МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра Систем Информатики

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Образовательная программа 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА.

АНАЛИЗ ДАННЫХ
(направленность)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕШЕНИЯ ЗАДАЧ АНАЛИЗА СЕТЕВЫХ СТРУКТУР

(название темы прописными буквами)

утверждена распоряжением проректора по учебной работе № 326 от «12» октября 2018 г.

Смирнов Павел Владимирович, группа 18227 _____

(Фамилия, Имя, Отчество студента, группа) (подпись студента)

«К защите допущена»

Заведующий кафедрой,

д.ф.-м.н., профессор

Лаврентьев М. М./……..

(ФИО) / (подпись)

«31» мая 2020 г.

Руководитель ВКР

Dr.rer.nat, заведующий

лабораторией алгоритмики ММФ НГУ

ван Беверн Р. А./……...

(ФИО) / (подпись)

«31» мая 2020 г.

Дата защиты: «9» июля 2020 г.

Новосибирск, 2020г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра Систем Информатики
(название кафедры)

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА
Направленность: АНАЛИЗ ДАННЫХ

УТВЕРЖДАЮ

Зав. кафедрой д.ф.-м.н., профессор
Лаврентьев М. М.
(фамилия, И., О.)
……..
(подпись)
«12» октября 2018 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

Студенту Смирнову Павлу Владимировичу, группы 18227
(фамилия, имя, отчество, номер группы)

Тема: Разработка алгоритмов и программного обеспечения для решения задач анализа сетевых структур
(полное название темы выпускной квалификационной работы магистра)

утверждена распоряжением проректора по учебной работе от 12.10.2018 № 326

Срок сдачи студентом готовой работы 31 мая 2020 г.

Цель работы: Исследовать задачу об установлении связности в беспроводной сети на предмет наличия эффективного решения при малом числе связных компонент

Структурные части работы: Улучшение существующего алгоритма решения, реализация его и алгоритмов-конкурентов, экспериментальный анализ эффективности нового алгоритма, поиск эффективной редукции данных для задачи.

Руководитель ВКР
Dr.rer.nat, заведующий лабораторией
алгоритмики ММФ НГУ
ван Беверн Р. А./……..
(ФИО) / (подпись)
«12» октября 2018 г.

Задание принял к исполнению
по распоряжению № 326
Смирнов П. В./……..
(ФИО студента) / (подпись)
«12» октября 2018 г.

СОДЕРЖАНИЕ

1 Введение

Данная работа посвящена решению задачи Min-Power Symmetric Connectivity (далее просто MinPSC). Задача возникает в контексте беспроводных сенсорных сетей. Между узлами сети — беспроводными точками связи — можно устанавливать двустороннюю связь, если мощности каждой из точек достаточно, чтобы покрыть расстояние между ними. Для установления полной связности, то есть возможности передать сообщение от каждой точки до любой другой, потребуется выдать каждой точке некоторую мощность. Естественна задача минимизации суммарной выданной мощности, то есть минимизации энергозатрат. MinPSC представляет собой математическую формулировку этой задачи и является NP-трудной.

В рамках данной работы будет рассматриваться параметрический подход к решению задачи MinPSC. Теория параметрической сложности — это молодое направление в теории сложности вычислений. Идея этого направления — поиск малых параметров во входах задач, которые можно эксплуатировать для решения задачи быстрее, чем в общем случае [4]. Для входа размера n с параметром равным k параметризованные алгоритмы решают задачу за время $f(k)\,\mathrm{poly}(n)$. Заметим, что для частного случая NP-трудной задачи, который отличается от общего случая ограничением на параметр k, задача решается за полиномиальное время. Однако, это было бы верно и для алгоритма с трудоёмкостью $n^{O(k)}$, но такой алгоритм неэффективен уже при малых значениях k. Параметризованные же алгоритмы лучше, так как при изменении параметра трудоёмкость меняется на постоянный множитель. Помимо практического интереса теория параметрической сложности имеет значимость для теории сложности в целом: находя эксплуатируемые параметры, мы находим причины, почему та или иная задача NP-трудна. Имея параметризованный алгоритм решения NP-трудной задачи с трудоёмкостью $f(k)\,\mathrm{poly}(k)$, можно сказать, что задача трудна из-за параметра k, ведь экспоненциальная часть трудоёмкости $f(k)$ зависит только от параметра. Мы рассмотрим решение NP-трудной задачи MinPSC параметризованным алгоритмом, эксплуатирующим один параметр. Проверим, может ли он соревноваться с экспоненциальными алгоритмами, явно параметр не использующими.

Одним из подходов к решению задачи, используя малые значения параметра, является редукция данных [7]. Это уменьшение размера входа задачи, при котором не теряется оптимальное решение. Мы проверим, существует ли редукция данных, сжимающая вход задачи до размера полиномиального от значения параметра. Такие редукции несут большую ценность, так как для NP-трудных задач решение задачи на сжатом входе — всё ещё NP-трудная задача: любое уменьшение размера входа значительно сокращает время решения.

1.1 Постановка задачи

Введём задачу MinPSC, решению которой посвящена данная работа. (В постановке задачи присутствуют термины из теории графов, определения которых можно найти в разделе 2.)

Задача 1.1 (Min-Power Symmetric Connectivity)**.**
Вход: Пара (G, w) из графа $G = (V, E)$ и функции весов $w : E \to \mathbb{N}$.
Требуется: Найти функцию $p : V \to \mathbb{N}$, минимизирующую стоимость $\sum_{v \in V} p(v)$ при условии связности графа $G_p = (V, E_p)$, где E_p — множество всех рёбер $\{u, v\} \in E$ таких, что

$$\min\{p(u), p(v)\} \geq w(\{u, v\}). \tag{1}$$

Допустимым решением *будем называть любое отображение $p : V \to \mathbb{N}$, при котором граф G_p связен.* Оптимальное решение — *допустимое решение с минимальной стоимостью.*

Иногда мы будем называть решением само отображение p, а иногда граф G_p, так как понятно, какому минимальному по стоимости решению p он соответствует.

Метрическим случаем задачи *будем называть частный случай задачи, в котором G является полным графом и w удовлетворяет неравенству треугольника*

$$\forall u, v, t \in V : w(\{u, v\}) \leq w(\{u, t\}) + w(\{t, v\}).$$

Смысл введённых величин в контексте беспроводных сетей следующий. Множество вершин V — это множество беспроводных точек связи, или сенсоров. Рёбра E — возможные соединения. Обычно граф G полный, и все соединения возможны. Каждой точке u можно предоставить некоторую потребляемую мощность $p(u)$. Две точки u и v могут установить связь, если выполняется условие (1). Мы будем называть это условие *условием покрытия* для ребра $\{u, v\}$. Наконец, G_p — граф, рёбра которого представляют собой пары связавшихся точек. Цель при этом — минимизируя суммарную мощность, достичь связности всего графа.

