1009/1680
8	1553/2520	8651/13440
9	14827/22680	82069/120960
10	155273/226800	855371/1209600

Заметим, что для константного Δ, выражение $\Delta(1 - \alpha_\Delta)$ является константой и может быть вынесено из показателя степени.

Легко показать, что α_i, выбранные при помощи (1.5), удолетворяют условию (1.4.4). Для задачи MAX-2-CSP мы получим следующую последовательность: $\alpha_3 = 1/4$, $\alpha_4 = 3/8$, $\alpha_5 = 19/40$, $\alpha_6 = 131/240$. На любом шаге мы можем продолжить вычислять α_i по более слабой формуле (1.4). Это даст нам явный вид α_i для любого i, если α_k мы уже посчитали по более сильной формуле (1.5):

$$\alpha_i = \frac{i - k + (k+1)\alpha_k}{i+1} =$$
$$1 - \frac{k + 1 - (k+1)\alpha_k}{i+1} = 1 - \frac{k+1}{i+1}(1 - \alpha_k). \tag{1.6}$$

Начальные значения α_k для задач $(n, 3)$-MAX-2-SAT и $(n, 3)$-MAX-2-CSP приведены в Таблице 1.3.
По приведённым в таблице значениям α_i можно получить сложность алгоритма Max2Alg при средней степени вершин i. Сложность алгоритма равна $O^*(2^{n\alpha_i})$ на формулах со средней степенью i.

Так, например, взяв $k = 5$. Из (1.6) получим, для задачи MAX-2-SAT $\alpha_i = 1 - \frac{3.4}{d+1}$, для задачи MAX-2-CSP $\alpha_i = 1 - \frac{3.15}{d+1}$ при $d \geq 5$.
Взяв $k = 8$, получим: $\alpha_i = 1 - \frac{3.45}{d+1}$ для задачи MAX-2-SAT, $\alpha_i = 1 - \frac{3.20}{d+1}$ для задачи MAX-2-CSP при $d \geq 8$.

Это значит, что алгоритм Max2Alg решает задачу MAX-2-SAT (MAX-2-CSP) за время $O^*(2^{n(1-\frac{3.45}{d+1})})$ ($O^*(2^{n(1-\frac{3.20}{d+1})})$) при $d \geq 8$.

Перейдем от сложности относительно d к сложности относительно m. Для этого заменим n на $\frac{2m}{d}$, получим время выполнения $O^*(2^{\frac{2m\alpha}{d}})$. Для задачи MAX-2-CSP минимум получившейся последовательности в точке $d = 5$. Это является новым доказательством лучшей известной оценки $O^*(2^{\frac{m}{5.263}})$ ([37]) для задачи MAX-2-CSP относительно m — количества 2-клозов исходной формулы.

1.5 Задача выполнимости в разреженных графах

В этом разделе мы предлагаем алгоритм решения задачи MAX-2-CSP, экспонента времени работы которого (как функция от средней степени переменной) растет асимптотически медленнее, чем у алгоритма из предыдущего раздела.

Основная идея алгоритма заключается в том, что в графе средней степени $d \leq n/2$ можно найти сбалансированный разделитель T размера nc_d, где $c_d < 1$. Алгоритм для MAX-2-CSP будет расщепляться по всем вершинам из T, а в каждом листе дерева расщеплений будет решать задачу для всех компонент связности.

Мы покажем существование сбалансированных разделителей и эффективный алгоритм их поиска, основанный на результате Feige и Kogan [30]. Следующая лемма — модификация леммы 3.4 из [30] — показывает существование сбалансированных разделителей и зависимость их размера от средней степени графа d.

Лемма 1.5.1. *Пусть $G = (V, E)$ — простой связный граф на n вершинах и m ребрах, пусть средняя степень графа G равна $d = o(n)$. Тогда для любой константы $0 < \alpha < 1$ и достаточно больших значений d, в графе G существует разделитель T размера $n(1 - \frac{2\alpha \ln d}{d})$.*

Доказательство. Пусть $k = \frac{\alpha n \ln d}{d}$, где $\alpha < 1$ — некоторая константа. Мы выберем случайное множество R, состоящее из k вершин G. Покажем, что вероятность существования множества $D \subset V, D \cap R = \emptyset$ мощности не меньше k, что между R и D нет ребер: $(u, v) \in E : u \in R, v \in D$, строго больше 0. Тогда из вероятностного метода следует утверждение теоремы.

Для любой вершины $v \in V \backslash R$ веротяность того, что у v нет соседей в R равна $\frac{\binom{n-d_v}{k}}{\binom{n}{k}}$. Пусть X — случайная переменная, равная количеству вершин из $V \backslash R$, не имеющих соседей в R. Для каждой вершины $v \in V \backslash R$ введем переменную-индикатор I_v такую, что $I_v = 1$ тогда и только тогда, когда у v нет соседей в R. Получаем, что $X = \sum_{v \in V \backslash R} I_v$, а математическое ожидание

X равно

$$E[X] = \sum_{v \in V \setminus R} \frac{\binom{n-d_v}{k}}{\binom{n}{k}} \geq (n-k)\frac{\binom{n-d}{k}}{\binom{n}{k}} \geq \gamma n \prod_{i=0}^{k-1} \frac{n-d-i}{n-i}$$

для достаточно больших d и некоторого $\gamma < 1$. Первое неравенство следует из выпуклости функции суммы биномиальных коэффициентов. Значит, нам необходимо показать, что $\prod_{i=0}^{k-1} \frac{n-i}{n-d-i} \leq \frac{\gamma n}{k}$. Легко видеть, что

$$\prod_{i=0}^{k-1} \frac{n-i}{n-d-i} = \prod_{i=0}^{k-1}(1 + \frac{d}{n-d-i}) \leq (1 + \frac{d}{n-d-k})^k.$$

Т.к. $d = o(n)$, то мы можем выбрать достаточно большое значение d, что последнее выражение будет ограничено $(1+\frac{cd}{n})^k = O(\exp(\frac{ckd}{n}))$ для некоторого $c > 1$, что $\alpha c < 1$. При $k = \frac{n\alpha \ln d}{d}$ мы получим, что $\prod_{i=0}^{k-1} \frac{n-i}{n-d-i} = O(d^{\alpha c})$. С другой стороны, $\frac{\gamma n}{k} = \frac{\gamma d}{\alpha \ln d} = \Omega(\frac{d}{\ln d})$. Следовательно, для достаточно высоких степеней d, существуют графы-разделители размера $n(1 - \frac{2\alpha \ln d}{d})$. □

Теперь мы предложим алгоритм, время работы которого асимптотически лучше, чем время работы алгорита из предыдущего раздела на графах достаточно высоких средних степеней.

Алгоритм 6 Max-2-SAT-GraphSeparator — решение задачи MAX-2-SAT (MAX-2-CSP).

Вход: (G, S) – экземпляр задачи MAX-2-CSP.
Выход: $Opt(G, S)$.

1: Найти сбалансированный разделитель T графа G максимальной мощности
2: Рекурсивно расщепиться по каждой вершине $v \in T$
3: В каждом листе дерева расщеплений решить задачу MAX-2-CSP для всех компонент связности графа $V \setminus T$ полным перебором

Теорема 1.5.1. *Алгоритм* Max-2-CSP-GraphSeparator *находит оптимальное означивание переменных на* $2 - CSP$ *формулах средней степени* $d = o(n)$ *с* n *переменными за полиномиальную память и время* $O^*(2^{\frac{(1+c_d)n}{2}})$, *где* $c_d = 1 - \frac{2\alpha \ln d}{d}$, $0 < \alpha < 1$.

Доказательство. В графе G (k_1, k_2)-взаимно независимое множество может быть найдено за время $O(1.561^n)$ [58]. Очевидно, оставшиеся вершины представляют собой сбалансированный разделитель. Тогда, по Лемме 1.5.1, в графе существует сбалансированный разделитель размера $n(1 - \frac{2\alpha \ln d}{d})$. После

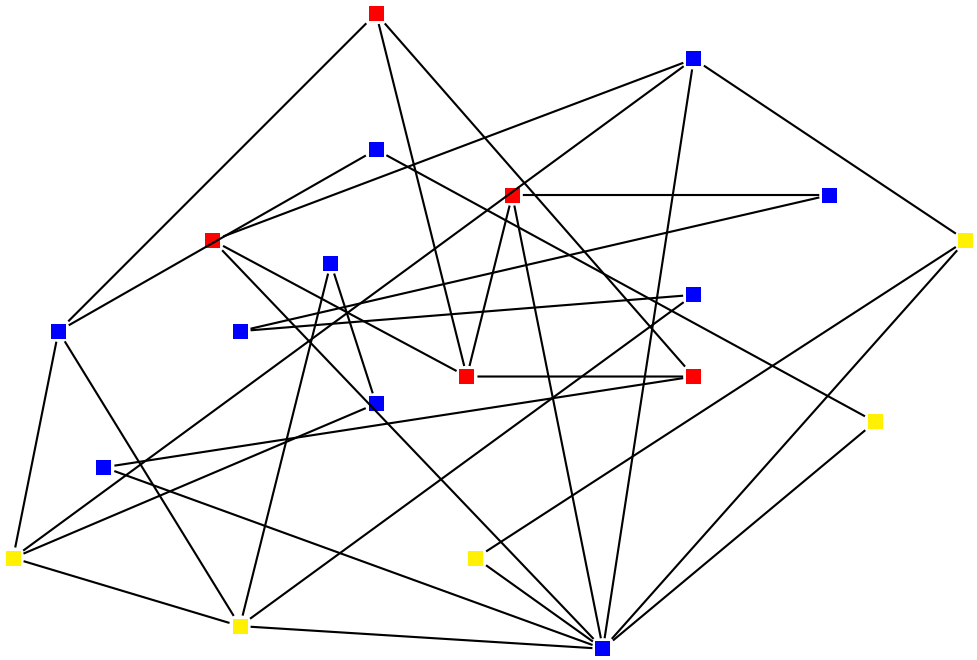

Рисунок 1.3 – Пример разделителя графа. Красные и желтые вершины образуют (5,5)-попарно независимое множество. Голубые вершины являются сбалансированным разделителем размера 10

расщепления по всем вершинам $v \in T$, в каждом листе дерева расщеплений мы независимо решаем задачи на компонентах связности, и объединяем решения, чтобы получить оптимальное означивание переменных. Предполагая, что мы применяем алгоритм полного перебора со временем работы $O^*(2^{\frac{n-nc_d}{2}})$, мы получаем общее время работы алгоритма $O^*(2^{nc_d} \cdot 2^{\frac{n-nc_d}{2}+1}) = O^*(2^{\frac{(1+c_d)n}{2}})$. □

Замечание 1.5.1. Графы-разделители широко применяются при построении алгоритмов на планарных графах, так как по планарной теореме о разделителях [61], планарные графы содержат разделители мощности $O(\sqrt{n})$. Оценки из Леммы 1.5.1 не приводят к асимптотически более быстрым полиномиальным алгоритмам. С другой стороны, существование разделителей линейного размера может привести к лучшим верхним оценкам на время работы экспоненциальных алгоритмов. Насколько нам известно, верхние оценки на размеры сбалансированных разделителей как в Лемме 1.5.1 не исследовались ранее. Интересно изучить вопрос существования лучших верхних оценок относительно средних степеней графа. Также, интересно понять, экспоненциальные алгоритмы для каких задач могут быть построены, используя похожие техники.

ГЛАВА 2
ЗАДАЧА КОММИВОЯЖЁРА

2.1 Постановка задачи

Гамильтонов цикл графа — это цикл, проходящий по каждой вершине графа ровно один раз. Задача коммивояжёра (ЗК) заключается в поиске гамильтонова цикла минимального веса в полном графе с целыми неотрицательными весами рёбер. Метрическая задача коммивояжёра — задача коммивояжёра на графах, удовлетворяющих неравенству треугольника. Ориентированная (асимметрическая) задача коммивояжёра — это задача коммивояжёра на ориентированных графах.

Обозначим через n количество вершин исходного графа, а через M — максимальный вес ребра. $O^*(\cdot)$ скрывает полиномиальные множители от длины входа, т.е. $\mathrm{poly}(n, \log M)$. Вес оптимального гамильтонова цикла в графе G обозначим через $\mathrm{OPT}(G)$. Мы будем опускать G в данной нотации, если из контекста будет понятно, с каким графом мы работаем. Наша цель — по заданному ε найти гамильтонов цикл длины не больше, чем $(1+\varepsilon)\mathrm{OPT}(G)$.

2.2 Известные результаты

2.2.1 Точные алгоритмы

Bellman [12], Held и Karp [44] разработали алгоритм динамического программирования для ЗК. Этот алгоритм использует память и время $O^*(2^n)$ и до сих пор является самым быстрым алгоритмом решения общей задачи о коммивояжёре. Лучший известный алгоритм с полиномиальной памятью имеет время работы $O^*(4^n n^{\log n})$ (Gurevich и Shelah [41], Björklund и Husfeldt [16]).

Koivisto и Parviainen [54] показали, что ЗК может быть решена за время $O^*(2^{2n-t}n^{\log(n-t)})$ и память $O^*(2^t)$ для любого $t = n, n/2, n/4....$ Также авторы предложили алгоритм, который для любого $\sqrt{2} < S < 2$ решает ЗК за $O^*(T^n)$ шагов и память $O^*(S^n)$, так что $TS < 4$.

Kohn, Gottlieb и Kohn [52], Karp [50], Bax и Franklin [9] предложили алгоритм для ЗК со временем работы $O^*(2^n \cdot M)$ и памятью $O^*(M)$, где M — максимальный вес ребра. Björklund [15] разработал алгоритм Монте-Карло со временем работы $O^*(1.657^n \cdot M)$ и экспоненциально маленькой вероятностью

ошибки.

Даже для частного случая ЗК — метрической задачи коммивояжёра — не известны лучшие точные алгоритмы решения. Открытым вопросом является вопрос о существовании алгоритма со временем работы $O^*(2^n) = 2^n \cdot \mathrm{poly}(n, \log M)$ и полиномиальной памятью (см Open Problem 2.2.b, Woeginger [86]).

Eppstein [28] предложил алгоритм для ЗК в кубических графах со временем работы 1.260^n, для графов степени 4 — со временем работы 1.890^n. Iwama и Nakashima [49] улучшили оценку для кубических графов до 1.251^n. Gebauer [38] разработал алгоритм для графов степени 4 со временем работы 1.733^n и экспоненциальной памятью.

Björklund, Husfeldt, Kaski и Koivisto [79] разработали алгоритм со временем работы $O(2-\varepsilon)^n$ для ЗК в графах ограниченной степени, где ε зависит только от степени графа.

Для неориентированной версии задачи о гамильтоновом цикле (ГЦ) оценка $O^*(2^n)$ была улучшена. Björklund [15] предложил алгоритм, решающий ГЦ за время $O^*(1.657^n)$, используя лишь полиномиальную память. Более того, предложенный алгоритм решает задачу ГЦ на двудольных графах за время $O^*(2^{n/2})$.

2.2.2 Приближённые алгоритмы

Общая задача коммивояжёра Известно, что в предположении P$\neq$ NP, общая ЗК не может быть приближена с каким-либо полиномиально вычислимым фактором (Sahni и Gonzalez [75]).

Задача называется сильно NP-трудной, если она остаётся NP-трудной, даже когда все её числовые параметры ограничены многочленом от длины входа (Garey и Johnson [35]). Если P$\neq$NP, сильно NP-трудные задачи не имеют FPTAS (Vazirani [83]). ЗК и метрическая ЗК являются сильно NP-трудными задачами (см., например, Garey и Johnson [36]).

Неориентированная метрическая задача коммивояжёра Существует 2-приближающий алгоритм для метрической ЗК (Rosenkrantz, Stearns и Lewis [74]). Лучшее известное приближение 3/2 получено Christofides [22].

Самая сильная известная нижняя оценка принадлежит Papadimitriou и Vempala [71]: если P$\neq$NP, неориентированная метрическая ЗК не может быть приближена за полиномиальное время с точностью лучше, чем $\frac{117}{116}$.

Ориентированная метрическая задача коммивояжёра Frieze, Galbiati, Maffioli [32] показали, что ориентированная версия метрической ЗК может быть приближена с фактором $\log_2 n$, Bläser [17] улучшил оценку до $0.999 \log_2 n$. Kaplan, Lewenstein, Shafrir и Sviridenko [5] разработали $4/3 \log_3 n$-приближение. Feige и Singh [31] предложили $2/3 \log_2 n$-приближение. Недавно Asadpour, Goemans, Madry, Gharan и Saberi [4] улучшили фактор приближения до $O(\log n / \log \log n)$.

Papadimitriou и Vempala [71] доказали, что если P≠NP, то ориентированная метрическая ЗК не может быть приближена за полиномиальное время с точностью лучше, чем $\frac{220}{219}$

Другие частные случаи Полиномиальная приближённая схема (PTAS) — это алгоритм, который для любого фиксированного ε, за полиномиальное время находит решение, не превосходящее $(1+\varepsilon)$OPT, где OPT — оптимальное решение задачи. Для евклидовой версии ЗК найдена полиномиальная схема приближений (Arora [6] [7], Mitchell [64]).

Gharan, Saberi и Singh [39] разработали приближённый алгоритм с фактором $3/2 - \varepsilon$ для графической версии ЗК. Mömke, Svensson [65] привели 1.461-приближение для графической ЗК. Mucha [67] улучшил фактор приближения до 1.444.

(1,2)-ЗК — неориентированная задача коммивояжёра на графах, в которых веса всех рёбер равны 1 или 2. Даже этот случай задачи является MAX-SNP-трудным (Papadimitriou, Yannakakis [72]). Berman и Karpinski [13] разработали полиномиальный алгоритм, приближающий эту задачу с фактором 8/7.

Приближение ЗК за экспоненциальное время Boria, Bourgeois, Escoffier и Paschos [29] предложили следующие экспоненциальные $(1+\varepsilon)$-приближения метрической версии ЗК:

- за время $O^*(4^{(1-\varepsilon/2)n} n^{\log n})$ и полиномиальную память для любого $\varepsilon \leq 2/3$,
- за время $O^*(2^{(1-\varepsilon/2)n})$ и экспоненциальную память для любого $\varepsilon \leq 2/5$.

2.2.3 Новые результаты

Лучший известный алгоритм решения задачи о коммивояжёре (ЗК) использует память и время $O^*(2^n)$, а лучший алгоритм с полиномиальной памятью имеет время выполнения $O^*(4^n n^{\log n})$. Открытым вопросом является

вопрос о существовании алгоритма со временем работы $O^*(2^n)$ и полиномиальной памятью. Известно, что в предположении $P \neq NP$, общая ЗК не может быть приближена с любым полиномиально вычислимым фактором.

В этой главе мы предлагаем алгоритм, который для любого $0 < \varepsilon < 1$, находит $(1+\varepsilon)$-приближение ориентированной ЗК за время $O^*(2^n\varepsilon^{-1})$, используя память $O^*(\varepsilon^{-1}) = \varepsilon^{-1} \cdot \mathrm{poly}(n, \log M)$. То есть, для любого фиксированного $\varepsilon > 0$, алгоритм использует время $O^*(2^n)$ и лишь полиномиальную память для нахождения $(1+\varepsilon)$-приближения.

Как было указано выше, лучшее известное полиномиальное приближение даже для неориентированной метрической версии ЗК — 1.5 (для ориентированной — $O(\frac{\log n}{\log\log n})$). Если P≠NP, то за полиномиальное время не может быть найдено $\frac{117}{116}$-приближение неориентированной метрической ЗК (см. раздел 2.2.2). Если нам необходимо, например, $\frac{1001}{1000}$-приближение, то мы должны использовать точные алгоритмы, которые требуют либо экспоненциальную память, либо время $4^n n^{\log n}$. Экспоненциальные алгоритмы редко используются на практике, но в любом случае, мы можем их запустить и дождаться ответа. Однако, если алгоритм использует экспоненциальную память, то мы не имеем даже возможности запустить алгоритм на входах разумного размера. Предложенный алгоритм может найти 1001/1000-приближение общей ЗК за время $O^*(2^n)$, используя лишь полиномиальную память.

2.3 Экспоненциальная схема приближения задачи коммивояжёра

2.3.1 Описание алгоритма

Разработанный алгоритм использует идею, предложенную для построения полностью полиномиальной приближённой схемы для задачи о рюкзаке в работе Ibarra, Kim [47]. Авторы используют псевдополиномиальный алгоритм для задачи о рюкзаке, работающий за время $O(nW)$, где n — количество предметов, W — вместимость рюкзака. Полностью полиномиальная схема приближения задачи о рюкзаке сначала делит веса всех предметов на $\alpha(\varepsilon, n, W)$, затем вызывает псевдополиномиальный алгоритм. Полученный ответ может не оказаться оптимальным, т.к. некоторые веса не просто поделены на α, но и округлены. Однако простой анализ показывает, что округление не сильно сказывается на результате.

Мы используем похожую идею. Через OPT обозначим оптимальное решение ЗК. Для того, чтобы получить полиномиальный по памяти приближённый алгоритм, мы сначала разделим веса всех рёбер на достаточно боль-

шое число α, затем запустим точный алгоритм, основанный на формуле включений-исключений. Время работы полученного алгоритма будет равно $2^n \cdot \mathrm{OPT}/\alpha$ и длина полученного цикла не будет превосходить $\mathrm{OPT} + \alpha n$. Теперь мы хотим выбрать α так, чтобы

$$\frac{\mathrm{OPT}}{\mathrm{poly(n)}} \leq \alpha \leq \frac{\varepsilon \mathrm{OPT}}{n}.$$

Метрическая версия ЗК может быть приближена за полиномиальное время, то есть, мы можем найти β, что $\mathrm{OPT} \leq \beta \leq \mathrm{OPT} \cdot \log n$ и взять $\alpha = \frac{\beta \cdot \varepsilon}{n \log n}$. Тогда полученный алгоритм будет иметь время работы $O^*(2^n \varepsilon^{-1})$ и использовать память $O^*(\varepsilon^{-1})$. Позже мы покажем, что такое значение α может быть найдено и для ориентированной неметрической версии ЗК, что позволит обобщить предложенный алгоритм на случай общей задачи коммивояжёра.

Сравнивая предложенный алгоритм с алгоритмом из работы Boria et al. [29], отметим, что для любого фиксированного $\varepsilon > 0$, алгоритм Boria либо использует экспоненциальную память, либо имеет время выполнения $O^*(c^n)$, где $c > 2$.

2.3.2 Алгоритм включений-исключений

Алгоритм включений-исключений основан на следующей хорошо известной формуле (доказательство может быть найдено, например, в Bax [10]).

Теорема 2.3.1. *Пусть X — конечное множество, f, g — функции, определённые на всех подмножествах X. Если для любого $A \subseteq X$ выполняется равенство*

$$g(A) = \sum_{B \subseteq A} f(B)\,,$$

то для любого $A \subseteq X$:

$$f(A) = \sum_{B \subseteq A} (-1)^{|A|-|B|} g(B). \tag{2.1}$$

Например, для проверки существования в графе $G(V, E)$ гамильтонова пути из вершины s в вершину t за время $O^*(2^n)$ и полиномиальную память, можно определить для $A \subseteq V$, $f(A)$ как количество путей (не обязательно простых) длины $n-1$, проходящих по всем вершинам из A. Тогда $f(V)$ равно количеству гамильтоновых путей из s в t. Легко видеть, что $g(A)$ — количество путей (не обязательно простых) длины $n-1$, проходящих только по вершинам из A. Осталось заметить, что $g(A)$ может быть посчитано за полиномиальное время (динамическим программированием или возведением матрицы смежности, индуцированной A, в степень $n-1$).

Обычно работы Kohn, Gottlieb and Kohn [52], Karp [50], Bax and Franklin [9] цитируются как алгоритмы со временем работы $O^*(2^n \cdot M)$, использующие память $O(n \cdot M + n^2)$, где M — максимальнй вес ребра. Для начала нам необходимо показать, что время работы алгоритма включений-исключений может быть оценено как $O^*(2^n \cdot \mathrm{OPT})$, а память — как $O(n \cdot \mathrm{OPT} + n^2)$.

Задача распознавания для ЗК может быть сформулирована следующим образом: по параметру k необходимо определить, существует ли в данном графе G гамильтонов цикл длины не более k. Мы покажем, что задача распознавания может быть решена за время $O^*(2^n k) = 2^n k \cdot \mathrm{poly}(n)$ и память $O^*(k) = k \cdot \mathrm{poly}(n)$.

Через $c(U)$ обозначим количество циклов в $U \subseteq V$ длины не более k, содержащих ровно n рёбер. Отметим, что $c(U)$ может быть вычислено динамическим программированием за время $O(kn^3)$ и память $O(kn)$ (достаточно заполнить таблицу D размера $n \times k \times n$, где $D[v, l, t]$ — количество путей длины l, содержащих t ребер, заканчивающихся в вершине v; оценки приведены в арифметических операциях, но все числа содержат не более $O(n)$ битов, поэтому битовая сложность отличается не более чем на линейный фактор).

Алгоритм 7 InclExcl-Decision-TSP — решение задачи распознавания для ориентированной ЗК за время $O^*(2^n k)$ и память $O^*(k)$.

Вход: $G = (V, E)$ — полный взвешенный ориентированный граф, k — параметр задачи распознавания.

Выход: количество гамильтоновых путей длины $\leq k$.

```
1: res = 0
2: for U ⊆ V do
3:     res = res + (−1)^{|V|−|U|} c(U)
4: end for
5: return res
```

Корректность алгоритма InclExcl-Decision-TSP следует из формулы (2.1). Время работы алгоритма можно оценить как $O^*(2^n k)$, а память — как $O^*(k)$.

Бинарный поиск по параметру k даёт следующий алгоритм (InclExcl-TSP) решения оптимизационной версии ориентированной ЗК.

Лемма 2.3.1. *Алгоритм* InclExcl-TSP *решает ориентированную ЗК за время* $O^*(2^n \cdot OPT)$ *и память* $O^*(OPT)$.

Доказательство. Мы используем двоичный поиск для поиска такого минимального k, что алгоритм InclExcl-Decision-TSP возвращает положительное количество циклов. Нам необходимо $O(\log \mathrm{OPT})$ вызовов InclExcl-

Алгоритм 8 InclExcl-TSP — решение ориентированной ЗК за время $O^*(2^n \cdot \mathrm{OPT})$ и память $O^*(\mathrm{OPT})$.

Вход: $G = (V, E)$ — полный взвешенный ориентированный граф.

Выход: длина кратчайшего гамильтонова цикла.

1: установить $k = 1$ и удваивать k, пока InclExcl-Decision-TSP(G, k)$= 0$

2: использовать двоичный поиск на полученном интервале для нахождения минимального такого k, что InclExcl-Decision-TSP(G, k)> 0

Decision-TSP с параметром $k \leq 2\mathrm{OPT}$, а $O(\log OPT)$ полиномиально от длины входа. Следовательно, время работы составляет $O^*(2^n \cdot \mathrm{OPT})$, а память — $O(n \cdot \mathrm{OPT} + n^2) = O^*(\mathrm{OPT})$.

□

Замечание 2.3.1. Алгоритм InclExcl-TSP был предложен в работе Karp [50]. Мы лишь отметили, что время работы и использованная память могут быть оценены как $O^*(2^n \cdot \mathrm{OPT})$ и $O^*(\mathrm{OPT})$, соответственно.

Замечание 2.3.2. Алгоритм InclExcl-TSP может быть модифицирован для нахождения самого цикла, а не просто установления факта его существования.

2.3.3 Алгоритм решения метрической ЗК

Обозначим через Approx-Asym-TSP алгоритм, дающий $(\log n)$-приближение ориентированной метрической задачи коммивояжёра (алгоритмы перечислены в разделе 2.2.2). Мы предлагаем следующую приближённую схему для ориентированной метрической ЗК.

Лемма 2.3.2. *Алгоритм* Approx-Metric-TSP *находит гамильтонов цикл длины* $\leq (1 + \varepsilon)OPT$ *в графе* G.

Доказательство. Пусть OPT и OPT$'$ — длины оптимальных гамильтоновых циклов в графе G до деления весов рёбер на α и после, соответственно.
Длина полученного алгоритмом пути RES $\leq \alpha \cdot \mathrm{OPT}'$. Обозначим через $I = (e_1, \ldots, e_n)$ рёбра оптимального цикла в исходном графе G. Тогда OPT$'$ не превосходит $\sum_{i \in I} \lceil w(e_i)/\alpha \rceil$. Следовательно,

$$\mathrm{RES} \leq \alpha \cdot \mathrm{OPT}' \leq \alpha \cdot \sum_{i \in I} \lceil w(e_i)/\alpha \rceil \leq \sum_{i \in I} (w(e_i) + \alpha) = \mathrm{OPT} + \alpha n.$$

Из $\beta \leq \mathrm{OPT} \cdot \log n$ получаем $\alpha n \leq \varepsilon \mathrm{OPT}$ и RES $\leq (1 + \varepsilon) \cdot \mathrm{OPT}$. □

Алгоритм 9 Approx-Metric-TSP — $(1+\varepsilon)$-приближение метрической ЗК за время $O^*(2^n\varepsilon^{-1})$ и память $O^*(\varepsilon^{-1})$.

Параметр: $\varepsilon > 0$ — фактор приближения.
Вход: $G = (V, E)$ — полный взвешенный ориентированный граф.
Выход: гамильтонов цикл веса $\leq (1+\varepsilon)\mathrm{OPT}(G)$.

1: $\beta =$ Approx-Asym-TSP(G)
2: разделить веса всех рёбер на α: $w(e) = \lceil w(e)/\alpha \rceil$, где

$$\alpha = \frac{\beta \cdot \varepsilon}{n \log n}$$

3: **return** InclExcl-TSP(G)

Лемма 2.3.3. *Время работы алгоритма* Approx-Metric-TSP *равно* $O^*(2^n\varepsilon^{-1})$, *память* — $O^*(\varepsilon^{-1})$.

Доказательство. Время работы шага 1 полиномиально (см. раздел 2.2.2). По Лемме 2.3.1, время работы шага 3 составляет $O^*(2^n \cdot \mathrm{OPT}')$, а память — $O^*(\mathrm{OPT}')$. Оценка $\beta \geq \mathrm{OPT}$ даёт

$$\mathrm{OPT}' \leq \frac{\mathrm{OPT}}{\alpha} + n \leq \frac{n \log n}{\varepsilon} + n = O^*(\varepsilon^{-1}).$$

Следовательно, время работы алгоритма Approx-Metric-TSP $O^*(2^n\varepsilon^{-1})$, используемая память — $O^*(\varepsilon^{-1})$. □

2.3.4 Алгоритм решения общей ЗК

В предыдущем разделе мы рассматривали случай метрической версии ЗК, т.к. предложенный алгоритм вызывал полиномиальный алгоритм приближения ЗК, чтобы вычислить значение β. Сейчас мы покажем как избежать вызова приближённого алгоритма и обобщить результат на случай общей задачи коммивояжёра. Единственное различие алгоритмов заключается в шагах 2-5 поиска 4-приближения общей ЗК за экспоненциальное время.

Лемма 2.3.4. *Алгоритм* Approx-TSP *находит гамильтонов цикл длины* $\leq (1+\varepsilon)OPT(G)$ *в графе* G.

Доказательство. Нам достаточно показать, что после шага 5 значение β является 4-приближением OPT, т.е., $\mathrm{OPT} \leq \beta \leq 4\mathrm{OPT}$. После этого шага алгоритм идентичен Approx-Metric-TSP.