Физический смысл весов и условия покрытия следующий. Мощность $p(u)$ точки u влияет на дальность $a(u)$, на которую точка может передавать сообщения. В реальности имеет место зависимость вида $a(u) = (p(u))^{\frac{1}{\gamma}}$ для некоторого $2 \leq \gamma \leq 4$ [14]. Для установления связи две точки u и v на расстоянии $r(u, v)$ должны быть способны обмениваться сообщениями, а значит необходимо $a(u) \geq r(u, v)$ и $a(v) \geq r(u, v)$. Возведя неравенства в степень γ, получим $\min\{p(u), p(v)\} \geq r(u, v)^\gamma$. Поэтому при выборе весов $w(\{u, v\}) \coloneqq r(u, v)^\gamma$ условие покрытия становится физически корректным. Заметим, что w является метрикой лишь при $\gamma = 1$. В дальнейших рассуждениях мы не будем ограничивать вид функции w, за исключением раздела 4.2, в которой мы будем считать w метрикой.

Также мы будем рассматривать соответствующую задаче MinPSC задачу распознавания. (Для формального определения терминов из параметрической теории сложности см. раздел 2.)

Задача 1.2 (MinPSC(t))**.**

Вход: Пара (G, w): граф $G = (V, E)$, веса $w : E \to \mathbb{N}$ и число $t \in \mathbb{N}$.

Требуется: Определить, существует ли функция $p : V \to \mathbb{N}$ такая, что $\sum_{v \in V} p(v) \leq t$ при условии связности графа $G_p = (V, E_p)$, где E_p — множество всех рёбер $\{u, v\} \in E$ таких, что:

$$p(u), p(v) \geq w(\{u, v\}).$$

Задача отличается от MinPSC тем, что является не оптимизационной задачей, а задачей распознавания. В ней не требуется минимизировать $\sum_{v \in V} p(v)$, но нужно определить, существует ли решение стоимости не больше t. Параметризация для такой задачи аналогична параметризации для задачи MinPSC.

В теории сложности результаты труднорешаемости обычно доказываются для задач распознавания, а затем результаты распространяются на задачи оптимизации [8]. В частности, для задач распознавания существует понятие кернелизации. Мы будем пользоваться именно формулировкой задачи распознавания в разделе 4.1, когда будем проверять у задачи наличие ядра полиномиального размера.

Заметим сразу, что задачи MinPSC и MinPSC(t) полиномиально сводимы друг к другу по Тьюрингу. Действительно, решив задачу MinPSC для входа задачи MinPSC(t), можно сравнить значение целевого функционала со значением t и дать ответ к задаче MinPSC(t). В обратную сторону сведение тоже нетрудно показать. Пусть есть алгоритм для задачи MinPSC(t), решающий задачу за время $f(|X|)$ на входе X. Рассмотрим вход $Y = (G, w)$ задачи MinPSC, где $G = (V, E)$. Найдём оценку сверху M на целевой функционал. Подойдёт значение M, равное $|V| \cdot \max_{e \in E} w(e)$. Действительно, каждой точке достаточно выдать мощность, равную весу максимального ребра: это позволит ей покрыть любое ребро. (Лучшая оценка будет приведена позже в разделе 1.3). Заметим сразу, что M является $2^{O(|Y|)}$. Теперь двоичным поиском на отрезке $[0, M]$ найдём минимальную суммарную мощность точек. Для этого необходимо проверять возможность достичь значения функционала, равного произвольному числу t на отрезке $[0, M]$. Это можно сделать, решив задачу MinPSC(t) для входа Y за время $f(|Y|)$. Получили алгоритм с трудоёмкостью $f(|Y|) \cdot O(\log |M|)$, что является $f(|Y|) \cdot O(|Y|)$. Получили полиномиальное сведение. Факт полиномиальной сводимости говорит о том, что задачи в некотором смысле “одинаково трудны”.

1.2 Параметризация задачи

Введённая задача MinPSC является NP-трудной [3; 10]. Поэтому для того, чтобы эффективно её решать, нам нужен параметризованный алгоритм, а значит в первую очередь нужен подходящий параметр. Параметр должен позволять решать задачу с такой трудоёмкостью,

чтобы экпоненциальная её часть зависела только от этого параметра, а также параметр должен быть способным принимать достаточно малые значения.

Параметр. Параметр, рассматриваемый нами для задачи MinPSC, вводится следующим образом [1]: иногда в задаче часть решения уже видна, и не нужно тратить экспоненциальное время, чтобы эту часть решения искать. Для обозначения этого частичного решения была введена следующая сущность:

Определение 1.3. Нижние границы вершин — *это отображение* $l : V \to \mathbb{N}$ *такое, что для любого оптимального решения* $p : V \to \mathbb{N}$ *и вершины* $v \in V$ *выполняется* $p(v) \geq l(v)$.

Предположим, что мы уже имеем некоторые нижние границы вершин. (О том, как их искать, рассказано дальше.) Используя эти границы, можно доказать для некоторых рёбер, что они обязательно войдут в оптимальное решение.

Определение 1.4. *Ребро* $\{u, v\}$ *входного графа* G *называется* обязательным ребром*, если* $l(u) \geq w(\{u, v\})$ *и* $l(v) \geq w(\{u, v\})$.

Подграф G_l *графа* G*, содержащий только все обязательные рёбра, назовём* обязательным подграфом*. Компоненты связности обязательного подграфа называем* обязательными компонентами *графа* G *и обозначаем* $G_1 = (V_1, E_1), \ldots, G_C = (V_C, E_C)$*, где* C — число обязательных компонент.

Для обязательных рёбер выполняется условие покрытия 1 при любом оптимальном решении. Значит эти рёбра в такое решение войдут. При этом в оптимальное решение войдёт и обязательный подграф.

Исследуемым параметром является число обязательных компонент C.

Вычисление нижних границ вершин. Определение параметра влечёт два закономерных вопроса: откуда берутся нижние границы вершин и почему введённый параметр вообще хорош?

Ответим сначала на первый вопрос: нижние границы вершин нужно находить. Причём находить их нужно как можно бóльшими по значению. Действительно, чем больше значения нижних границ, тем больше рёбер войдут в обязательный подграф, а значит будет меньше обязательных компонент. Малое значение параметра — как раз то, к чему мы стремимся.

На то, каким именно методом искать нижние границы, мы не накладываем никаких ограничений. Приведём однако пример. Для графа $G = (V, E)$ с весами $w : E \to \mathbb{N}$ можно найти нижние границы вершин $l : V \to \mathbb{N}$ следующим образом. Будем искать нижние границы для каждой вершины $u \in V$ отдельно. Для вершины u рассмотрим граф $G[V \backslash \{u\}]$. Пусть граф распался на компоненты связности $G_1^v = (V_1, E_1), \ldots, G_k^v = (V_k, E_k)$. Ясно, что $p(u) \geq \min_{v \in V_i \cap N(u)} w(\{u, v\})$ для любого $i \in \{1, \ldots k\}$, иначе связности в исходной задаче не

добиться: вершина v не может быть связана ни напрямую с компонентой связности G_i^v, ни через другие компоненты связности, ведь те тоже не могут быть связаны с G_i^v напрямую. Тогда можно положить $l(u)$ равным $\max_{i=1}^{k} \min_{v \in V_i \cap N(u)} w(\{u, v\})$.

Заметим, что метод работает лучше, когда в графе есть точки сочленения: для них нижние границы выбираются из нескольких компонент связности, образовавшихся после удаления точки сочленения. Метод можно улучшить так: в [12] предложено удалять рёбра графа, вес которых превышает верхнюю оценку стоимости оптимального решения. В качестве такой оценки авторы статьи используют удвоенную стоимость минимального остовного дерева. После удаления рёбер в графе могут появиться точки сочленения, и метод станет более эффективным. Мы, конечно, могли испортить полезные свойства графа, удалив рёбра, например метричность w. Однако удалённые рёбра при желании можно вернуть. Нижние границы при этом сохранятся.