Пусть β_{end} обозначает значение β в последней итерации цикла (перед шагом 4). Т.к. G' на последней итерации цикла содержит цикл длины $\leq 2n$, G

Алгоритм 10 Approx-TSP — $(1+\varepsilon)$-приближение общей ЗК за время $O^*(2^n\varepsilon^{-1})$ и память $O^*(\varepsilon^{-1})$.

Параметр: $\varepsilon > 0$ — фактор приближения.
Вход: $G = (V, E)$ — полный взвешенный ориентированный граф.
Выход: гамильтонов цикл веса $\leq (1+\varepsilon)\mathrm{OPT}(G)$.

1: $\beta = 1$
2: **repeat**
3: пусть G' — граф, полученный из G делением весов всех ребер на α': $w'(e) = \lceil w(e)/\alpha' \rceil$, где
$$\alpha' = \frac{\beta}{n}$$
4: $\beta = 2\beta$
5: **until** InclExcl-Decision-TSP$(G', 2n) > 0$
6: разделить все веса на α: $w(e) = \lceil w(e)/\alpha \rceil$, где
$$\alpha = \frac{\beta \cdot \varepsilon}{4n}$$
7: **return** InclExcl-TSP(G)

содержит цикл длины $\leq 2n\alpha_{end} = 2\beta_{end}$, следовательно, OPT $\leq 2\beta_{end}$. Теперь покажем, что OPT $\geq \beta_{end}/2$. Пусть $\beta_{prev} := \beta_{end}/2$, а $\alpha_{prev} = \beta_{prev}/n$. Заметим, что β_{prev} и α_{prev} — значения α и β на предыдущей итерации цикла. Если бы G содержал цикл длины не больше $\beta_{end}/2$, то на предыдущей итерации G' содержал бы цикл длины $\beta_{prev}/\alpha_{prev} + n = 2n$, что привело бы к выходу из цикла.

Следовательно, $2\beta_{end}$ является 4-приближением OPT. □

Лемма 2.3.5. *Время работы алгоритма* Approx-TSP *равно* $O^*(2^n + 2^n\varepsilon^{-1})$, *память* — $O^*(\varepsilon^{-1})$.

Доказательство. Время выполнения шага 5 равно $O^*(2^n \cdot 2n) = O^*(2^n)$, память — $O^*(2n) = O^*(1)$. Алгоритм совершает $O(\log_2(4 \cdot \mathrm{OPT})) = \mathrm{poly}(n, \log M)$ вызовов алгоритма InclExcl-Decision-TSP. По аналогии с Леммой 2.3.3, время выполнения Approx-TSP равно $O^*(2^n + 2^n\varepsilon^{-1})$, а память — $O^*(\varepsilon^{-1})$. □

Замечание 2.3.3. Отметим, что улучшение времени работы предложенного алгоритма до $O^*(2^n \cdot \log \varepsilon^{-1})$ должно быть сложной задачей, т.к. вызвав этот алгоритм с параметром $\varepsilon = (nM+1)^{-1}$, мы решили бы задачу ЗК за время $O^*(2^n)$, используя лишь полиномиальную память.

ГЛАВА 3
ЗАДАЧА О КРАТЧАЙШЕЙ ОБЩЕЙ НАДСТРОКЕ

3.1 Постановка задачи

В задаче о кратчайшей общей надстроке (SCS) необходимо найти кратчайшую строку, содержащую каждую из n заданных строк $\{s_1, \ldots, s_n\}$ в качестве подстроки. Задача о надстроке является NP-трудной задачей и имеет множество практических применений, в том числе сборка генома, хранение и сжатие информации. По этой причине было разработано множество приближенных алгоритмов решения задачи о кратчайшей общей надстроке. Долгое время лучшим приближением задачи SCS было приближение Sweedyk [78] с фактором 2.5 (такое же приближение следует из 2/3-приближения задачи MAX-ATSP [5, 70]). Недавно эта оценка была улучшена Mucha [68] до $2\frac{11}{23}$. Лучшим известным результатом о неприближаемости задачи (в предположении P $\neq$ NP) является оценка $\frac{345}{344}$ Karpinski и Schmied [51]. Лучший известный алгоритм точного решения задачи о надстроке выполняется за время $O^*(2^n)$ ($O^*(\cdot)$ скрывает полиномиальные факторы от длины входа), где n — количество входных строк.

Мы будем обозначать через $\mathcal{S} = \{s_1, \ldots, s_n\}$ множество входных строк над алфавитом Σ, через n — количество строк. Не умаляя общности, мы будем предполагать, что ни одна входная строка не является подстрокой некоторой другой входной строки. Под задачей об r-надстроке мы понимаем задачу о кратчайшей надстроке, где длины всех входных строк не превосходят r. В частности, задача о 3-надстроке — это задача о кратчайшей общей надстроке множества строк длины не более 3.

Мы обозначаем пустую строку через ε. $u \sqsupset v$ ($u \sqsubset v$) обозначает, что строка u является суффиксом (префиксом) строки v. Через $s \circ t$ мы обозначаем конкатенацию строк s и t. $\mathrm{overlap}(s,t)$ — самый длинный суффикс s, являющейся префиксом t. Через $\mathrm{prefix}(s,t)$ мы обозначаем первые $|s| - |\mathrm{overlap}(s,t)|$ символов строки s, а через $\mathrm{suffix}(s,t)$ —последние $|t| - |\mathrm{overlap}(s,t)|$ символов t. Следовательно,

$$\mathrm{prefix}(s,t) \circ \mathrm{overlap}(s,t) = s \text{ и } \mathrm{overlap}(s,t) \circ \mathrm{suffix}(s,t) = t\,.$$

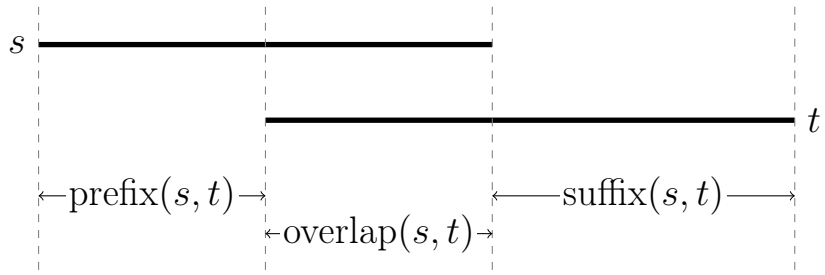

Рисунок 3.1 – Примеры функций prefix, overlap и suffix. Например, overlap(ABACBA, BABCA) = BA**, prefix**(ABACBA, BABCA) = ABAC**, suffix**(ABACBA, BABCA) = BCA

3.2 Известные результаты

Не известно может ли задача о кратчайшей общей надстроке быть решена за время, меньшее $O^*(2^n)$. Заметим, что SCS является перестановочной задачей: чтобы найти строку, содержащую подстроки s_i в заданном порядке, достаточно просто максимально пересечь подстроки в этом порядке. Таким образом, тривиальный алгоритм имеет время выполнения $O^*(n!)$. Рассмотрим граф суффиксов заданных подстрок: множество вершин графа совпадает с множеством строк $\{s_1, \ldots, s_n\}$, вершины s_i и s_j соединены дугой веса $|\text{suffix}(s_i, s_j)|$, где $\text{suffix}(s_i, s_j)$ равен такой кратчайшей строке, что s_j — суффикс $s_i \circ \text{suffix}(s_i, s_j)$ (символ $\circ$ обозначает конкатенацию строк). Теперь задача о надстроке может быть решена алгоритмом поиска кратчайшего пути коммивояжера в графе суффиксов. Для задачи коммивояжера известны алгоритмы со временем работы $O^*(2^n)$ [12, 44].

У задачи о кратчайшей общей надстроке есть два естественных частных случая: случай с ограниченным алфавитом и случай со строками ограниченной длины. Задача о кратчайшей общей надстроке строк длины не более r называется r-надстрокой (r-SCS). Заметим, что если оба параметра ограничены, то задача вырождается, т.к. количество различных входов задачи в таком случае ограничено константой. Известно, что SCS над бинарным алфавитом и 3-SCS являются NP-трудными задачами. В то же время 2-SCS может быть решена за линейное время [34], а 2-SCS с повторениями — за квадратное [1]. Vassilevska [82] показала, что SCS над бинарным алфавитом не может быть значительно проще общего случая. Для этого она предложила полиномиальное сведение общей задачи о надстроке к надстроке над бинарным алфавитом, которое сохраняет количество входных строк. Следовательно, из α-приближения SCS над бинарным алфавитом будет следовать α-приближение общего случая задачи. Это также означает, что из $O^*(c^n)$-алгоритма для надстроки над бинарным алфавитом следует $O^*(c^n)$-алгоритм

для надстроки в общем случае. Значит, SCS над маленьким алфавитом не может быть многим проще общего случая. В этой главе мы покажем, что SCS над короткими строками является более простой задачей.

В этой главе мы покажем, что задача о 3-надстроке может быть решена за время $O^*(3^{n/3})$ и полиномиальную память. Также мы покажем, что для любой константы r, задача о кратчайшей r-надстроке может быть решена вероятностным алгоритмом с односторонней ошибкой за время $O^*(2^{(1-c(r))n})$, где $c(r) = (1 + 2r^2)^{-1}$.

Текущая ситуация с точными алгоритмами для задачи о кратчайшей общей надстроке похожа на ситуацию с другими NP-трудными задачами, такими как задача выполнимости SAT, задача максимальной выполнимости MAX-SAT, задача о коммивояжере, задача о раскраске. Например, несмотря на большее количество усилий, лучшие известные алгоритмы для общих случаев этих задач выполняются за время $O^*(2^n)$ (где n количество переменных/вершин). Но для частных случаев задач известны лучшие верхние оценки:

- SAT: $O(1.308^n)$ для 3-SAT [45]; $O^*((2 - 2/k)^n)$ для k-SAT [76, 66];
- MAX-SAT: $O(1.731^n)$ для MAX-2-SAT [84, 53], $O(1.109^n)$ для $(n, 3)$-MAX-2-SAT [57]; $O((2 - \varepsilon)^n)$ на формулах константной плотности [25, 57];
- ЗК: $O(1.2186^n)$ в кубических графах [26]; $O((2 - \varepsilon)^n)$ в графах ограниченной максимальной/средней степени [79, 24];
- Задача о раскраске: $O(1.3289^n)$ для 3-раскраски [11]; $O((2 - \varepsilon)^n)$ для графов ограниченной максимальной степени [80];

Улучшение оценки $O^*(2^n)$ для каждой из этих задач (а также для задачи о надстроке) остается открытым вопросом.

3.3 Суффиксные графы и задача коммивояжера

Очевидно, что $s \circ \mathrm{suffix}(s, t)$ — кратчайшая строка, содержащая s и t в таком порядке. Более того, кратчайшая строка, содержащая подстроки $s_{i_1}, \ldots, s_{i_n}$ в этом порядке — это

$$s_{i_1} \circ \mathrm{suffix}(s_{i_1}, s_{i_2}) \circ \cdots \circ \mathrm{suffix}(s_{i_{n-1}}, s_{i_n}) .$$

Таким образом, в задача о кратчайшей общей надстроке необходимо найти перестановку n входных строк, минимизирующую суммарную длину суффиксной функции. Как было сказано раньше, суффиксный граф — это полный ориентированный граф на n вершинах $\{s_1, \ldots, s_n\}$, в котором вершины

s_i and s_j соединены дугой веса $|\text{suffix}(s_i, s_j)|$. Решение SCS соответствует решению ЗК в этом графе, связь этих задач используется в большинстве приближенных алгоритмов решения задачи о надстроке. Известны алгоритмы динамического программирования для решения задачи ЗК: Bellman [12], Held и Karp [44], эти алгоритмы используют $O^*(2^n)$ времени и памяти. Также известны алгоритмы, основанные на формуле включений-исключений, использующие время $O^*(2^n \cdot M)$ и память $O^*(M)$ [52, 50, 9] (M — максимальный вес дуги). Lokshtanov и Nederlof [62] предложили алгоритм решения ЗК за время $O^*(2^n \cdot M)$ с полиномиальной $O^*(1) = \text{poly}(n, \log M)$ памятью. Для неориентированной версии ЗК Björklund [15] недавно предложил вероятностный алгоритм со временем работы $O^*(1.657^n \cdot M)$. Отметим, что в задаче о надстроке M не превышают длину входа, поэтому алгоритмы включения-исключения решают задачу о надстроке за время $O^*(2^n)$ time, используя лишь полиномиальную память. Это лучшая известная оценка для задачи о надстроке.

3.4 Графы де Брюйна и задача о сельском почтовом

В этом разделе мы рассмотрим графы де Брюйна. Эти графы имеют широкое применение в сборке генома — одном из самых важных приложений задачи о надстроке [73]. Но графы де Брюйна до сих практически не применялись в теоретических работах о надстроке. Рассмотрим множество входных строк длины 3: $\mathcal{S} = \{s_1, \ldots, s_n\}$ над алфавитом Σ. Граф де Брюйна DG — это полный взвешенный ориентированный граф (с петлями, но без кратных ребер) с множеством вершин Σ^2. Две различные вершины s и t в нем соединяются дугой веса $|\text{suffix}(s, t)|$. Также, для каждой строки из Σ^2, состоящей из двух одинаковых символов, в графе есть петля веса 1. То есть, вес дуги (s, t) равняется количеству символов, которые нам необходимо дописать, проходя от строки s к строке t (в частности, проходя от строки `AA` к ней самой, нам нужно дописать еще один символ `A`, что соответствует петле веса 1). Таким образом, все дуги графа DG имеют вес 1 или 2. Заметим, что любая строка s длины 3 над алфавитом Σ задает дугу веса 1 в графе DG: это дуга, соединяющая префикс s длины 2 с ее суффиксом длины 2. Следовательно, решение SCS соответствует кратчайшему пути в DG, проходящему по всем ребрам $E_{\mathcal{S}}$, заданным строками из $\mathcal{S}$.

Эта задача известна как ориентированная задача о сельском почтовом (directed rural postman path problem). В задаче о сельском почтовом по заданному взвешенному графу $G = (V, E)$ и подмножеству дуг $E_R \subseteq E$, необходимо найти кратчайший путь в G, проходящий по всем дугам из E_R. Дуги из E_R называются обязательными, все остальные дуги графа называются оп-

циональными. Путь, проходящий по всем обязательным ребрам, называется сельским путем.

Задача о сельском почтовом имеет множество практических применений (см., например, [27, 40]). В то же время для задачи не известны нетривиальные точные алгоритмы. Как и для задач SCS и ЗК, для задачи о сельском почтовом существует переборный алгоритм со временем работы $O^*(n!)$ (n обозначает количество обязательных дуг) и алгоритм динамического программирования со временем работы $O^*(2^n)$. Задача о сельском почтовом обобщает задачу коммивояжера и задачу о китайском почтовом. Если множество обязательных ребер образует компоненту слабой связности, то задача разрешима за полиномиальное время: нам необходимо просто добавить дуги минимального суммарного веса к несбалансированным вершинам. Это может быть выполнено алгоритмом поиска минимального по весу совершенного паросочетания в соответствующем двудольном графе (см., например, [3]). Если же множество обязательных ребер образует более, чем одну компоненту слабой связности, то задача становится NP-сложной [59]. Так как теперь нам не достаточно просто добавить минимальные по весу ребра к несбалансированным вершинам, но нам также нужно гарантировать, что полученный граф связен.

Однако, задача о 2-надстроке может быть решена за полиномиальное время, даже если входные строки не образуют одну компоненту слабой связности. Важным свойством 2-надстроки (в отличие от 3-надстроки) является то, что строки из различных слабых компонент не пересекаются. То есть, мы можем искать сельские пути в каждой компоненте отдельно.