Перейдём ко второму вопросу, заданному в начале раздела: чем хорош именно этот параметр? Параметры хороши своими малыми размерами, так как это позволяет снизить экспоненциальный множитель трудоёмкости алгоритма решения или уменьшить размер ядра в случае кернелизации по параметру. Введённый параметр может достигать действительно малых значений. Например, если мы расположим беспроводные точки связи в вершинах регулярной сетки, что является энергоэффективным способом размещения [15; 16], то пользуясь одним лишь вторым методом поиска нижних границ вершин, мы получим всего одну компоненту связности в обязательном подграфе, т. е. $C = 1$. Если точки связи начнут выходить из строя, то число компонент может начать увеличиваться. Тогда для восстановления связности выделяемую мощность придётся перераспределить, что создаст пример задачи MinPSC с малым значением C.

1.3 Известные результаты и наши

Известные результаты. Перечислим знания о задаче, важные в контексте данной работы. Известно, что задача MinPSC является NP-трудной [3; 10]. Для решения задачи MinPSC в общем случае существуют формулировки задач целочисленного линейного программирования (ЦЛП) [12]. В работе, в которой они представлены, приводится широкий набор ограничений, позволяющий решать задачу быстрее. Также существуют верхняя и нижняя оценка на величину оптимального решения задачи MinPSC. Оценкой снизу является стоимость минимального остовного дерева для входного графа, оценка сверху — две стоимости минимального остовного дерева [10]. Задача MinPSC является APX-трудной, и поиск $(1 + \epsilon)$-приближённого решения для некоторого $\epsilon > 0$ является NP-трудной задачей [5].

Для задачи MinPSC существует алгоритм решения с трудоёмкостью

$$\ln \frac{1}{\epsilon} \cdot \left(\frac{4e^2}{\sqrt{2\pi}} \right)^C \cdot \frac{1}{\sqrt{C}} \cdot O(9^C |V|^4),$$

где $\epsilon > 0$ — допустимая вероятность ошибки [1]. Алгоритм параметризован по числу обязательных компонент C. Также существует алгоритм, работающий за время $C^{O(C \log C)} \cdot |V|^{O(1)}$ и не допускающий ошибок [1]. Последний алгоритм интересен с теоретической точки зрения, но на практике предпочтение нужно отдавать первому алгоритму. Конечно каждый из алгоритмов способен решать и задачу распознавания MinPSC(t). Для задачи распознавания существует кернелизация (редукция данных с гарантированными оценками результативности [7]), так как существует параметризованный алгоритм [6]. Размер ядра зависит лишь от параметра, но имеет экспоненциальный размер.

Наши результаты. Исправлена ошибка в параметризованном алгоритме из [1]. Этот параметризованный алгоритм ускорен — теперь он быстрее, чем решение ЦЛП формулировок при помощи CPLEX. Ускорение было достигнуто как оптимизациями, улучшающими оценку трудоёмкости, так и редукцией данных, не имеющей доказуемых оценок результативности. Трудоёмкость ускоренного алгоритма получилась равной

$$\ln \frac{1}{\epsilon} \cdot \left(\frac{4e^2}{\sqrt{2\pi}} \right)^C \cdot \frac{1}{\sqrt{C}} \cdot O(9^C |E| + 4^C |V||E| + |V||E| \log |V|),$$

где $\epsilon > 0$ — допустимая вероятность ошибки. Проведён вычислительный эксперимент, показывающий эффективность параметризованного алгоритма, а также эффективность редукции данных. Несмотря на эффективность редукции, доказано, что для задачи распознавания MinPSC(t) не существует ядра полиномиального размера, если не схлопывается полиномиальная иерархия. Однако предложена приближённая редукция данных для метрического случая задачи MinPSC, сохраняющая линейное от параметра число вершин в графе и сводящая вход задачи MinPSC ко входу близкой задачи, решаемой тем же параметризованным алгоритмом из [1].

2 Терминология

В этом разделе мы введём понятия, необходимые для понимания последующих разделов данной работы. Нам понадобятся основные понятия теории графов, а также параметрической теории сложности.

2.1 Основные понятия теории графов

Граф — это пара $G = (V, E)$, состоящая из множества *вершин* V и множества *рёбер* $E \subseteq \{\{u, v\} \mid \{u, v\} \subseteq V\}$. Вершины u и v для ребра $\{u, v\}$ являются *концами*, а вершина u и ребро $\{u, v\}$ называются *инцидентными*. Ребро с совпадающими концами называем *петлёй*. Множество вершин, которые связаны с вершиной u ребром, обозначаем $N(u) \coloneqq \{v \in V \mid \{u, v\} \in E\}$. При этом $|N(u)|$ называется *степенью* вершины u. Заметим, что если в графе есть петля $\{u, u\}$, то $u \in N(u)$. Граф $G = (V, E)$ называем *полным*, если $E = \{\{u, v\} \mid \{u, v\} \subseteq V, u \neq v\}$, т. е. если граф содержит все возможные рёбра, кроме петель. *Взвешенным* называется граф $G = (V, E)$, с функцией весов $w : E \to \mathbb{N}$. Для ребра $\{u, v\} \in E$ значение $w(\{u, v\})$ называется *весом* этого ребра.

Подграфом графа $G = (V, E)$ называется любой такой граф $G' = (V', E')$, что $V' \subseteq V$ и $E' \subseteq E$. Подграфом графа $G = (V, E)$, *порождённым* подмножеством вершин $V' \subseteq V$, называется граф $G' = (V', E')$ такой, у которого множество рёбер $E' = \{\{u, v\} \mid \{u, v\} \in E \wedge \{u, v\} \subseteq V\}$. Иначе говоря порождённый подграф содержит все рёбра графа G, концами которых являются вершины из V'. Подграф графа G, порождённый вершинами V', обозначаем $G[V']$. *Остовным* подграфом графа $G = (V, E)$ называется подграф $G' = (V, E')$ графа G, то есть подграф на всех вершинах графа G.

Путём в графе $G = (V, E)$ называется последовательность вершин $(u_1, \ldots, u_k) \in V^k, k \geq 1$ такая, что для соседних в пути вершин u_i и u_{i+1} в графе есть ребро $\{u_i, u_{i+1}\} \in E$. При этом число k — *длина* пути. Вершина u_1 называется *началом* пути, а вершина u_k — его *концом*. Путь с началом в вершине u и концом в вершине v мы будем также называть путём "между" вершинами u и v. *Циклом* называется путь, состоящий не меньше чем из двух веришн, начало и конец которого совпадают. Если в графе нет ни одного цикла, то граф называется *ацикличным*.

Граф $G = (V, E)$ называется *связным*, если для любой пары вершин $\{u, v\} \subseteq V$ в графе существует путь между вершинами u и v. Максимальный по включению вершин и рёбер связный подграф G' графа G называется *компонентой связности* графа G. *Мостом* называется ребро, при удалении которого из графа число компонент связности возрастает. *Точкой сочленения* называется вершина, при удалении которой из графа вместе со всеми

рёбрами, концами которых она является, число компонент связности в графе возрастает.