Для решения задачи о 3-надстроке мы будем использовать описанную связь между графами де Брюйна и задачей о сельском почтовом. В задаче об r-надстроке мы применим известный алгоритм решения задачи о сельском почтовом к иерархическому графу, структура которого будет описана позже. Здесь же мы приводим известный результат о решении задачи о сельском почтовом.

Теорема 3.4.1. *[Gutin, Wahlström, Yeo [42]] Пусть $G = (V, A)$ — ориентированный взвешенный мультиграф, $R \subseteq A$ — подмножество дуг, $l = poly(|V|)$, тогда существует вероятностный алгоритм с односторонней ошибкой, проверяющий существование сельского пути длины l за время $O^*(2^k)$, где k — количество компонент слабой связности в подграфе G, индуцированном R.*

3.5 Задача о 3-надстроке

В этом разделе мы предлагаем алгоритм, решающий задачу о 3-надстроке за время $O^*(3^{n/3})$ и полиномиальную память. Алгоритм основан на поиске кратчайшего пути сельского почтового в графе де Брюйна исходного множества строк. Алгоритм перебирает все возможные варианты, предварительно сузив область поиска до $3^{n/3}$, а затем проверяет каждый вариант за полиномиальное время. Мы покажем, что для случая 3-надстроки достаточно найти вершины, в которых оптимальный путь входит в каждую компоненту слабой связности. После мы покажем, что в слабой компоненте на k дугах не более k потенциальных точек входа. Т.к. общее число дуг равно n, то время работы примерно равно $k^{n/k}$, что не превосходит $3^{n/3}$ для натуральных k.

3.5.1 Описание алгоритма

Множество $\mathcal{S}$ входных строк длины 3 определяет множество обязательных дуг $E_{\mathcal{S}}$ графа де Брюйна DG. В этом графе мы ищем оптимальный сельский путь, т.е. кратчайший путь, проходящий по всем обязательным дугам. Опциональные дуги имеют вес 1 или 2, в то время как обязательные дуги всегда имеют вес 1. Для каждой вершины графа нам известно количество входящих и выходящих обязательных дуг в оптимальном сельском пути, но нам не известно количество таких опциональных дуг.

Заметим два простых свойства оптимального сельского пути.

- Оптимальный сельский путь не начинается и не заканчивается опциональной дугой (удаление такой дуги привело бы к пути меньшей длины).
- Существует оптимальный сельский путь, который не содержит двух подряд идущих опциональных дуг. Две такие дуги могут быть заменены одной. Т.к. веса всех дуг равны 1 или 2, то это не увеличит общий вес пути.

Через $d^{\text{re}}_{\text{in}}(v)$ и $d^{\text{re}}_{\text{out}}(v)$ мы обозначаем количество обязательных входящих и исходящих дуг вершины v: $d^{\text{re}}_{\text{in}}(v) = |\{(u,v) \in E_{\mathcal{S}}\}|$ и $d^{\text{re}}_{\text{out}}(v) = |\{(v,w) \in E_{\mathcal{S}}\}|$. Аналогично, для пути P графа DG, через $d^{\text{op}}_{\text{in}}(P,v)$ и $d^{\text{op}}_{\text{out}}(P,v)$ мы обозначаем количество опциональных входящих и исходящих дуг вершины v пути P:

$$\begin{aligned} d^{\text{op}}_{\text{in}}(P,v) &= |\{(u,v) \mid (u,v) \in P, (u,v) \notin E_{\mathcal{S}}\}|, \\ d^{\text{op}}_{\text{out}}(P,v) &= |\{(v,w) \mid (v,w) \in P, (v,w) \notin E_{\mathcal{S}}\}|. \end{aligned}$$

Напомним, что путь может проходить по некоторым вершинам несколько раз, поэтому некоторые степени могут быть больше 1. Аналогично, путь P

может проходить по некоторым дугам несколько раз, поэтому множества в определениях $d^{\text{op}}_{\text{in}}(P, v)$ и $d^{\text{op}}_{\text{out}}(P, v)$ являются мультимножествами. Другими словами, $d^{\text{op}}_{\text{in}}(P, v)$ ($d^{\text{op}}_{\text{out}}(P, v)$) — количество входов (выходов) пути P в вершину v по опциональным дугам.

Определение 3.5.1. Конфигурация — это пара функций $f = (f_{\text{in}}, f_{\text{out}})$ из V в $\mathbb{N}$. Конфигурация указывает сколько каждая вершина пути имеет опциональных входящих и исходящих дуг. Мы говорим, что конфигурация совместна с путем P, если для каждой вершины v: $f_{\text{in}}(v) = d^{\text{op}}_{\text{in}}(P, v)$ и $f_{\text{out}}(v) = d^{\text{op}}_{\text{out}}(P, v)$. Путь в DG естественным образом определяет конфигурацию.

Определение 3.5.2. Мы называем конфигурацию $f = (f_{\text{in}}, f_{\text{out}})$ нормальной, если выполняются следующие три условия:

- Конфигурация совместна хотя бы с одним сельским путем. В частности, для любой вершины v, кроме, возможно, двух вершин (этими двумя вершинами являются начальная и конечная вершины пути):

$$f_{\text{in}}(v) + d^{\text{re}}_{\text{in}}(v) = f_{\text{out}}(v) + d^{\text{re}}_{\text{out}}(v). \tag{3.1}$$

- Для каждой компоненты слабой связности $\mathcal{C}$ из $E_{\mathcal{S}}$,

$$\sum_{v \in \mathcal{C}} \min\{f_{\text{in}}(v), f_{\text{out}}(v)\} \leq 1\,. \tag{3.2}$$

  Т.е., каждая компонента слабой связности содержит не более одной вершины, которая имеет и входящие, и выходящие опциональные дуги. Более того, если такая вершина существует, то она имеет либо всего лишь одну исходящую дугу, либо всего лишь одну исходящую дугу. Такая вершина называется особенной вершиной.

- Если вершина v имеет только входящие (исходящие) обязательные дуги, то она имеет только исходящие (входящие) опциональные дуги:

$$(d^{\text{re}}_{\text{out}}(v) = 0 \Rightarrow f_{\text{in}}(v) = 0) \text{ и } (d^{\text{re}}_{\text{in}}(v) = 0 \Rightarrow f_{\text{out}}(v) = 0). \tag{3.3}$$

  В частности, вершина v с $\min\{d^{\text{re}}_{\text{in}}(v), d^{\text{re}}_{\text{out}}(v)\} = 0$ не может быть особенной.

Определение 3.5.3 (нормальный путь)**.** Нормальным путем называется путь с нормальной конфигурацией.

Следующие леммы (которые будут доказаны позже) поясняют мотивацию изучения нормальных конфигураций.

Лемма 3.5.1. *Существует оптимальный сельский путь, являющийся нормальным путем.*

Лемма 3.5.2. *По заданной нормальной конфигурации f некоторого оптимального сельского пути, за полиномиальное время может быть найден оптимальный сельский путь, совместный с конфигурацией f.*

Осталось показать, что число нормальных конфигураций не слишком велико.

Лемма 3.5.3. *Компонента слабой связяности $\mathcal{C}$ из E_S, содержащая k дуг, имеет не более k различных нормальных конфигураций.*

Лемма 3.5.4. *Все нормальные конфигурации могут быть перечислены за время $O^*(3^{n/3})$, используя лишь полиномиальную память.*

Из этих четырех лемм следует главная теореме этого раздела.

Теорема 3.5.1. *Задача о кратчайшей общей 3-надстроке может быть решена за время $O^*(3^{n/3})$ и полиномиальную память.*

Доказательство. По Лемме 3.5.4, все нормальные конфигурации могут быть перечислены за время $O^*(3^{n/3})$ и полиномиальную память. По Лемме 3.5.1, хотя бы одна нормальная конфигурация соответствует оптимальному сельскому пути. По такой конфигурации оптимальный путь может быть найден с помощью Леммы 3.5.2. □

3.5.2 Доказательства лемм

В этом разделе мы приводим анализ алгоритма и доказываем леммы из предыдущего раздела. В доказательствах мы рассматриваем пути в графах как последовательности вершин. Прописные буквы обозначают вершины, а заглавные — части пути, т.е. (возможно пустые) последовательности вершин. Например, чтобы указать, что пусть P начинается в вершине s, проходит через вершину v и заканчивается в вершине t, мы будем писать $P = sAvBt$. На рисунках обязательные ребра обозначены жирными линиями, опциональные ребра — тонкими линиями, волнистые линии обозначают части пути.

Доказательство Леммы 3.5.1. Пусть P — оптимальный сельский путь, содержащий минимальное количество опциональных дуг. Мы покажем, что если P — не нормальный путь, то количество опциональных дуг в P может быть уменьшено без увеличения веса пути. Мы будем делать это заменами пар дуг на одну дугу. Т.к. веса всех дуг равны 1 и 2, то такие замены не увеличат вес пути P.

Рассмотрим компоненту слабой связности $\mathcal{C}$, пусть x — последняя вершина $\mathcal{C}$ в пути P. Чтобы гарантировать, что (3.2) выполняется, мы сначала преобразуем P так, что для всех вершин v из $\mathcal{C}$, кроме, возможно, вершины x, выполняется

$$\min\{d^{\mathrm{op}}_{\mathrm{in}}(P, v), d^{\mathrm{op}}_{\mathrm{out}}(P, v)\} = 0\,.$$

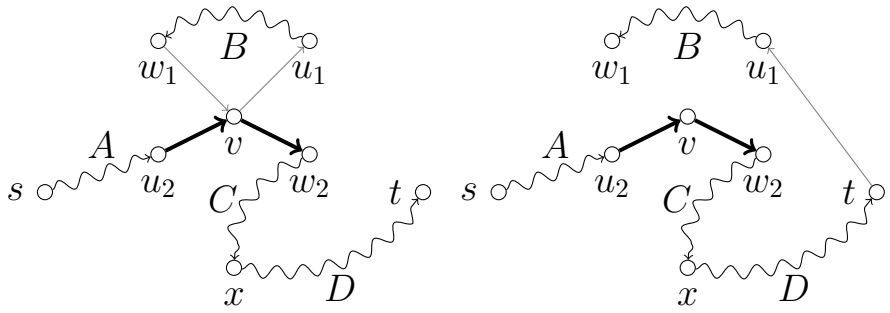

Рисунок 3.2 – Первое преобразование Леммы 3.5.1

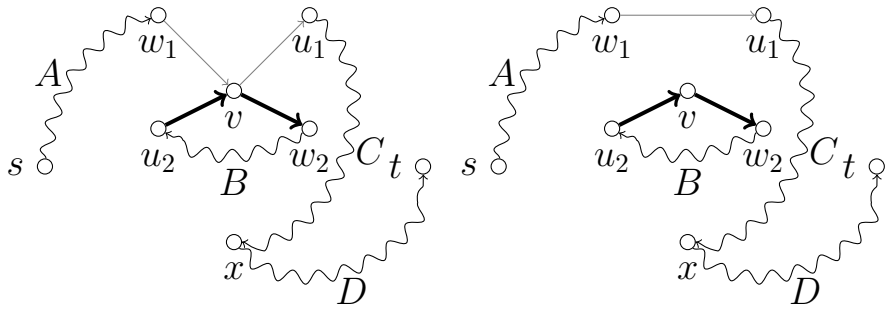

Рисунок 3.3 – Второе преобразование Леммы 3.5.1

Предположим, что в $\mathcal{C}$ существует вершина $v \neq x$, для которой не выполняется это равенство. Обозначим входящие и исходящие опциональные дуги v через (w_1, v) и (v, u_1), соответственно. Пусть u_2, w_2 — это такие вершины, что дуга (u_2, v) предшествует дуге (v, u_1) в пути P, а дуга (v, w_2) следует за дугой (w_1, v) в P. Т.к. путь не содержит двух подряд идущих опциональных дуг, то дуги (u_2, v) и (v, w_2) отличаются от дуг (w_1, v) и (v, u_1). Рассмотрим два случая (см. Рисунки 3.2 и 3.3): путь сначала проходит по дугам (u_2, v), (v, u_1) или по дугам (w_1, v), (v, w_2).

1. Путь P выглядит как $sAu_2vu_1Bw_1vw_2CxDt$ (т.е., P сначала проходит по (u_2, v) и (v, u_1), а потом по (w_1, v) и (v, w_2)). Мы заменяем этот путь на $sAu_2vw_2CxDtu_1Bw_1$. Отметим, что хоть эта замена и увеличивает количество опциональных дуг из вершины t, она уменьшает общее количество опциональных дуг.
2. Путь P имеет вид $sAw_1vw_2Bu_2vu_1CxDt$. В этом случае мы заменяем дуги (w_1, v) и (v, u_1) новой дугой (w_1, u_1). В результате мы получаем путь sAw_1u_1CxDt и цикл vw_2Bu_2v. Напомним, что $\mathcal{C}$ — компонента слабой связности. Значит, новый путь содержит хотя бы одну общую вершину с циклом. Следовательно, мы можем вклеить полученный цикл в путь.

Оба преобразования уменьшают общее количество опциональных дуг и не разбивают путь.

Сейчас мы покажем, что путь P может быть изменен так, что $\min\{d^{\text{op}}_{\text{in}}(P,x), d^{\text{op}}_{\text{out}}(P,x)\} \leq 1$. Предположим, что у x в P есть хотя бы две входящие и хотя бы одна исходящая опциональные дуги. Пусть (v,x) и (x,w) — первая входящая и первая исходящая опциональные дуги x в пути P, соответственно. Рассмотрим два случая, проиллюстрированных на Рисунке 3.4.

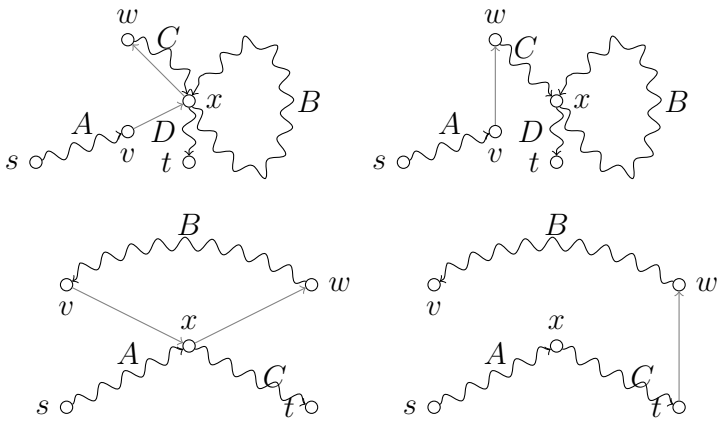

Рисунок 3.4 – Преобразование пути, применяемое в Лемме 3.5.1

1. P сначала проходит по дуге (v,x), а потом по дуге (x,w). Т.к. путь P содержит хотя бы две опциональные дуги из x, то P выглядит как $sAvxBxwCxDt$. Тогда мы можем заменить его на путь $sAvwCxBxDt$.
2. P сначала проходит по дуге (x,w). В этом случае P выглядит как $sAxwBvxCt$ и может быть заменен на $sAxCtwBv$.

Таким образом, P выполняет (3.2).

Теперь покажем, как преобразовать P, чтобы выполнить (3.3). Рассмотрим вершину $v \in \mathcal{C}$, не умаляя общности предположим, что она не имеет обязательных входящих дуг (т.е., $d^{\text{re}}_{\text{in}}(v) = 0$). Предположим, что P содержит опциональную дугу (v,w). Т.к. P не может начинаться с опциональной дуги, то в пути есть дуга (u,v), предшествующая (v,w), и являющаяся опциональной. Но тогда две опциональные дуги (u,v) и (v,w) могут быть заменены одной дугой (u,w). Это снова противоречит предположению, что P содержит минимальное количество опциональных дуг. Случай $d^{\text{re}}_{\text{out}}(v) = 0$ аналогичен рассмотренному. Теперь P выполняет (3.3).