Связный ацикличный граф называется *деревом*. Вершина степени 1 в дереве называется *листом*. Известная вычислительная задача — поиск в графе $G = (V, E)$ с весами $w : E \to \mathbb{N}$ остовного подграфа-дерева $T = (V, E')$ с минимальной стоимостью $\sum_{\{u,v\} \in E'} w(\{u, v\})$. Мы будем называть такой подграф *минимальным остовным деревом*. Таких подграфов может быть много, но стоимость у них одна. Поэтому мы будем говорить о стоимости минимального остовного дерева. Задача поиска маинимального остовного дерева и его стоимости решается за полиномиальное время [13].

Деревом Штейнера в графе $G = (V, E)$ с множеством *терминалов* $K \subseteq V$ является подграф-дерево графа G, содержащий все вершины из K. Задача о минимальном дереве Штейнера — это задача поиска в графе G с весами w дерева Штейнера, сумма весов рёбер которого минимальна.

2.2 Параметризация и кернелизация

В теории параметрической сложности мы будем придерживаться терминологии Флюма и Грохе [6].

Определение 2.1. Задачей распознавания *называется множество* $P \subseteq \Sigma^*$, *где* Σ — *алфавит задачи.* Вход *задачи* — $x \in \Sigma^*$. *Длина входа* x, *или его размер, обозначается через* $|x|$. *Решить задачу для входа* x — *значит определить, принадлежит ли* x *множеству* P.

Определение 2.2. Задачей оптимизации *называется тройка* $(sol, cost, goal)$, *где*

- sol — *функция, сопоставляющая входу задачи множество* допустимых решений. *Функция должна быть определена на* Σ^*, *при этом отношение*

$$R = \{(x, y) \mid x \in \Sigma^* \wedge y \in sol(x)\}$$

  должно удовлетворять условию $|y| \leq p(|x|)$ *для фиксированного полинома* p *и всех пар* $(x, y) \in R$. *Также определение принадлежности пары* (x, y) *отношению* R *должно быть решаемо за полиномиальное время.*

- $cost : R \to \mathbb{N}_+$ — *функция, определяющая* стоимость *решения.*

- $goal \in \{\min, \max\}$ *определяет, решается задача* минимизации *(если* $goal = \min$*) или* максимизации *(если* $goal = \max$*).*

Функция opt, определённая на множестве входов задачи, задаётся выражением $opt(x) := goal\{cost(x, y) \mid y \in sol(x)\}$. *Решение* $y \in sol(x)$ *для входа* $x \in \Sigma^*$ *называется* оптимальным, *если* $cost(x, y) = opt(x)$. *Цель задачи — найти оптимальное решение для заданного входа.*

Определение 2.3. Параметризация *для задачи P — это полиномиально вычислимая функция $\kappa : \Sigma^* \to \mathbb{N}$ из входов задачи P в область значений* параметра. *Пара (P, κ) называется* параметризованной задачей.

Параметр может быть ограничением на решение — размер ответа, например. Параметры могут быть простыми функциями от входа. Простой пример такого параметра — максимальная степень вершины в графе. Иногда параметры являются совсем не простыми функциями от входа. Пример такого параметра — древовидная ширина, отображающая сходность графа с деревом в некотором смысле. И хотя вычисление древовидной ширины само по себе является NP-трудной задачей, можно считать параметром приближение древовидной ширины или делать значение древовидной ширины частью входа. Эксплуатация такого параметра позволяет ограничить экспоненциальную часть трудоёмкости параметром для многих задач [6].

Определение 2.4. Параметризованный алгоритм *для параметризованной задачи (P, κ) — это алгоритм, решающий задачу на любом входе $X \in \Sigma^*$ за время не больше $f(\kappa(X)) \cdot p(|X|)$ для некоторой вычислимой функции $f : \mathbb{N} \to \mathbb{N}$ и полинома p.*

Если для NP-трудной задачи есть параметризованный алгоритм относительно некоторой параметризации κ, то смотреть на это можно с двух сторон. Во-первых задачу можно решать эффективно для малых значений параметра: при фиксированном параметре задача решается за полиномиальное время с полиномом заранее известной степени, например n^3 для любого k. Во-вторых, зная о существовании параметризованного алгоритма для некоторого параметра, мы понимаем, в чём заключается сложность задачи: в этом самом параметре, именно от него трудоёмкость зависит экспоненциально.

Определение 2.5. Кернелизация *для параметризованной задачи распознавания (P, κ) — это полиномиальный алгоритм, который преобразует вход x задачи P ко входу x', называемому* ядром, *задачи P такому, что $x \in P \iff x' \in P$, а также $|x'| \leq g(\kappa(x))$ для некоторой вычислимой функции $g : \mathbb{N} \to \mathbb{N}$, называемой* размером ядра.

Кернелизация — это эффективное сжатие входа задачи в “эквивалентный” вход, ограниченный в размере лишь параметром. Под “эквивалентностью” здесь понимается следующее: ответ для исходного входа положительный тогда и только тогда, когда ответ для сжатого входа положительный.

Определение 2.6. Сведение с полиномиальным преобразованием параметра *(ППП) параметризованной задачи распознавания (P, κ) к параметризованной задаче распознавания (P', κ') — это алгоритм, который за полиномиальное время преобразует вход $x \in \Sigma^*$ ко входу $x' \in \Sigma^*$ так, что*

(i) $x \in P \iff x' \in P'$,

(i) $\kappa'(x') \leq p(\kappa(x))$,

где p *— некоторый фиксированный полином. При этом говорим, что* P *сводима с ППП к* P'.

Заметим, что сводимость задач с ППП транзитивна, т. е. если задача (P_1, κ_1) сводима с ППП к задаче (P_2, κ_2), а последняя сводима с ППП к задаче (P_3, κ_3), то задача (P_1, κ_1) сводима с ППП к задаче (P_3, κ_3).

Нам понадобится понятие WK[1]-трудных задач. Такие задачи задачи не имеют полиномиальных ядер, если не схлопывается полиномиальная иерархия [9]. Для определения WK[1]-трудности введём следующую задачу.

Задача 2.7 (об остановке НМТ)**.**
Вход: Недетерминированная машина Тьюринга (НМТ) M и целое число t.
Требуется: Определить, остановится ли машина M через t шагов, если подать ей на вход пустую строку.

Определение 2.8. WK[1] *— это класс параметризованных задач распознавания, сводимых с ППП к задаче 2.7, параметризованной по значению* $t \log |M|$. *Задача является* WK[1]-трудной*, если к ней сводима с ППП любая задача из WK[1]. Задача является* WK[1]-полной*, если она содержится в WK[1] и является WK[1]-трудной.*

Заметим, что по транзитивности сводимости с ППП для показания WK[1]-трудности задачи достаточно показать сведение к ней с ППП хотя бы одной WK[1]-трудной задачи.

3 Параметризованный алгоритм и ускорение

Параметризованный алгоритм для задачи MinPSC с числом обязательных компонент в качестве параметра был представлен в [1]. Он описан в разделе 3.1. Оказалось, что алгоритм содержит ошибку, а также является непрактичным, как показали наши эксперименты. Мы предлагаем более быстрый алгоритм в разделе 3.2. Далее, в разделе 3.3 будут предложены эвристические оптимизации для алгоритма без доказуемых оценок результативности. Наконец, в разделе 3.4 проводим вычислительный эксперимент, потверждающий эффективность алгоритма.

3.1 Описание известного алгоритма

Входом алгоритма является граф $G = (V, E)$ и веса рёбер w. Будем полагать, что мы некоторым образом нашли нижние границы вершин l. Обязательный граф $G_l = (V, E_l)$ при этом состоит из C обязательных компонент, которые мы будем обозначать $G_1 = (V_1, E_1), \ldots, G_C = (V_C, E_C)$.

Поскольку все рёбра графа G_l появились из-за нижних границ, то они точно лежат в решении. Тогда, чтобы сделать граф связным, необходимо и достаточно добавить в граф $C - 1$ рёбро, связывающее обязательные компоненты. Будем такие рёбра называть *ключевыми*. Концов ключевых рёбер в сумме не больше $2C - 2$.

Для удобства введём новый граф G_l^* с весовой функцией w_l^* следующим образом:

$$
\begin{aligned}
&G_l^* = (V, E^*) \\
&E^* = E \cup \{\{u, v\} \subseteq V \mid u \text{ и } v \text{ находятся в одной обязательной компоненте}\} \\
&w_l^* : E^* \to \mathbb{N} \\
&w_l^*(\{u, v\}) = \begin{cases} 0, & \text{если } u \text{ и } v \text{ в одной обязательной компоненте} \\ w(\{u, v\}) & \text{иначе} \end{cases}
\end{aligned}
$$

В этом графе по сравнению с исходным добавились полные подграфы на обязательных компонентах, причём веса рёбер внутри обязательных компонент нулевые. Можно решать задачу на таком графе с обязательством сделать мощности вершин не меньше нижних границ l. Это проще тем, что больше не надо следить за связностью внутри обязательных компонент. Теперь задача сводится к задаче нахождения связного подграфа в G_l^*, который

- содержит не больше $2C - 2$ вершин,
- содержит хотя бы одну вершину из каждой обязательной компоненты графа G,

- минимизирует суммарную мощность, которую надо добавить к нижним границам, чтобы в решение вошёл этот подграф. Эту мощность будем называть “затратами на включение подграфа в решение”.

Для нахождения такого подграфа алгоритм из [1] перебирает раскраски всех вершин в не более $2C-2$ цветов, и затем ищет связный подграф в G_l^*, который содержит вершину каждого цвета ровно по разу, и который минимизирует затраты на его включение в решение. Для фиксированной раскраски алгоритм решает следующую задачу.

Задача 3.1.

Вход: Четвёрка $(G_l^*, w_l^*, \mathrm{col}, l)$: граф $G_l^* = (V, E_l^*)$, веса $w_l^* : E_l^* \to \mathbb{N}$, цвета вершин $\mathrm{col} : V \to \mathbb{N}$, нижние границы вершин $l : V \to \mathbb{N}$. Через S обозначаем множество цветов $\{\mathrm{col}(v) \mid v \in V\}$.

Требуется: Найти связный подграф $T = (W, F)$ графа G_l^* такой, что col отображает W на C биективно, и T минимизирует

$$\sum_{u \in W} \max\{0, \max_{\{u,v\} \in F} w_l^*(\{u,v\}) - l(u)\}.$$

Алгоритм, перебирающий раскраски, работает с заранее заданной вероятностью ошибки ϵ. Доказательство достаточности такого перебора для получения правильного ответа с вероятностью не меньше $1-\epsilon$ содержится в статье [1]. Ниже приведён сам алгоритм.

Алгоритм 1: для решения задачи MinPSC

Вход: Граф $G = (V, E)$, функция весов w, нижние границы вершин $l \colon V \to \mathbb{N}$ и верхняя граница на вероятность ошибки $\varepsilon \in (0, 1)$.

Выход: Решение p^*, оптимальное с вероятностью не меньше $1 - \varepsilon$.

1. $C \leftarrow$ число компонент связности в G_l, обязательном подграфе G.
2. **для всех** $c_1, c_2, \ldots, c_C \in \mathbb{N}^+$, т. ч. $\sum_{i=1}^{C} c_i \le 2C-2$ **выполнять**
3. выбрать попарно непересекающиеся $S_i \subseteq \{1, \ldots, 2C-2\}$, т. ч. $|S_i| = c_i$ для всех $i \in \{1, \ldots, C\}$.
4. **повторять** $\ln \varepsilon / \ln(1 - \prod_{i=1}^{C} c_i! / c_i^{c_i})$ **раз**
5. **для** $i \in \{1, \ldots, C\}$, случайно покрасить вершины G_l^i, используя цвета из S_i; раскраску назовём $\mathrm{col} \colon V \to \mathbb{N}$.
6. Решить задачу 3.1 на входе $(G_l^*, w_l^*, \mathrm{col}, l)$
7. Пусть $T = (W, F)$ — лучшее найденное решение задачи 3.1.
8. **вернуть** $p^* : \forall u \in V, p^*(u) := \max(\{l(u)\} \cup \{w(\{u,v\}) \mid \{u,v\} \in F\})$.

Остаётся решить задачу 3.1. Заметим, что минимизируемая в ней функция — это разница минимальных мощностей, требуемых для связности подграфа T, и нижних границ вершин. Такая задача решается с помощью динамического программирования. Будем считать

значения $D[v, q, S']$ для состояний $[v, q, S']$, где $v, q \in V$ — вершины графа и $S' \subseteq S, |S'| \neq \emptyset$, — множество цветов. Смысл величины $D[v, q, S']$ следующий. Это минимальное приращение мощностей вершин, которое нужно совершить, чтобы все цвета из S' были связаны, и вершина v при этом имела мощность $\max\{l(v), w_l^*(\{v, q\})\}$.

Приведём формулу, вычисление которой позволяет получить ответ. В оригинальной работе [1] допущена ошибка: минимизируется сумма мощностей вершин подграфа T, а не приращений мощностей вершин подграфа T относительно нижних границ. Мы приводим исправленную формулу. Вес ребра $\{u, v\} \notin E_l^*$ в формуле считаем равным ∞. Через $N(v)$ обозначаем множество соседей вершины v в графе G_l^*. (Далее смысл формулы будет объяснён.)