Теперь мы можем заключить, что любой сельский путь с минимальным количеством дуг выполняет свойства (3.2) и (3.3). Свойство (3.1) для такого пути выполняется очевидным образом. Следовательно, любой такой путь является нормальным. □

Доказательство Леммы 3.5.2. Далее мы будем предполагать, что мы знаем первую вершину s и последнюю вершину t оптимального сельского пути, который мы ищем. Т.к. первая и последняя дуги такого пути являются обязательными дугами, то мы можем перечислить все пары (s, t) за время $O(n^2)$.

Для поиска пути мы изменим граф DG и множество обязательных дуг $E_{\mathcal{S}}$ следующим образом:

- Введем $|\Sigma|$ новых вершин, пронумерованных символами алфавита, соединим их со всеми остальными вершинами графа дугами равными длинам суффиксов двух соответствующих строк. Например, $w(\mathtt{A}, \mathtt{AB}) = 1$, $w(\mathtt{A}, \mathtt{BC}) = 2$, $w(\mathtt{BC}, \mathtt{A}) = 1$, $w(\mathtt{BA}, \mathtt{A}) = 0$, $w(\mathtt{A}, \mathtt{B}) = 1$.
- Для каждой вершины v исходного графа DG, пронумерованного строкой $\mathtt{AB}$, добавим $f_{\text{in}}(v)$ копий дуги $(\mathtt{A}, \mathtt{AB})$ и $f_{\text{out}}(v)$ копий дуги $(\mathtt{AB}, \mathtt{B})$ в множество обязательных дуг $E_{\mathcal{S}}$.

Обозначим полученный граф DG', а полученное множество обязательных дуг — $E'_{\mathcal{S}}$. Отметим, что $E'_{\mathcal{S}}$ — мультимножество, которое может содержать несколько копий обязательных дуг (непример, $f_{\text{in}}(\mathtt{AB})$ копий дуги $(\mathtt{A}, \mathtt{AB})$).

Пусть $C_1, \ldots, C_p$ — компоненты слабой связности $E_{\mathcal{S}}$, а $C'_1, \ldots, C'_q$ — компоненты слабой связности $E'_{\mathcal{S}}$. Понятно, что $q \leq p$, а для каждой компоненты C_i существует C'_j, что $C_i \subseteq C'_j$.

Сначала мы покажем, что вес оптимального сельского пути с конфигурацией f в DG равен весу оптимального сельского пути в DG'. Действительно, мы можем заменить опциональные дуги оптимального сельского пути P, совместного с конфигурацией f в графе DG, следующим образом: дугу $(\mathtt{AB}, \mathtt{CD})$ (веса 2) заменить тремя дугами $(\mathtt{AB}, \mathtt{B})$, $(\mathtt{B}, \mathtt{C})$, $(\mathtt{C}, \mathtt{CD})$ (суммарного веса $0+1+1=2$), дугу $(\mathtt{AB}, \mathtt{BC})$ (веса 1) — двумя дугами $(\mathtt{AB}, \mathtt{B})$, $(\mathtt{B}, \mathtt{BC})$ (суммарного веса $0+1=1$). Полученный путь P' — сельский путь графа DG': мы заменили ровно $f_{\text{out}}(\mathtt{AB})$ опциональных дуг из вершины $\mathtt{AB}$ новыми дугами $(\mathtt{AB}, \mathtt{B})$. Более того, этот путь имеет такой же вес. В обратную сторону: пусть P' — оптимальный сельский путь в графе DG'. Простым удалением всех вершин, помеченных одним символом, мы получим сельский путь P, совместный с f, вес которого не превосходит вес P'.

Теперь мы покажем, что оптимальный сельский путь в DG' может быть найден за полиномиальное время. Для этого покажем, что нам достаточно решить задачу отдельно на каждой компоненте слабой связности $E'_{\mathcal{S}}$.

Пусть P' — это оптимальный сельский путь в DG'. Переведем его обратно в путь P графа DG удалением всех вершин, помеченных одним символом. Пусть $P = A_1 A_2 \ldots A_k$, где каждый A_i принадлежит одной компоненте слабой связности C'_j графа $E'_{\mathcal{S}}$, а A_i и A_{i+1} принадлежат разным компонентам. Обозначим через u_i, v_i первую и последнюю вершины A_i (напомним, что

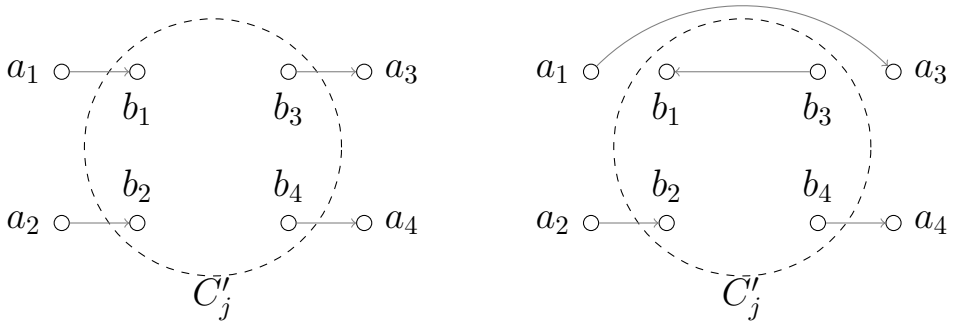

Рисунок 3.5 – Преобразование пути, применяемое в Лемме 3.5.2.

путь не содержит двух подряд идущих опциональных дуг). Важно заметить, что каждая дуга (v_i, u_{i+1}) имеет вес 2. Действительно, если $w(v_i, u_{i+1}) = 1$, то $v_i = \texttt{AB}$ и $u_{i+1} = \texttt{BC}$. Заметим, что (v_i, u_{i+1}) является опциональной дугой, т.к. v_i и u_{i+1} принадлежат различным компонентам слабой связности $E'_{\mathcal{S}}$ (а, следовательно, и различным компонентам слабой связности $E_{\mathcal{S}}$). Это означает, что $f_{\text{out}}(v_i) > 0$ и $f_{\text{in}}(u_{i+1}) > 0$. Но тогда дуги $(\texttt{AB}, \texttt{B})$ и $(\texttt{B}, \texttt{BC})$ обязательны в DG' и, следовательно, v_i и u_{i+1} принадлежат одной компоненте $E'_{\mathcal{S}}$.

Докажем, что существует оптимальный сельский путь P' в DG', который проходит по каждой компоненте $E'_{\mathcal{S}}$ последовательно. Для этого мы покажем, что если P' входит в одну компоненту $E'_{\mathcal{S}}$ более одного раза, то мы можем уменьшить количество опциональных дуг между компонентами (не увеличивая суммарный вес пути). Как и ранее, переведем путь P' обратно в путь P. Предположим, что путь P входит в компоненту C'_j хотя бы дважды, т.е. в P есть две дуги (a_1, b_1) и (a_2, b_2), что $b_1, b_2 \in C'_j$,а $a_1, a_2 \notin C'_j$. Предположим, что C'_j — не последняя компонента пути P (второй случай аналогичен). Это значит, что P должен выйти из компоненты C'_j хотя бы дважды. То есть, P содержит две опциональные дуги (b_3, a_3), (b_4, a_4), что $b_3, b_4 \in C'_j$, а $a_3, a_4 \notin C'_j$. Тогда заменим дуги (a_1, b_1) и (b_3, a_3) дугами (b_3, b_1) и (a_1, a_3) (см. Рисунок 3.5).

Легко видеть, что такое преобразование не изменяет степени вершин. Чтобы показать, что полученное множество дуг является путем, а не циклом и путем, мы заметим, что b_1, b_2, b_3, b_4 принадлежат одной компоненте слабой связности. Также, вес пути не увеличился (т.к. $w(a_1, b_1) = w(b_3, a_3) = 2$, а $w(b_3, b_1), w(a_1, a_3) \leq 2$).

Таким образом, чтобы найти оптимальный сельский путь в DG', нам достаточно найти оптимальный путь в каждой компоненте $E'_{\mathcal{S}}$ отдельно, а затем произвольным образом соединить найденные пути (напомним, что задача о сельском почтовом на слабосвязных графах решается за полиномиальное время). □

Доказательство Леммы 3.5.3. Пусть

$$\mathrm{mindeg}^{\mathrm{re}}(v) = \min\{d^{\mathrm{re}}_{\mathrm{in}}(v), d^{\mathrm{re}}_{\mathrm{out}}(v)\},\ \mathrm{mindeg}^{\mathrm{op}}(v) = \min\{f_{\mathrm{in}}(v), f_{\mathrm{out}}(v)\}.$$

По определению нормальной конфигурации (см. (3.3)), каждая компонента слабой связности содержит не более одной особенной вершины, т.е. вершины с $\mathrm{mindeg}^{\mathrm{op}} = 1$.

Рассмотрим два случая.

$\mathcal{C}$ является Эйлеровой компонентой связности. Тогда $\mathcal{C}$ содержит не более k вершин (содержит ровно k вершин, если это простой цикл). Если $E_{\mathcal{S}}$ состоит не только из $\mathcal{C}$, то $\mathcal{C}$ обязано содержать особенную вершину в кажом сельском пути, следовательно, количество различных нормальных конфигураций $\mathcal{C}$ равно k. Если же $E_{\mathcal{S}}$ равно $\mathcal{C}$, то оптимальный сельский путь может быть найден за полиномиальное время.

$\mathcal{C}$ не является Эйлеровой компонентой связности. Согласно (3.3), достаточно показать, что $\mathcal{C}$ содержит не более $(k-1)$ вершины с ненулевым $\mathrm{mindeg}^{\mathrm{re}}$. Тогда либо одна из этих вершин является особенной, либо в компоненте нет особенных вершин — не более k различных конфигураций.

Чтобы показать, что в $\mathcal{C}$ не более $(k-1)$ вершины с ненулевым $\mathrm{mindeg}^{\mathrm{re}}$, рассмотрим два случая.

1. Удалением направлений дуг $\mathcal{C}$ мы получим простой путь на k дугах. Тогда $\mathcal{C}$ содержит $(k+1)$ вершину, но начальная и конечная вершины имеют $\mathrm{mindeg}^{\mathrm{re}}$.
2. $\mathcal{C}$ содержит не более k вершин. Еси количество вершин строго меньше k, то оценка немедленно следует. Если же количество вершин равно k, то мы найдем вершину с нулевым $\mathrm{mindeg}^{\mathrm{re}}$. Для этого выберем произвольную вершину из $\mathcal{C}$ и начнем двигаться из нее. В итоге мы либо попадем в вершину нулевой выходящей степени (из чего сразу же следует оценка), либо найдем цикл. Т.к. $\mathcal{C}$ не является простым циклом, то одна из вершин цикла должна иметь сумму входящей и исходящей степеней хотя бы 3. Но тогда в $\mathcal{C}$ должна быть вершина с суммой степеней 1, что заканчивает доказательство.

□

Доказательство Леммы 3.5.4. Пусть $E_{\mathcal{S}}$ состоит из t компонент слабой связности $\mathcal{C}_1, \ldots, \mathcal{C}_t$, n_i — количество обязательных дуг в $\mathcal{C}_i$ ($n_1 + \cdots + n_t = n$). По Лемме 3.5.3, у $\mathcal{C}_i$ не более n_i различных нормальных конфигураций. Следовательно, общее количество нормальных конфигураций $E_{\mathcal{S}}$ не превосходит $\prod_{i=1}^{t} n_i$. Индукцией по n покажем, что это произведение не превосходит $3^{n/3}$.

База индукции $n = 1$ очевидна. Шаг индукции:

$$\prod_{i=1}^{t} n_i = n_t \cdot \prod_{i=1}^{t-1} n_i \leq 3^{\frac{n-n_t}{3}} n_t = 3^{\frac{n-n_t}{3}+\log_3 n_t}.$$

Это выражение не превосходит $3^{n/3}$, т.к. для любого $n_t \in \mathbb{N}$, $\log_3 n_t \leq n_t/3$.

Теперь мы можем перечислить все нормальные конфигурации: нам достаточно выбрать особенную вершину в каждой компоненте слабой связности. Действительно, если вершина $v \neq s, t$ — особенная вершина, то $\min\{f_{\text{in}}(v), f_{\text{out}}(v)\} = 1$, иначе $\min\{f_{\text{in}}(v), f_{\text{out}}(v)\} = 0$. Точные значения $f_{\text{in}}(v)$ и $f_{\text{out}}(v)$ могут быть получены из равенства (3.1). □

3.5.3 Пример работы алгоритма

В этом разделе мы приводим пример работы описанного алгоритма. Пусть

$$\mathcal{S} = \{\texttt{EFG}, \texttt{CDB}, \texttt{HFG}, \texttt{HGH}, \texttt{ABC}, \texttt{BCE}, \texttt{GHG}, \texttt{BCD}, \texttt{DBC}\}$$

— вход алгоритма. Множество $E_{\mathcal{S}}$ обязательных дуг показано на Рисунке 3.6(а). Любая надстрока $\mathcal{S}$ задает естественный путь в графе DG, проходящий по вем дугам $E_{\mathcal{S}}$ (сельский путь). Нам необходимо найти кратчайший путь P, удовлетворяющий этому условию.

Мы можем перечислить все возможные пары начальной и конечной вершин пути за время $O(n^2)$. Поэтому, далее мы будем рассматривать лишь пути, идущие из вершины `AB` в вершину `FG`.

Мы не знаем какие опциональные дуги входят в путь P, но мы знаем, что этот путь содержит хотя бы одну опциональную дугу, выходящую из `CE` (т.к. P должен входить в `CE` по обязательной дуге). Лемма 3.5.1 утверждает, что существует оптимальный путь, содержащий ровно одну опциональную дугу, выходящую из `CE` (т.к. мы могли бы уменьшить количество опциональных дуг, если бы `CE` имел и входящие и исходящие опциональные дуги). Одна из вершин `GH` и `HG` должна быть особенной в пути P (т.е., должна иметь входящую и исходящую опциональные дуги).

По Лемме 3.5.4, количество нормальных конфигураций не превосходит $3^{n/3}$. В нашем примере ровно 8 нормальных конфигураций: первая компонента либо не содержит особенных вершин, либо особенной вершиной является одна из `BC`, `CD`, `DB`, в третьей компоненте — одна из вершин `HG` и `GH` должна быть особенной.