Для $|S'| = 1$:

$$D[v, q, S'] = \begin{cases} \max\{0, w_l^*(\{v, q\}) - l(v)\}, & \text{если } \mathrm{col}(v) \in S' \\ \infty & \text{иначе} \end{cases} \tag{2}$$

Для $v \in V, q \in N(v), S' \subseteq S, |S'| \geq 2$:

$$\begin{gathered} D[v, q, S'] = \min\left(A[v, q, S'] \cup B[v, q, S']\right) \\ A[v, q, S'] = \left\{ \begin{array}{l} D[v, q, S_1] + D[v, q, S_2] - \max\{0, w_l^*(\{v, q\}) - l(v)\} \\ \quad \text{для всех } S_1 \subsetneq S' \text{ и } S_2 \subsetneq S', \\ \quad\quad \text{т. ч. } S_1 \cup S_2 = S' \text{ и } S_1 \cap S_2 = \{\mathrm{col}(v)\} \end{array} \right\} \\ B[v, q, S'] = \left\{ \begin{array}{l} D[u, q', S' \backslash \{\mathrm{col}(v)\}] + \max\{0, w_l^*(\{v, q\}) - l(v)\} \\ \quad \text{для всех } u \in N(v) \text{ и } q' \in N(u), \\ \quad\quad \text{т. ч. } w_l^*(\{u, q'\}) \geq w_l^*(\{u, v\}) \\ \quad\quad \text{и } w_l^*(\{v, q\}) \geq w_l^*(\{u, v\}) \end{array} \right\} \end{gathered} \tag{3}$$

Граф T ищется в виде дерева (в любом подходящем графе T можно найти остовное поддерево и выдать в качестве ответа). Каждое состояние соответствует некоторому дереву T'. Формулы определяют оптимальное построение этого графа, решая каким из трёх способов дерево T' строится из ранее построенных поддеревьев с меньшим числом цветов. В формуле 2 дерево T' состоит из единственной вершины v. В формуле 3 дерево T' строится одним из двух способов: он является объединением двух поддеревьев, пересекающихся в единственной вершине v (тогда v — точка сочленения в дереве), и тогда стоимость такого построения посчитана в множестве A; либо в дерево T' добавился подграф, состоящий лишь из вершины v (тогда v является листом дерева).

Алгоритм должен вычислить значения D, A и B для всех возможных состояний $[v, q, S']$. Множество $A[v, q, S']$ — это приращения мощностей для различных способов связать два подграфа с наборами цветов S_1 и S_2 в один через вершину v. Приращения $B[v, q, S']$ возникают при попытке присоединить к уже собранному подграфу на цветах $S' \backslash \{\mathrm{col}(v)\}$ новую вершину v,

для чего мы ищем связующее ребро $\{u, v\}$ подходящего веса. Подробное объяснение сведения задачи к данной формулировке выходит за границы этой работы и может быть найдено в [1].

Решив задачу с помощью динамического программирования, мы можем найти минимум $D[v, q, S]$ по $v \in V$ и $q \in V$. Это затраты на включение искомого подграфа в решение. Обозначим состояние, на котором достигается минимум через $[v^*, q^*, S]$.

Для восстановления ответа можно запоминать для каждого состояния $[v, q, S']$ множество *состояний-предшественников* $P[v, q, S']$: если значение $D[v, q, S']$ является минимумом в $A[v, q, S']$, то запомним множества S_1 и S_2 (из формулы 3), для которых достигался минимум в $A[v, q, S']$, и состояниями-предшественниками будут $P[v, q, S'] = \{[v, q, S_1], [v, q, S_2]\}$; иначе, если минимум $D[v, q, S']$ достигался в $B[v, q, S']$, то одноэлементным множеством состояний-предшественников является $P[v, q, S'] = \{[u, q', S' \backslash \mathrm{col}(v)]\}$ для u и q' (из формулы 3), для которых был достигнут минимум в $B[v, q, S']$. Зная P, нетрудно восстановить приращения мощностей. Для этого запустим обход в глубину по состояниям, начиная в состоянии $[v^*, q^*, S]$. Из текущего состояния $[v, q, S']$ при этом обходе можно переходить лишь в множество состояний $P[v, q, S']$. Все состояния, которые мы обойдём таким обходом обозначим через X. Для каждого состояния $[v, q, S'] \in X$ берём в граф T вершину v. Мощность этой вершине нужно выдать равную $\max\{\{l(v)\} \cup \{w(\{v, \tilde{q}\}) \mid [v, \tilde{q}, \tilde{S}] \in X\}\}$.

Утверждение 3.2. *Для задачи 3.1 существует алгоритм с трудоёмкостью* $O(3^{|S|}|V|^2 + 2^{|S|}|V|^4)$.

Доказательство. Приведём алгоритм вычисления значений D и P.

Алгоритм 2: для решения задачи 3.1

Вход: Четвёрка $(G_l^*, w_l^*, \mathrm{col}.l)$: граф $G_l^* = (V, E_l^*)$, функция весов $w_l^* : E_l^* \to \mathbb{N}$, цвета вершин $\mathrm{col} : V \to \mathbb{N}$, нижние границы вершин $l : V \to \mathbb{N}$.

Выход: Значения D из формул 2 и 3, значения P для восстановления ответа.

1. $S \leftarrow \{\mathrm{col}(v) \mid v \in V\}$.
2. $D \leftarrow$ массив $|V| \times |V| \times 2^{|S|}$
3. $P \leftarrow$ массив $|V| \times |V| \times 2^{|S|}$
4. **для** $S' \subset S, |S'| \neq \emptyset$ в порядке неубывания $|S'|$ **выполнять**
5. **для** $v, q \in V$ **выполнять**

6. `// применяем формулу 2`
7. **если** $|S'| = 1$ **тогда**
8. $D[v] \leftarrow \max\{0, w_l^*(\{v, q\}) - l(v)\}$, если $\mathrm{col}(v) \in S'$, иначе ∞
9. **перейти к следующей итерации цикла в строке 5**

10. `// применяем формулу 3`
11. $D[v, q, S'] \leftarrow \infty$
12. $P[v, q, S'] \leftarrow \emptyset$
13. **для** $S_1 \subsetneq S', \{\mathrm{col}(v)\} \in S_1$ **выполнять**
14. $S_2 \leftarrow S' \backslash S_1 \cup \{\mathrm{col}(v)\}$
15. $A \leftarrow D[v, q, S_1] + D[v, q, S_2] - \max\{0, w_l^*(\{v, q\}) - l(v)\}$
16. **если** $A < D[v, q, S']$ **тогда**
17. $D[v, q, S'] \leftarrow A$
18. $P[v, q, S'] \leftarrow \{[v, q, S_1], [v, q, S_2]\}$
19. **для** $u \in N(v), q' \in N(u)$ **выполнять**
20. **если** $w_l^*(\{u, q'\}) \geq w_l^*(\{u, v\})$ и $w_l^*(\{v, q\}) \geq w_l^*(\{u, v\})$ **тогда**
21. $B \leftarrow D[u, q', S' \backslash \{\mathrm{col}(v)\}] + \max\{0, w_l^*(\{v, q\}) - l(v)\}$
22. **если** $B < D[v, q, S']$ **тогда**
23. $D[v, q, S'] \leftarrow B$
24. $P[v, q, S'] \leftarrow \{[u, q', S' \backslash \{\mathrm{col}(v)\}]\}$

25. **вернуть** D, P

Алгоритм напрямую вычисляет формулы для D, запоминая при этом состояния-предшественники P. Оценим трудоёмкость алгоритма. Строки с 7 по 12 будут выполнены $O(2^{|S|}|V|^2)$ раз, так как цикл в строке 4 перебирает $O(2^|S|)$ значений, а цикл в строке 5 — $O(|V|^2)$ значений. Так как цикл в строке 19 перебирает $O(|V|^2)$ значений, то суммарно цикл исполнится $2^{|S|}|V|^4$ раз. Цикл в строке 13 суммарно исполнится $3^{|S|}|V|^2$ раз, так как $O(|V|^2)$

значений перебирает цикл 5, и $3^{|S|}$ — число подмножеств подмножеств множества S — не больше, чем столько значений переберут вместе циклы в строках 4 и 13. Итого $O(3^{|S|}|V|^2 + 2^{|S|}|V|^4)$ времени на выполнение алгоритма.