Рассмотрим одну из этих конфигураций (см. Таблицу 3.1): В этой конфигурации только вершина `HG` является особенной. Чтобы найти оптимальный

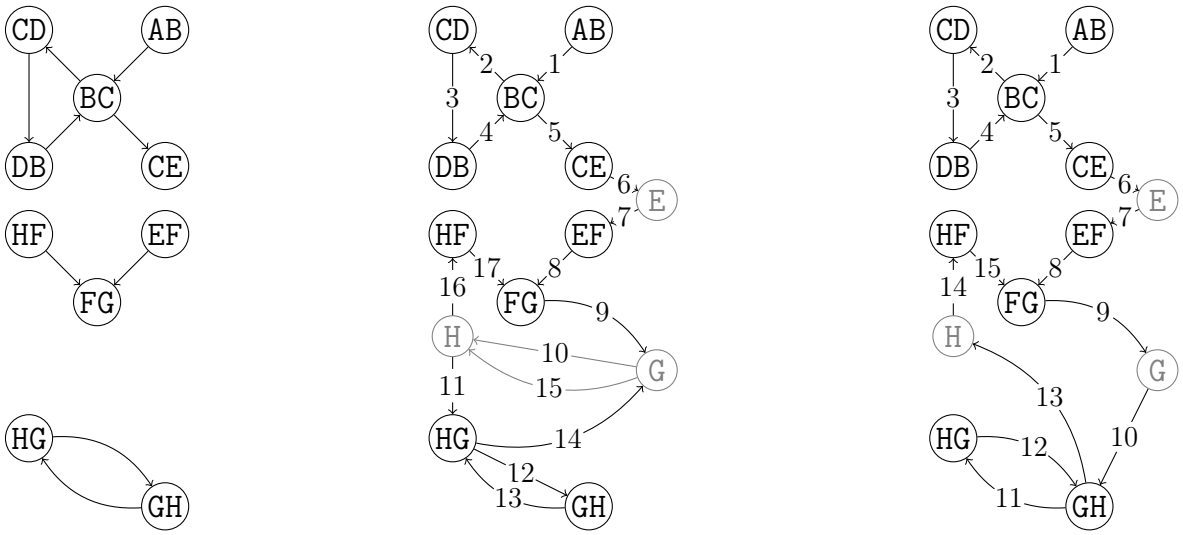

Рисунок 3.6 – (a) граф де Брюйна множества строк {EFG, CDB, HFG, HGH, ABC, BCE, GHG, BCD, DBC} (показаны только обязательные дуги). (b) Эйлеров путь, входящий в нижнюю компоненту через вершину HG, задает надстроку ABCDBCEFGHGHGHFG длины 16. (c) Кратчайший путь входит в нижнюю компоненту через вершину GHи задает строку ABCDBCEFGHGHFG длины 14

Таблица 3.1 – Пример нормальной конфигурации

	AB	BC	CD	DB	CE	HF	FG	EF	HG	GH
f_{in}	0	0	0	0	0	1	0	1	1	0
f_{out}	0	0	0	0	1	0	1	0	1	0

сельский путь, совместный с рассматриваемой конфигурацией, мы добавляем вершины, соответствующие однобуквенным строкам, каждой исходной строке мы добавляем соответствующее количество входящих и исходящих дуг. В полученном графе только однобуквенные вершины не сбалансированны. Мы балансируем степени однобуквенных вершин произвольным образом и находим Эйлеров цикл в полученном графе. Для рассматриваемой конфигурации мы получим надстроку ABCDBCEFGHGHGHFG длины 16 (см. Рисунок 3.6(b), дуги, добавленные для баланса степеней, показаны серым). Лемма 3.5.2 показывает, как восстановить путь по конфигурации.

Если мы рассмотрим другую конфигурацию, в которой вершина GH (а не вершина HG) является особенной в третьей компоненте связности, то мы получим кратчайшую надстроку ABCDBCEFGHGHFG длины 14, которая и является

кратчайшей надстрокой на данном входе (см. Рисунок 3.6(c)).

3.6 Задача об r-надстроке

В этом разделе мы покажем, что для любой константы r, задача о кратчайшей r-надстроке может быть решена за время $O^*(2^{(1-c(r))n})$, где $c(r) = (1 + 2r^2)^{-1}$. Для этого мы введем понятие иерархического графа, которое позволит нам свести задачу о надстроке строк длины r к задаче о сельском почтовом с $k = (1 - c(r))n$ компонентами слабой связности. После мы используем алгоритм Gutin, Wahlström, Yeo [42], решающий задачу о сельском почтовом за время $O^*(2^k)$.

3.6.1 Иерархические графы

Определение 3.6.1. Иерархический граф $HG_{\mathcal{S}} = (V, A)$ множества строк $\mathcal{S}$ — это взвешенный ориентированный граф:

- множество вершин V которого состоит из всех префиксов и суффиксов (включая пустую строку ε) строк из $\mathcal{S}$,
- между двумя вершинами $u, v \in V$, есть дуга $(u, v) \in A$, когда
  - либо u — префикс v длины $|v|-1$; вес дуги в этом случае $w(u, v) = 1$,
  - либо v — суффикс u длины $|u|-1$; в этом случае вес дуги $w(u, v) = 0$.

Для множества дуг $A' \subseteq A$ введем следующие обозначения:

$$
\begin{aligned}
N^{up}_{in}(v, A') &= \{(u, v) \in A' \colon |u| = |v| + 1\}, \\
N^{down}_{in}(v, A') &= \{(u, v) \in A' \colon |u| = |v| - 1\}, \\
N^{up}_{out}(v, A') &= \{(v, u) \in A' \colon |u| = |v| + 1\}, \\
N^{down}_{out}(v, A') &= \{(v, u) \in A' \colon |u| = |v| - 1\}.
\end{aligned}
$$

Пусть также $d^{up}_{in}(v, A') = |N^{up}_{in}(v, A')|$. d^{down}_{in}, d^{up}_{out} и d^{down}_{out} определяются аналогично. Для вершины v дуги из $N^{up}_{in}(v, A')$ и $N^{up}_{out}(v, A')$ называются верхними, а дуги $N^{down}_{in}(v, A')$ и $N^{down}_{out}(v, A')$ — нижними. Пример иерархического графа и верхних и нижних дуг приведен на Рисунке 3.7.

В построенном графе мы ищем кратчайший путь из ε в ε, проходящий по всем вершинам из $\mathcal{S}$. Легко видеть, что длина пути из ε в ε равняется длине строки, соответствующей этому пути (т.к. каждая дуга, идущая вверх, имеет вес 1 и добавляет один символ в строку). Напрмиер, в графе на Рисунке. 3.7

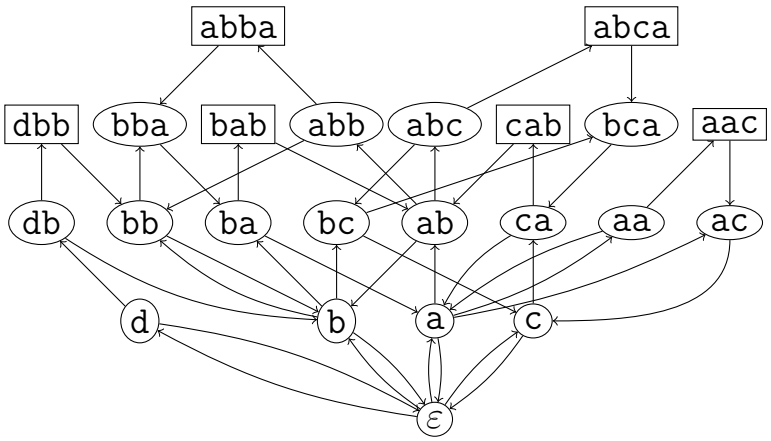

Рисунок 3.7 – Иерархический граф для множества строк $\mathcal{S} = \{\texttt{abba}, \texttt{abca}, \texttt{dbb}, \texttt{bab}, \texttt{cab}, \texttt{aac}\}$. Исходные строки $\mathcal{S}$ показаны в прямоугольниках

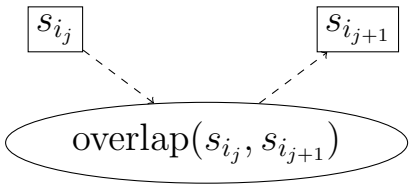

Рисунок 3.8 – Естественный путь между вершинами s_{i_j} и $s_{i_{j+1}}$

путь $\varepsilon \to \texttt{b} \to \texttt{ba} \to \texttt{bab} \to \texttt{ab} \to \texttt{abc} \to \texttt{bc} \to \texttt{c} \to \varepsilon$ имеет длину 4, ему соответсвует строка `babc` длины 4.

Заметим, что каждая строка из $\mathcal{S}$ имеет ровно одну входящую и одну исходящую дугу в $HG_{\mathcal{S}}$. Обозначим множество этих дуг через R. Любой оптимальный путь должен проходить по дугам R, поэтому мы называем эти дуги обязательными. Формально, $R = \{(pref(s), s) \colon s \in \mathcal{S}\} \cup \{(s, suf(s)) \colon s \in \mathcal{S}\}$. Полученная задача — это задача о сельском почтовом с одним лишь исключением: оптимальный путь в нашей задаче также должен проходить через вершину ε. Для этого мы вводим обязательную дугу веса 0 из вершины ε в себя же.

Для пары вершин u и v, которые не являются подстроками друг друга, назовем следующий путь естественным: из вершины u путь сначала идет вниз в вершину overlap(u, v), а после — вверх в вершину v (см. Рисунок 3.8). Мы называем сельский путь нормальным, если между любыми двумя подряд идущими вершинами из $\mathcal{S}$ он меняет направление только один раз (т.е., любой подпуть между подряд идущими вершинами из $\mathcal{S}$ является естественным).

Легко видеть, что всегда существует нормальный оптимальный путь (напомним, что никакая входная строка $s \in \mathcal{S}$ не является подстрокой другой входной строки $s' \in \mathcal{S}$).

Если все пересечения строки $s \in \mathcal{S}$ с другими строками из $\mathcal{S}$ короткие, то оптимальный сельский путь имеет длинный подпуть вниз из вершины s. На Рисунке 3.7 ни одна строка из $\mathcal{S}$ не начинается с `bb`. Поэтому любой оптимальный путь в $HG_{\mathcal{S}}$ должен идти вниз из `dbb` хотя бы до b. Определим это формально:

Определение 3.6.2. Обозначим через $\mathrm{maxpref}_{\mathcal{S}}(s)$ ($\mathrm{maxsuf}_{\mathcal{S}}(s)$) самый длинный префикс (суффикс) строки $s \in \mathcal{S}$, который также является суффиксом (префиксом) некоторой другой строки из $\mathcal{S}$. Очевидно, что

$$\begin{aligned} |\,\mathrm{maxpref}_{\mathcal{S}}(s)| &= \max\{|\,\mathrm{overlap}(s', s)| \colon s' \in \mathcal{S} \setminus \{s\}\}\,, \\ |\,\mathrm{maxsuf}_{\mathcal{S}}(s)| &= \max\{|\,\mathrm{overlap}(s, s')| \colon s' \in \mathcal{S} \setminus \{s\}\}\,. \end{aligned}$$

Например, для $\mathcal{S} = \{\mathtt{abba}, \mathtt{abca}, \mathtt{dbb}, \mathtt{bab}, \mathtt{cab}, \mathtt{aac}\}$, $\mathrm{maxpref}_{\mathcal{S}}(\mathtt{abca}) = \mathtt{ab}$, $\mathrm{maxsuf}_{\mathcal{S}}(\mathtt{abca}) = \mathtt{ca}$, $\mathrm{maxpref}_{\mathcal{S}}(\mathtt{dbb}) = \varepsilon$, $\mathrm{maxsuf}_{\mathcal{S}}(\mathtt{dbb}) = \mathtt{b}$.

Определение 3.6.3. Расширенным множеством обязательных дуг $ER \subseteq E(HG_{\mathcal{S}})$ мы называем объединение множества обязательных дуг R и следующих множеств:

- пути вверх к s: все дуги (u, v), где $u, v \sqsupset s$, $|u| = |v| - 1$ и $|u| \geq |\,\mathrm{maxpref}(s)|$;
- пути вниз из s: все дуги (u, v), где $u, v \sqsubset s$, $|u| = |v| + 1$ и $|v| \geq |\,\mathrm{maxsuf}(s)|$;
- цикл $(\varepsilon, \varepsilon)$ веса 0.

Пример расширенного множества обязательных дуг приведен на Рисунке 3.9(b).

Лемма 3.6.1. *Длина кратчайшей надстроки множества строк $\mathcal{S}$ равна длине оптимального сельского пути в графе $HG_{\mathcal{S}}$ со множеством обязательных дуг ER.*

Доказательство. Представим оптимальный сельский пути w как последовательность вершин $v_0 = \varepsilon, v_1, \ldots, v_k = \varepsilon$. Этот путь естественным образом задает строку s: сначала $s = \varepsilon$; каждый раз, когда путь поднимается вверх (т.е., $|v_i| = |v_{i-1}| + 1$) — добавим соответствующий символ в строку s. Таким образом мы поддерживаем два следующих инварианта:

- длина текущей строки равна длине пройденного пути;
- в вершине v_i текущая строка s содержит v_i как суффикс.

Т.к. w проходит по всем вершинам из $\mathcal{S}$, полученная строка s является надстрокой множества $\mathcal{S}$. Также, длина s равна длине w. Следовательно, длина кратчайшей надстроки не превосходит длины оптимального сельского пути.

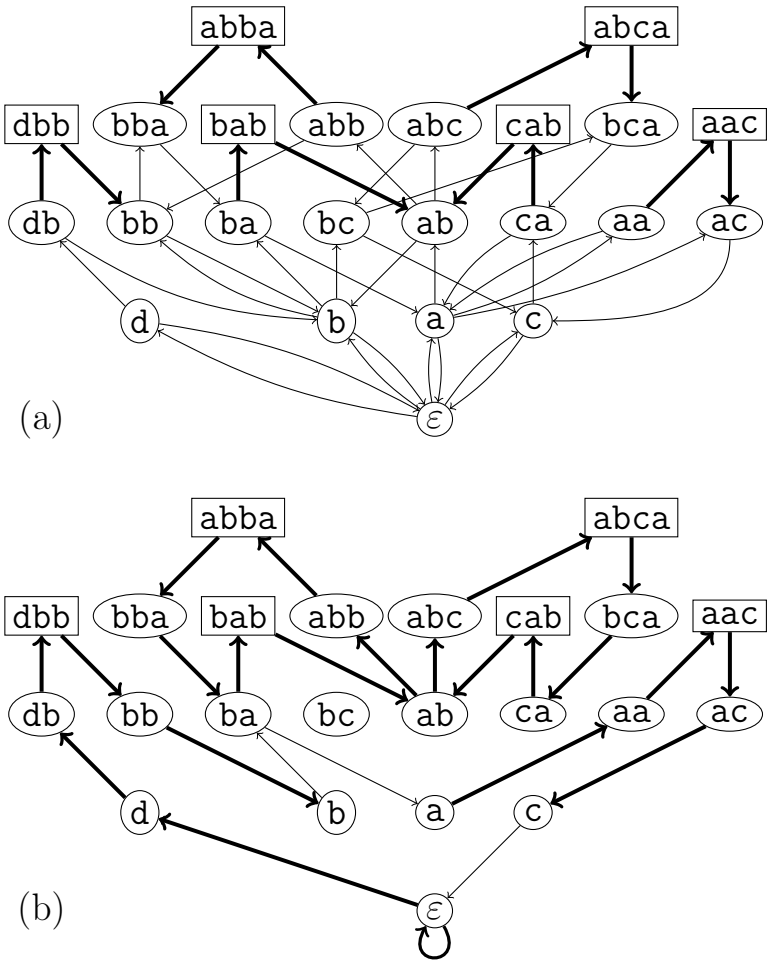

Рисунок 3.9 – (a) Множество R обязательных дуг показано жирным. (b) Кратчайшая надстрока `dbbabcabbaac` **задает путь длины 12 в графе HG. Заметим, что кратчайшая надстрока и оптимальный пути задаются перестановкой** $\sigma = (\texttt{dbb}, \texttt{bab}, \texttt{abca}, \texttt{cab}, \texttt{abba}, \texttt{aac})$**. Расширенное множество обязательных дуг** ER **показано жирным**

В обратную сторону: рассмотрим надстроку s множества исходных строк $\mathcal{S}$. s задает порядок $s_{i_1}, \ldots, s_{i_n}$ исходных строк. Рассмотрим следующий сельский путь. Путь начинается в вершине ε, идет вверх до вершины s_{i_1}, после для всех $j = 1, \ldots, n-1$ путь идет вних из вершины s_{i_j} в вершину $\mathrm{overlap}(s_{i_j}, s_{i_{j+1}})$ и вверх в вершину $s_{i_{j+1}}$, в конце путь идет из вершины s_{i_n} в ε. Легко видеть, что длина полученного пути равна длине s. Это также корректный сельский путь, т.к.для любой s_{i_j}

$$\begin{aligned} |\operatorname{overlap}(s_{i_{j-1}}, s_{i_j})| &\leq |\operatorname{maxpref}_{\mathcal{S}}(s_{i_j})|, \\ |\operatorname{overlap}(s_{i_j}, s_{i_{j+1}})| &\leq |\operatorname{maxsuf}_{\mathcal{S}}(s_{i_j})|. \end{aligned}$$

Следовательно, путь точно проходит по всем дугам из ER. Таким образом, длина оптимального сельского пути не превосходит длины кратчайшей надстроки. □

Определение 3.6.4. Нижняя вершина графа $HG_{\mathcal{S}}$ — это вершина, у которой нет нижних дуг в ER, но есть хотя бы одна верхняя дуга в ER: $d_{in}^{down}(v, ER) = d_{out}^{down}(v, ER) = 0$ и $d_{in}^{up}(v, ER) + d_{out}^{up}(v, ER) \geq 1$. Обозначим множество нижних вершин через V_b.

Лемма 3.6.2.

$$V_b = \{\text{maxpref}_{\mathcal{S}}(s), \text{maxsuf}_{\mathcal{S}}(s) \colon s \in \mathcal{S}\}.$$

Доказательство. Очевидно, каждая нижняя вершина является $\text{maxpref}_{\mathcal{S}}(s)$ или $\text{maxsuf}_{\mathcal{S}}(s)$ какой-то входной строки $s \in \mathcal{S}$. В обратную сторону: рассмотрим вершину $v = \text{maxsuf}_{\mathcal{S}}(s)$, предположим, что в нее входит верхняя дуга $(u, v) \in ER$. Эта дуга обязана принадлежать пути из $\text{maxpref}_{\mathcal{S}}(t)$ вверх в t для какой-то строки $t \in \mathcal{S}$. Тогда $v \sqsubset t$ и $v \sqsupset s$, и v длиннее $\text{maxpref}_{\mathcal{S}}(t)$, что противоречит определению $\text{maxpref}_{\mathcal{S}}(t)$. Аналогичным образом можно показать, что у v нет выходящих нижних дуг. Следовательно, $\text{maxsuf}_{\mathcal{S}}(s)$ действительно является нижней вершиной. Аналогично, $\text{maxpref}_{\mathcal{S}}(s)$ тоже является нижней вершиной. □

Например, для множества $\mathcal{S}$ на Рисунке 3.7, $\{\text{maxpref}_{\mathcal{S}}(s) \colon s \in \mathcal{S}\} = \{\varepsilon, \texttt{ba}, \texttt{ab}, \texttt{ca}, \texttt{a}\}$ $\{\text{maxsuf}_{\mathcal{S}}(s) \colon s \in \mathcal{S}\} = \{\texttt{b}, \texttt{ab}, \texttt{ca}, \texttt{ba}, \texttt{c}\}$.

Определение 3.6.5. Нижняя вершина называется хорошей, если она не является подстрокой никакой другой нижней вершины. Множество хороших вершин мы обозначаем через V_g.

На Рисунке 3.9(b) нижние вершины — ε, b, ba, ab, ca, a, c. Из них вершины ba, ab иca являются хорошими.

Лемма 3.6.3. $r^2|V_g| \geq |V_b|$.

Доказательство. Напомним, что нижняя вершина v не является хорошей тогда и только тогда, когда существует другая нижняя вершина u, что v — подстрока u. Другими словами, из вершины v в графе HG существует путь вверх в вершину u. Это позволяет рекурсивно опредлить следующее отображение: $f \colon V_b \to V_g$: если $v \in V_b$ — хорошоая, то $f(v) = v$; иначе пусть $u \in V_b$ — такая вершина, что v является подстрокой u; $f(v) = f(u)$ (это возможно, т.к. $|u| > |v|$). Таким образом, мы поднимаемся вверх из вершины v пока не достигнем хорошей вершины. Отметим, что $v \in V_b$ всегда является подстрокой $f(v)$. Т.к. любая строка длины r имеет не более r^2 подстрок, $r^2|V_g| \geq |V_b|$. □

Теорема 3.6.1. *Расширенное множество обязательных дуг ER состоит из не более чем $(1 - \frac{1}{2r^2+1})n$ компонент слабой связности.*

Доказательство. Пусть k — общее количество компонент слабой связности ER, а m_i — количество компонент слабой связности, которые содержат ровно i строк из $\mathcal{S}$. Тогда

$$\sum_{i=1}^{n} m_i = k \text{ и } \sum_{i=1}^{n} i m_i = n\,.$$

Каждая компонента слабой связности содержит хотя бы одну нижнюю вершину, следовательно, $k \geq |V_b|$.

Также, каждая компонента слабой связности, содержащая i входных строк, содержит не более i хороших вершин. Действительно, хорошая вершина — это вершина, в которой встречаются путь из $s \in \mathcal{S}$ и путь в $s' \in \mathcal{S}$. В то же время, каждой строке $s \in \mathcal{S}$ соответствует не более двух хороших вершин (по Лемме 3.6.2, одна $\mathrm{maxpref}_{\mathcal{S}}(s)$ и одна $\mathrm{maxsuf}_{\mathcal{S}}(s)$). Следовательно, $\sum_{i=2}^{n} i m_i \geq |V_g|$ (компоненты, не содержащие входных строк, не содержат и хороших вершин).

Используя полученные оценки и Лемму 3.6.3, получаем:

$$\begin{aligned} n = \sum_{i=1}^{n} i m_i = k + \sum_{i=1}^{n} (i-1) m_i \geq k + \sum_{i=2}^{n} \frac{i m_i}{2} \geq \\ \geq k + \frac{|V_g|}{2} \geq k + \frac{|V_b|}{2r^2} \geq k + \frac{k}{2r^2} = k\left(1 + \frac{1}{2r^2}\right). \end{aligned}$$

□

3.6.2 Описание алгоритма

Теорема 3.6.2. *Существует вероятностный алгоритм с односторонней ошибкой, решающий задачу r-SCS на n строках за время* $O^*\left(2^{\left(1-\frac{1}{2r^2+1}\right)n}\right)$.

Доказательство. По Лемме 3.6.1, для поиска кратчайшей надстроки множества строк $\mathcal{S}$ достаточно найти кратчайший сельский путь в графе $HG_{\mathcal{S}}$ с множеством обязательных дуг ER. По Теореме 3.6.1, ER содержит не более $k \leq (1 - \frac{1}{1+2r^2})n$ компонент слабой связности. Наконец, по Теореме 3.4.1, за время 2^k можно проверить существует ли в графе сельский пути длины $l = O^*(1)$. В графе ER $l = O^*(1)$, т.к. длина кратчайшей надстроки множества $\mathcal{S}$ не превосходит rn. □

Замечание 3.6.1. Предложенный алгоритм поднимает следующий вопрос: можно ли решить общий случай задачи о кратчайшей надстроке за время $O^*((2-\varepsilon)^n)$? Также было бы интересно получить результаты о сложности

решения общего случая задачи, такие как сведение ЗК к SCS или сложность в предположении сильной гипотезы экспоненциального времени.

ЗАКЛЮЧЕНИЕ

Основные научные результаты диссертации

1. Для параметризованной задачи MAX-SAT получена новая верхняя оценка 1.3579^k. Эта оценка является первым улучшением оценки 1.3695^k, доказанной Chen и Kanj более 10 лет назад. Для задач MAX-2-SAT и MAX-2-CSP получены верхние оценки $O^*(2^{n(1-\frac{10/3}{d+1})})$ и $O^*(2^{n(1-\frac{3}{d+1})})$, соответственно. Предложенный алгоритм улучшает известную оценку $O^*(2^{n(1-\frac{2}{d+1})})$, доказанную Scott и Sorkin. Мы доказываем, что элементарная реализация алгоритма, производящая лишь расщепления по переменным максимальной степени, имеет рекордное время выполнения в худшем случае на задачах MAX-2-SAT и MAX-2-CSP. Также мы показываем, что предложенный алгоритм эффективно решает задачи MAX-2-SAT и MAX-2-CSP на формулах высокой средней степени, а также задачи, параметризованные количеств клозов формулы [91, 92, 93, 94, 97].
2. Для задач MAX-2-SAT и MAX-2-CSP получена верхняя оценка $O(2^{c_d n})$, где $c_d = 1 - \frac{2\alpha \ln d}{d}$, $\alpha < 1$ — константа, для случая, когда $d = \Omega(1); d = o(n)$. Важно отметить, что экспонента времени выполнения предложенного алгоритма (как функция от средней степени переменной) растет асимптотически медленнее, чем у всех известных алгоритмов решения рассматриваемых задач [91].
3. Для задачи коммивояжера получен алгоритм, который для любого $0 < \varepsilon < 1$, находит $(1+\varepsilon)$-приближение за время $O^*(2^n \varepsilon^{-1})$, используя память $O^*(\varepsilon^{-1})$. Предыдущие алгоритмы приближения задачи коммивояжера либо находили слишком плохое приближение, либо использовали экспоненциальную память, что делало их совершенно неприменимыми на практике. Мы предлагаем приближенный алгоритм, который использует лишь полиномиальную память, что делает поиск приближенного решения задачи о коммивояжере задачей, разрешимой на практике. Лучшее известное полиномиальное приближение даже для неориентированной метрической версии задачи коммивояжера — 1.5 (для ориентированной — $O(\frac{\log n}{\log \log n})$). Если P≠NP, то за полиномиальное время не может быть найдено $\frac{117}{116}$-приближение неориентированной метрической задачи коммивояжера. Если нам необходимо найти, например, $\frac{1001}{1000}$-приближение, то мы должны использовать точные алгоритмы, которые требуют либо экспоненциальную память, либо время $4^n n^{\log n}$. Экспоненциальные алгоритмы редко используются на практике, но в

любом случае, мы можем их запустить и дождаться ответа. Однако, если алгоритм использует экспоненциальную память, то мы не имеем даже возможности запустить алгоритм на входах разумного размера. Предложенный алгоритм может найти 1001/1000-приближение общей задачи коммивояжера за время $O^*(2^n)$, используя лишь полиномиальную память [89].

4. Для задачи о 3-надстроке получена верхняя оценка $3^{n/3}$, разработан алгоритм Монте-Карло с односторонней ошибкой, решающий задачу об r-надстроке за время $2^{n\left(1-\frac{1}{1+2r^2}\right)}$. Лучший известный алгоритм решения задачи о 3-надстроке имел время выполнения 2^n, а предложенный нами алгоритм выполняется за время $3^{n/3} \approx 1.45^n$. Такой алгоритм позволяет решать задачу о кратчайшей 3-надстроке для значительно больших размеров входных данных [90, 95, 96].

Рекомендации по практическому использованию результатов

В то время, как новые идеи доказательства верхних оценок на время работы алгоритмов имеют, скорее, теоретическую ценность, полученные алгоритмы могут успешно использоваться на практике. Главной целью диссертационной работы является получение экспоненциального выигрыша во времени решения NP-трудных задач. С этой целью в работе были рассмотрены задачи, обобщающие многие известные задачи из класса NP. Предложенные алгоритмы решения рассматриваемых задач также эффективны для задач, сводимых к рассматриваемым. Так, например, алгоритмы решения задач MAX-2-CSP распространяются и на решение задачи о максимальном разрезе, а приближенная схема решения задачи о коммивояжере дает эффективные алгоритмы приближения задачи о кратчайшей общей надстроке. В диссертационной работе мы предложили алгоритмы, улучшающие экспоненту времени выполнения NP-трудных задач. Интересно заметить, что многократное увеличение вычислительных мощностей не дает такого ускорения во времени выполнения, как новый алгоритм с меньшей экспонентой во времени выполнения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

Список использованных источников

1. Algorithms for three versions of the shortest common superstring problem / M. Crochemore, M. Cygan, C. Iliopoulos [et al] // Proceedings of the 21st annual conference on Combinatorial pattern matching. — CPM'10. — Springer-Verlag, 2010. — P. 299–309.

2. Alon, N. Large induced degenerate subgraphs / N. Alon, J. Kahn, P. D. Seymour // Graphs and Combinatorics. — 1987. — Vol. 3, № 1. — P. 203–211.

3. An algorithm for the Rural Postman problem on a directed graph / N. Christofides, V. Campos, A. Corberan, E. Mota // Netflow at Pisa. — Springer Berlin Heidelberg, 1986. — Vol. 26 of Mathematical Programming Studies. — P. 155–166.

4. An $O(\log n/\log\log n)$-approximation Algorithm for the Asymmetric Traveling Salesman Problem / A. Asadpour, M. X. Goemans, A. Madry [et al] // Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms. — SODA'10. — Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010. — P. 379–389.

5. Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular Multigraphs / H. Kaplan, M. Lewenstein, N. Shafrir, M. Sviridenko // J. ACM. — 2005. — Vol. 52. — P. 602–626.

6. Arora, S. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems / S. Arora // In Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS'96). — 1996. — P. 2–11.

7. Arora, S. Nearly Linear Time Approximation Schemes for Euclidean TSP and Other Geometric Problems / S. Arora // In Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS'97). — 1997. — P. 554–563.

8. Bansal, N. Upper bounds for maxsat: Further improved / N. Bansal, V. Raman // ISAAC'99. — Springer, 1999. — P. 247–258.

9. Bax, E. A Finite-Difference Sieve to Count Paths and Cycles by Length / E. Bax, J. Franklin // Inf. Process. Lett. — 1996. — Vol. 60. — P. 171–176.

10. Bax, E. T. Recurrence-Based Reductions for Inclusion and Exclusion Algorithms Applied to #P Problems / E. T. Bax. — 1996.

11. Beigel, R. 3-coloring in time $O(1.3289^n)$ / R. Beigel, D. Eppstein // Journal of Algorithms. — 2005. — Vol. 54, № 2. — P. 168–204.

12. Bellman, R. Dynamic Programming Treatment of the Travelling Salesman Problem / R. Bellman // J. ACM. — 1962. — Vol. 9. — P. 61–63.

13. Berman, P. 8/7-approximation Algorithm for (1,2)-TSP / P. Berman, M. Karpinski // Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. — SODA'06. — New York, NY, USA: ACM, 2006. — P. 641–648.

14. Binkele-Raible, D. A new upper bound for max-2-sat: A graph-theoretic approach / D. Binkele-Raible, H. Fernau // Journal of Discrete Algorithms. — 2010. — Vol. 8, № 4. — P. 388–401.

15. Björklund, A. Determinant Sums for Undirected Hamiltonicity / A. Björklund // Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science. — FOCS'10. — Washington, DC, USA: IEEE Computer Society, 2010. — P. 173–182.

16. Björklund, A. Exact Algorithms for Exact Satisfiability and Number of Perfect Matchings / A. Björklund, T. Husfeldt // Algorithmica. — 2008. — Vol. 52. — P. 226–249.

17. Bläser, M. A New Approximation Algorithm for the Asymmetric TSP with Triangle Inequality / M. Bläser // Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. — SODA'03. — Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. — P. 638–645.

18. Bliznets,. A New Upper Bound for $(n, 3)$-MAX-SAT / Bliznets // Zapiski nauchnikh seminarov POMI. — 2012. — P. 5–14.

19. Calabro, C. The complexity of satisfiability of small depth circuits / C. Calabro, R. Impagliazzo, R. Paturi // Parameterized and Exact Computation. — Springer, 2009. — P. 75–85.

20. Chen, J. Improved Exact Algorithms for Max-Sat / J. Chen, I. Kanj // LATIN 2002: Theoretical Informatics. — Springer, 2002. — Vol. 2286 of LNCS. — P. 98–119.

21. Chen, J. Improved Exact Algorithms for Max-Sat / J. Chen, I. Kanj // Discrete Applied Mathematics. — 2004. — Vol. 142, № 1-3. — P. 17–27.

22. Christofides, N. Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem: Tech. Rep. 338 / N. Christofides: Graduate School of Industrial Administration, CMU, 1976.

23. Croce, F. D. An exact algorithm for MAX-CUT in sparse graphs / F. D. Croce, M. Kaminski, V. Paschos // Operations Research Letters. — 2007. — Vol. 35, № 3. — P. 403 – 408.

24. Cygan, M. Faster exponential-time algorithms in graphs of bounded average degree / M. Cygan, M. Pilipczuk // Automata, Languages, and Programming / Ed. by F. Fomin, R. Freivalds, M. Kwiatkowska, D. Peleg. — Springer Berlin Heidelberg, 2013. — Vol. 7965 of Lecture Notes in Computer Science. — P. 364–375.