Посчитаем, сколько времени нам потребуется на то, чтобы целиком восстановить ответ к задаче 3.1. Нахождение состояния $[v^*, q^*, S]$, в котором достигается минимальное приращение мощностей, делается за время, не большее, чем проход по всем состояниям, что асимптотически не больше времени вычисления значений D для этих состояний. Для восстановления мощностей вешин потребуется сделать обход состояний-предшественников, описанный ранее. Времени на это уйдёт также асимптотически не больше, чем на вычисление всех состояний. Тогда это время можно не учитывать в O-нотации итоговой трудоёмкости. □

3.2 Снижение трудоёмкости

При проведении экспериментов оказалось, что алгоритм с трудоёмкостью, как в утверждении 3.2, неконкурентоспособен. В этом разделе приведём более быстрый алгоритм. Оптимизации будет подлежать лишь решение задачи 3.1. Новый алгоритм является усовершенствованным алгоритмом из утверждения 3.2.

Теорема 3.3. *Задача 3.1, возникающая при решении задачи MinPSC, параметризованной по числу необходимых компонент, может быть решена за время* $O(3^{|S|}|E| + 2^{|S|}|V||E| + |E| \log |V|)$.

Замечание. Алгоритму на вход можно подавать исходный граф G с весами w вместо графа G_l^*, в который добавлены рёбра нулевого веса внутри обязательных компонент, и весов w_l^*. Рассуждения в доказательстве теоремы при этом не изменятся. Вход (G, w, col, l) удобнее для алгоритма 1, однако для того, чтобы вход нового алгоритма решения задачи 3.1 был согласован со входом задачи, мы оставим вход $(G_l^*, w_l^*, \mathrm{col}, l)$.

Доказательство. Входом алгоритма является четвёрка $(G_l^*, w_l^*, \mathrm{col}, l)$. Сначала подготовим граф $G_l^* = (V, E_l^*)$: найдём в нём обязательные компоненты (они будут теми же, что и в исходном графе G) и для каждой вершины запомним номер компоненты, в которой она находится за время $O(|V| + |E_l^*|) = O(|V|^2)$ обходом графа в глубину. Теперь удалим из графа все рёбра внутри обязательных компонент, а затем для каждой вершины $u \in V$ добавим в граф ребро $\{u, u\}$. Новый граф назовём $G^\circ = (V, E^\circ)$. Функцию весов w° для него определим следующим образом для любого ребра $\{u, v\} \in E^\circ$:

1. Если u и v находятся в одной обязательной компоненте, в том числе если $u = v$, то $w^\circ(\{u, v\}) = 0$.

2. Если u и v в разных компонентах, то $w^\circ(\{u,v\}) = w_l^*(\{u,v\})$.

Заметим, что w° для любого ребра $\{u,v\} \in E^\circ$ вычисляется за константное время, так как мы посчитали номера обязательных компонент для каждой вершины.

Теперь отсортируем всех соседей $v \in N(u)$ вершины u (с учётом её самой) по неубыванию значения $w^\circ(\{u,v\})$ за время $O(|E|\log|V|)$. Отсортированный список соседей вершины u будем обозначать $N_{\text{sorted}}(u)$. Чтобы в дальнейшем быстро считать значения таблицы динамического программирования, посчитаем для каждого ребра $\{v,q\} \in E^\circ$ и вершин $u \in N(v)$, т. ч. $w^\circ(\{v,q\}) \geq w^\circ(\{u,v\})$, следующее:

$$X[v,q,u] \coloneqq \min\{j \in \{1,\dots,|N(u)|\} \mid w^\circ(\{u,q_j\}) \geq w^\circ(\{u,v\})\},$$

где $N_{\text{sorted}}(u) = \{q_1,\dots,q_{|N(u)|}\}$. Для вершины u значение $X[v,q,u]$ позволяет найти суффикс массива $N_{\text{sorted}}(u)$, каждая вершина q' на котором удвлетворяет условию из строки 20 алгоритма 2. Вычисление всех значений $X[v,q,u]$ для $\{v,q\} \in E^\circ$ и $u \in N(v)$ займёт $O(|V||E|\log|V|)$ времени, если использовать двоичный поиск по $N_{\text{sorted}}(u)$ для вычисления минимума.

Формула динамического программирования. Новый алгоритм, как и старый, будет вычислять таблицу значений $D[v,q,S']$. При этом формула вычислений начальных значений 2 остаётся той же с точностью до функции весов:

$$D[v,q,S'] = \begin{cases} \max\{0, w^\circ(\{v,q\}) - l(v)\}, & \text{если } \mathrm{col}(v) \in S' \\ \infty & \text{иначе} \end{cases} \tag{4}$$

Формула же 3 слегка изменится (отличия далее будут объяснены):

$$\begin{aligned}
&D[v,q,S'] = \min\left(A[v,q,S'] \cup B[v,q,S']\right) \\
&A[v,q,S'] = \left\{ \begin{array}{c} D[v,q,S_1] + D[v,q,S_2] - \max\{0, w^\circ(\{v,q\}) - l(v)\} \\ \text{для всех } S_1 \subsetneq S' \text{ и } S_2 \subsetneq S', \\ \text{т. ч. } S_1 \cup S_2 = S' \text{ и } S_1 \cap S_2 = \{\mathrm{col}(v)\} \end{array} \right\} \\
&B[v,q,S'] = \begin{cases} B_1[v,q,S'], & \text{если в } S' \backslash \mathrm{col}(v) \text{ есть цвета из обязательной,} \\ & \text{компоненты, в которой лежит вершина } v \\ B_2[v,q,S'], & \text{иначе} \end{cases} \\
&B_1[v,q,S'] = \{D[u,q',S'\backslash\{\mathrm{col}(v)\}] \mid \{u,q'\} \in E^\circ\} + \max\{0, w^\circ(v,q) - l(v)\} \\
&B_2[v,q,S'] = \left\{ \begin{array}{c} D[u,q',S'\backslash\{\mathrm{col}(v)\}] + \max\{0, w^\circ(\{v,q\}) - l(v)\} \\ \text{для всех } u \in N(v) \text{ и } q' \in N(u), \\ \text{т. ч. } w^\circ(\{u,q'\}) \geq w^\circ(\{u,v\}) \\ \text{и } w^\circ(\{v,q\}) \geq w^\circ(\{u,v\}) \end{array} \right\}
\end{aligned} \tag{5}$$

Множество $B[v, q, S']$ "распалось" на $B_1[v, q, S']$ и $B_2[v, q, S']$. При этом множество $B_2[v, q, S']$ имеет тот же смысл, что и множество $B[v, q, S']$ в формуле 3: это варианты присоединить вершину v с весом $w^\circ(\{v, q\})$ к искомому в задаче 3.1 подграфу, соединив её ребром с вершиной u. Но мы удалили из графа рёбра веса 0 внутри обязательных компонент, что делает невозможным некоторые варианты присоединения вершины v к подграфу. Чтобы учесть такие варианты, в формулу добавлено множество $B_1[v, q, S']$: варианты присоединения вершины v к подграфу через рёбра веса 0 внутри обязательных компонент.