25. Dantsin, E. MAX-SAT for formulas with constant clause density can be solved faster than in $O(2^n)$ time / E. Dantsin, A. Wolpert // Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing. — Vol. 4121 of LNCS. — 2006. — P. 266–276.

26. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth / H. L. Bodlaender, M. Cygan, S. Kratsch, J. Nederlof // Automata, Languages, and Programming. — Springer, 2013. — P. 196–207.

27. Eiselt, H. A. Arc Routing Problems, Part II: The Rural Postman Problem / H. A. Eiselt, M. Gendreau, G. Laporte // Operations Research. — 1995. — Vol. 43, № 3. — P. 399–414.

28. Eppstein, D. The Traveling Salesman Problem for Cubic Graphs / D. Eppstein // Algorithms and Data Structures. — Springer Berlin / Heidelberg, 2003. — Vol. 2748 of LNCS. — P. 307–318.

29. Exponential Approximation Schemata for Some Network Design Problems: Cahier du LAMSADE 303 / N. Boria, N. Bourgeois, B. Escoffier, V. T. Paschos. — Universite Paris-Dauphine: LAMSADE, 2011.

30. Feige, U. Balanced coloring of bipartite graphs / U. Feige, S. Kogan // J. Graph Theory. — 2010. — Vol. 64, № 4. — P. 277–291.

31. Feige, U. Improved Approximation Ratios for Traveling Salesperson Tours and Paths in Directed Graphs / U. Feige, M. Singh // Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques. — Springer Berlin / Heidelberg, 2007. — Vol. 4627 of LNCS. — P. 104–118.

32. Frieze, A. M. On the Worst-Case Performance of Some Algorithms for the Asymmetric Traveling Salesman Problem / A. M. Frieze, G. Galbiati, F. Maffioli // Networks. — 1982. — Vol. 12, № 1. — P. 23–39.

33. Fürer, M. Exact Max 2-Sat: Easier and Faster / M. Fürer, S. P. Kasiviswanathan // Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science. — SOFSEM 2007. — Springer-Verlag, 2007. — P. 272–283.

34. Gallant, J. On finding minimal length superstrings / J. Gallant, D. Maier, J. A. Storer // Journal of Computer and System Sciences. — 1980. — Vol. 20, № 1. — P. 50–58.

35. Garey, M. R. “Strong” NP-Completeness Results: Motivation, Examples, and Implications / M. R. Garey, D. S. Johnson // J. ACM. — 1978. — Vol. 25. — P. 499–508.

36. Garey, M. R. Computers and Intractability: A Guide to the Theory of NP-Completeness / M. R. Garey, D. S. Johnson. — New York, NY, USA: W. H. Freeman & Co., 1979.

37. Gaspers, S. A universally fastest algorithm for max 2-sat, max 2-csp, and everything in between / S. Gaspers, G. B. Sorkin // Journal of computer and system sciences. — 2012. — Vol. 78, № 1. — P. 305–335.

38. Gebauer, H. Finding and Enumerating Hamilton Cycles in 4-Regular Graphs / H. Gebauer // Theoretical Computer Science. — 2011. — Vol. 412, № 35. — P. 4579–4591.

39. Gharan, S. O. A Randomized Rounding Approach to the Traveling Salesman Problem / S. O. Gharan, A. Saberi, M. Singh // FOCS. — 2011. — P. 550–559.

40. Groves, G. Efficient heuristics for the Rural Postman Problem / G. Groves, J. van Vuuren // ORiON. — 2005. — Vol. 21, № 1. — P. 33–51.

41. Gurevich, Y. Expected Computation Time for Hamiltonian Path Problem / Y. Gurevich, S. Shelah // SIAM J. Comput. — 1987. — Vol. 16. — P. 486–502.

42. Gutin, G. Parameterized rural postman and conjoining bipartite matching problems / G. Gutin, M. Wahlström, A. Yeo // arXiv preprint arXiv:1308.2599. — 2013.

43. Håstad, J. Some optimal inapproximability results / J. Håstad // J. ACM. — 2001. — Vol. 48, № 4. — P. 798–859.

44. Held, M. The Traveling-Salesman Problem and Minimum Spanning Trees / M. Held, R. M. Karp // Mathematical Programming. — 1971. — Vol. 1. — P. 6–25.

45. Hertli, T. 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General / T. Hertli // Foundations of Computer Science (FOCS). — 2011. — P. 277 –284.

46. Hirsch, E. A. A new algorithm for max-2-sat / E. A. Hirsch // STACS 2000 / Springer. — 2000. — P. 65–73.

47. Ibarra, O. H. Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems / O. H. Ibarra, C. E. Kim // J. ACM. — 1975. — Vol. 22. — P. 463–468.

48. Impagliazzo, R. Which problems have strongly exponential complexity? / R. Impagliazzo, R. Paturi, F. Zane // Journal of Computer and System Sciences. — 2001. — Vol. 63, № 4. — P. 512–530.

49. Iwama, K. An Improved Exact Algorithm for Cubic Graph TSP / K. Iwama, T. Nakashima // Computing and Combinatorics. — Springer Berlin / Heidelberg, 2007. — Vol. 4598 of LNCS. — P. 108–117.

50. Karp, R. M. Dynamic Programming Meets the Principle of Inclusion and Exclusion / R. M. Karp // Operations Research Letters. — 1982. — Vol. 1, № 2. — P. 49–51.

51. Karpinski, M. Improved Lower Bounds for the Shortest Superstring and Related Problems / M. Karpinski, R. Schmied // CoRR. — 2011. — Vol. abs/1111.5442.

52. Kohn, S. A Generating Function Approach to the Traveling Salesman Problem / S. Kohn, A. Gottlieb, M. Kohn // ACN'77: Proceedings of the 1977 annual conference. — New York, NY, USA: 1977. — P. 294–300.

53. Koivisto, M. Optimal 2-constraint satisfaction via sum–product algorithms / M. Koivisto // Information processing letters. — 2006. — Vol. 98, № 1. — P. 24–28.

54. Koivisto, M. A Space-Time Tradeoff for Permutation Problems / M. Koivisto, P. Parviainen // Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms. — SODA'10. — Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010. — P. 484–492.

55. Kojevnikov, A. A new approach to proving upper bounds for MAX-2-SAT / A. Kojevnikov, A. S. Kulikov // Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. — SODA '06. — 2006. — P. 11–17.

56. Kulikov, A. New Bounds for MAX-SAT by Clause Learning / A. Kulikov, K. Kutzkov // Computer Science Theory and Applications. — Springer, 2007. — Vol. 4649 of LNCS. — P. 194–204.

57. Kulikov, A. New upper bounds for the problem of maximal satisfiability / A. Kulikov, K. Kutzkov // Discrete Mathematics and Applications. — 2009. — Vol. 19. — P. 155–172.

58. Kutzkov, K. An exact exponential time algorithm for counting bipartite cliques / K. Kutzkov // Information Processing Letters. — 2012. — Vol. 112, № 13. — P. 535 – 539.

59. Lenstra, J. K. Complexity of vehicle routing and scheduling problems / J. K. Lenstra, A. H. G. R. Kan // Networks. — 1981. — Vol. 11, № 2. — P. 221–227.

60. Lieberherr, K. J. Complexity of Partial Satisfaction / K. J. Lieberherr, E. Specker // J. ACM. — 1981. — Vol. 28. — P. 411–421.

61. Lipton, R. Applications of a planar separator theorem / R. Lipton, R. Tarjan // SIAM Journal on Computing. — 1980. — Vol. 9, № 3. — P. 615–627.

62. Lokshtanov, D. Saving space by algebraization / D. Lokshtanov, J. Nederlof // Proceedings of the 42nd ACM symposium on Theory of computing. — STOC '10. — ACM, 2010. — P. 321–330.

63. Mahajan, M. Parameterizing above guaranteed values: MaxSat and MaxCut / M. Mahajan, V. Raman // J. Algorithms. — 1999. — Vol. 31, № 2. — P. 335–354.

64. Mitchell, J. S. B. Guillotine Subdivisions Approximate Polygonal Subdivisions: A Simple Polynomial-Time Approximation Scheme for Geometric TSP, k-MST, and Related Problems / J. S. B. Mitchell // SIAM J. Comput. — 1999. — Vol. 28. — P. 1298–1309.

65. Mömke, T. Approximating Graphic TSP by Matchings / T. Mömke, O. Svensson // Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. — FOCS'11. — Washington, DC, USA: IEEE Computer Society, 2011. — P. 560–569.

66. Moser, R. A. A full derandomization of Schöning's k-SAT algorithm / R. A. Moser, D. Scheder // Proceedings of the 43rd annual ACM symposium on Theory of computing. — STOC '11. — ACM, 2011. — P. 245–252.

67. Mucha, M. Improved Analysis for Graphic TSP Approximation via Matchings / M. Mucha // CoRR. — 2011. — Vol. abs/1108.1130.

68. Mucha, M. Lyndon Words and Short Superstrings / M. Mucha // Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms. — SODA'13. — Society for Industrial and Applied Mathematics, 2013. — To appear.

69. Niedermeier, R. New Upper Bounds for MaxSat / R. Niedermeier, P. Rossmanith // Automata, Languages and Programming. — Springer, 1999. — Vol. 1644 of LNCS. — P. 705–705.

70. Paluch, K. Simpler Approximation of the Maximum Asymmetric Traveling Salesman Problem / K. Paluch, K. Elbassioni, A. van Zuylen // STACS '12. — Vol. 14 of LIPIcs. — 2012. — P. 501–506.

71. Papadimitriou, C. On the Approximability of the Traveling Salesman Problem / C. Papadimitriou, S. Vempala // Combinatorica. — 2006. — Vol. 26. — P. 101–120.

72. Papadimitriou, C. H. The Traveling Salesman Problem with Distances One and Two / C. H. Papadimitriou, M. Yannakakis // Math. Oper. Res. — 1993. — Vol. 18. — P. 1–11.

73. Pevzner, P. A. An Eulerian path approach to DNA fragment assembly / P. A. Pevzner, H. Tang, M. S. Waterman // Proc. Natl. Acad. Sci. — 2001. — Vol. 98, № 17. — P. 9748–9753.

74. Rosenkrantz, D. J. An Analysis of Several Heuristics for the Traveling Salesman Problem / D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis // SIAM J. Comput. — 1977. — Vol. 6, № 3. — P. 563–581.

75. Sahni, S. P-Complete Approximation Problems / S. Sahni, T. Gonzalez // J. ACM. — 1976. — Vol. 23. — P. 555–565.

76. Schöning, U. A probabilistic algorithm for k-sat based on limited local search and restart / U. Schöning // Algorithmica. — 2002. — Vol. 32, № 4. — P. 615–623.

77. Scott, A. D. Linear-programming design and analysis of fast algorithms for Max 2-CSP / A. D. Scott, G. B. Sorkin // Discrete Optimization. — 2007. — Vol. 4, № 3-4. — P. 260 – 287.

78. Sweedyk, Z. $2\frac{1}{2}$-Approximation Algorithm for Shortest Superstring / Z. Sweedyk // SIAM J. Comput. — 1999. — Vol. 29, № 3. — P. 954–986.

79. The Travelling Salesman Problem in Bounded Degree Graphs / A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto // Automata, Languages and Programming. — Springer Berlin / Heidelberg, 2008. — Vol. 5125 of LNCS. — P. 198–209.

80. Trimmed moebius inversion and graphs of bounded degree / A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto // Theory of Computing Systems. — 2010. — Vol. 47, № 3. — P. 637–654.

81. van Rooij, J. M. Exact algorithms for dominating set / J. M. van Rooij, H. L. Bodlaender // Discrete Applied Mathematics. — 2011. — Vol. 159, № 17. — P. 2147–2164.

82. Vassilevska, V. Explicit Inapproximability Bounds for the Shortest Superstring Problem / V. Vassilevska // Mathematical Foundations of Computer Science 2005. — Springer Berlin / Heidelberg, 2005. — Vol. 3618 of LNCS. — P. 793–800.

83. Vazirani, V. V. Approximation Algorithms / V. V. Vazirani. — Springer, 2003.

84. Williams, R. A new algorithm for optimal 2-constraint satisfaction and its implications / R. Williams // Theoretical Computer Science. — 2005. — Vol. 348, № 2-3. — P. 357–365.

85. Williams, V. V. Multiplying matrices faster than coppersmith-winograd / V. V. Williams // Proceedings of the 44th symposium on Theory of Computing / ACM. — 2012. — P. 887–898.

86. Woeginger, G. J. Open Problems Around Exact Algorithms / G. J. Woeginger // Discrete Appl. Math. — 2008. — Vol. 156. — P. 397–405.

87. Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT / J. Gramm, E. A. Hirsch, R. Niedermeier, P. Rossmanith // Discrete Applied Mathematics. — 2003. — Vol. 130, № 2. — P. 139 – 155.

88. Yannakakis, M. On the approximation of maximum satisfiability / M. Yannakakis // SODA'92. — 1992. — P. 1–9.

Список публикаций соискателя

89. Golovnev, A. Approximating asymmetric TSP in exponential time / A. Golovnev // International Journal of Foundations of Computer Science. — 2014. — Vol. 25, № 01. — P. 89–99.

90. Golovnev, A. Solving SCS for bounded length strings in fewer than 2^n steps / A. Golovnev, A. Kulikov, I. Mihajlin // Information Processing Letters. — 2014. — Vol. 114, № 08. — P. 421–425.

91. Golovnev, A. New exact algorithms for the 2-constraint satisfaction problem / A. Golovnev, K. Kutzkov // Theoretical Computer Science. — 2014. — Vol. 526. — P. 18–27.

92. Курбацкий, А. Верхние оценки для задач MAX-2-SAT и MAX-2-CSP относительно средней степени переменных / А. Курбацкий, А. Головнёв // Вестник БГУ. — 2013. — Сер. 1, № 3. — С. 103–107.

93. Golovnev, A. New upper bounds for MAX-2-SAT and MAX-2-CSP w.r.t. the average variable degree / A. Golovnev // Parameterized and Exact Computation: Proceedings of International Conference, Saarbrucken, 2011 / Ed. by D. Marx, P. Rossmanith. — Springer Berlin Heidelberg, 2012. — Vol. 7112 of *Lecture Notes in Computer Science*. — P. 106–117.

94. Bliznets, I. A new algorithm for parameterized MAX-SAT / I. Bliznets, A. Golovnev // Parameterized and Exact Computation: Proceedings of International Conference, Ljubljana, 2012 / Ed. by D. M. Thilikos, G. J. Woeginger. — Springer Berlin Heidelberg, 2012. — Vol. 7535 of *Lecture Notes in Computer Science*. — P. 37–48.

95. Golovnev, A. Approximating shortest superstring problem using de Bruijn graphs / A. Golovnev, A. Kulikov, I. Mihajlin // Combinatorial Pattern Matching: Proceedings of International Conference, Bad Herrenalb, 2013 / Ed. by J. Fischer, P. Sanders. — Springer Berlin Heidelberg, 2013. — Vol. 7922 of *Lecture Notes in Computer Science*. — P. 120–129.

96. Golovnev, A. Solving 3-superstring in $3^{n/3}$ time / A. Golovnev, A. Kulikov, I. Mihajlin // Mathematical Foundations of Computer Science 2013: Proceedings of International Conference, Vienna, 2013 / Ed. by K. Chatterjee, J. Sgall. — Springer Berlin Heidelberg, 2013. — Vol. 8087 of *Lecture Notes in Computer Science*. — P. 480–491.

97. Головнёв, А. Новая верхняя оценка для задачи MAX-2-CSP / А. Головнёв // Международный конгресс по информатике: информационные системы и технологии, БГУ, Минск, 2011 / С.В. Абламейко (отв. ред.). — Минск БГУ, 2011. — с. 270–274.