Заметим, что в старом алгоритм для каждой вершины $v \in V$ вершина q перебиралась во множестве V. В новом алгоритме q перебирается в $N(v)$, т. е. перебирается ребро $\{v, q\} \in E$: ещё в формуле 3 веса для пар вершин, не являющихся рёбрами, равны ∞, и для таких рёбер значение D равно ∞, что бесполезно.

Вычисление таблицы динамического программирования. Теперь вычислим значения $D[v, q, S']$ для $\{v, q\} \in E$ и $S' \subseteq S, |S'| \geq 1$. Будем вычислять значения $D[v, q, S']$ в порядке неубывания $|S'|$. Вычислив значения $D[v, q, S']$ для всех $\{v, q\} \in E$ и фиксированного S', сразу посчитаем минимальное приращение мощностей, которым можно связать множество цветов S':

$$Y[S'] \coloneqq \min\{D[v, q, S'] \mid \{v, q\} \in E\}.$$

Трудоёмкость при этом изменится в константное число раз. Также, вычислив значения $D[v, q, S']$ для фиксированных S' и v и всех $q \in N(v)$, вычислим для всех $j \in \{1, \ldots, |N(v)|\}$

$$Z[v, j, S'] \coloneqq \min\{D[v, q_i, S' \backslash \{\mathrm{col}(v)\}] \mid i \in \{1, \ldots, j\}\},$$

где $N_{\mathrm{sorted}}(v) = \{q_1, \ldots, q_{|N(v)|}\}$. Значение $Z[v, j, S']$ вычисляется за константное время через значения $Z[v, j-1, S']$ и $D[v, q_j, S' \backslash \{\mathrm{col}(v)\}]$, а значит итоговое время исполнения изменится на константный множитель.

Теперь покажем, как вычислять значения $D[v, q, S']$. Каждое из значений $D[v, q, S']$ для $\{v, q\} \in E$ и $|S'| = 1$ может быть вычислено по формуле 4 за константное время. Для $S' \geq 2$ мы вычисляем $D[v, q, S']$, считая, что значения D для $S'' \subsetneq S'$ уже вычислены. Значение $D[v, q, S']$ вычисляем, как $\min\{\min A[v, q, S'], \min B[v, q, S']\}$ по формуле 5. Значение $\min \emptyset$ считаем равным ∞.

Значения $\min A[v, q, S']$ для всех $\{v, q\} \in E$ и $S' \subseteq S$ вычисляются суммарно за время $O(3^{|S|}|E|)$, так как для фиксированной вершины $v \in V$ есть не больше $3^{|S|}$ способов выбрать такие S_1 и S_2, что $S_1 \cup S_2 = S'$ и $S_1 \cap S_2 = \{\mathrm{col}(v)\}$ для всех различных $S' \subseteq S$. Действительно, каждый элемент из $S \backslash \{\mathrm{col}(v)\}$ лежит либо в S_1, либо в S_2, либо в $S \backslash (S_1 \cup S_2)$.

Значение $\min B[v, q, S']$ вычисляется по-разному, в зависимости от того, есть ли в обязательной компоненте вершины v вершины любого цвета из $S' \backslash \{\mathrm{col}(v)\}$. Значение $\min B_1[v, q, S']$ считается за константное время для каждого ребра $\{v, q\} \in E^\circ$ и $S' \subseteq S$, используя $Y[S' \backslash \{v\}]$.

Тогда суммарное время, потраченное на вычисление всех значений $\min B_1[v, q, S']$, равно $O(2^{|S|}|E|)$. Значение $\min B_2[v, q, S']$ считается за время $O(|V|)$ для каждого ребра $\{v, q\} \in E$ и $S' \subseteq S$. Для этого переберём вершины $u \in N(v)$ такие, что $w^\circ(\{v, q\}) \geq w^\circ(\{u, v\})$, и возьмём минимум среди $Z[u, j, S'\backslash\{v\}]$, где $j = X[v, q, u]$. Тогда суммарное время потраченное на вычисление всех значений $\min B[v, q, S']$ равно $O(2^{|S|}|V||E|)$.

Осталось посчитать итоговую трудоёмкость. На подготовку в начале уйдёт $O(|V||E| \log |V|)$ времени, на вычисление значений A — $O(3^{|S|}|E|)$ времени и на вычисление значений B — $O(2^{|S|}|V||E|)$ времени. Восстановление подграфа, искомого в задаче 3.1, производится тем же образом, что и в старом алгоритме — через обход состояний-предшественников. □

3.3 Редукция данных

Улучшений, описанных выше, было недостаточно для того, чтобы обогнать другие алгоритмы по скорости. Для дальнейшего ускорения алгоритма на практике мы реализовали две эвристические оптимизации. Каждая из них по сути редукция данных.

1. Удалим из графа все рёбра, вес которых превышает верхнюю оценку на стоимость оптимального решения, ведь ни одно из них не может входить в оптимальное решение. В некотором виде такая оптимизация была применена в ЦЛП формулировке из работы [12]. В качестве верхней оценки мы, как и авторы упомянутой работы, взяли удвоенную стоимость минимального остовного дерева.

2. Удалим все вершины, которые не имеют инцидентных им рёбер, кроме как внутри их обязательных компонент. Такая оптимизация, конечно, некорректна в общем случае, ведь теряется связность внутри компонент. Но при нашей “внутренней” постановке задачи такое условие не нужно: не зря мы соединили все вершины обязательной компоненты между собой рёбрами нулевого веса. Для связности цветов между собой такие вершины тоже не нужны, а значит их можно спокойно удалить. Значит для задачи 3.1 такая редукция корректна.

Можно сделать замечание, что при входах с полными графами вторая редукция работает только после применения первой. Действительно, если граф полный, то вершин, не имеющих соседей снаружи своей компоненты, не найдётся.

Как будет показано в разделе 3.4, приведённые редукции сильно повлияли на возможность алгоритма конкурировать с решателем ЦЛП постановок задачи.

3.4 Вычислительный эксперимент

Экспериментальное исследование преследует две цели: сравнение скорости работы алгоритма с существующими аналогами и оценка эффективности эвристической редукции данных.

3.4.1 Генерация данных

Тестовые входы для задачи MinPSC генерировались следующим образом. Зафиксируем некоторое $N \in \mathbb{N}$. Рассмотрим квадратную сетку $N \times N$ с единичным шагом на плоскости. (Напомним, что сетки обладают свойством оптимального покрытия плоскости [15; 16].) Узлы сетки — вершины графа, или беспроводные сенсоры. Между любыми двумя сенсорами существует ребро с весом, равным квадрату евклидового расстояния между этими сенсорами. Понятно, как связать такой граф: необходимо и достаточно каждой вершине выдать мощность 1. Теперь из имеющихся N^2 вершин случайно выберем $K \cdot N^2$. Удалим их вместе со смежными им рёбрами. Это соответствует выходу некоторых сенсоров из строя. Для восстановления связности беспроводной сети требуется решить задачу MinPSC для полученного графа. Осталось уточнить, как выбираются вершины. Для этого были рассмотрены три подхода:

- **Тип 1. Независимый выход сенсоров из строя.** Подмножество вершин размера K выбирается равновероятно среди всех таких подмножеств.
- **Тип 2. Проблемы в определённой точке сети (в центре).** Каждая координата очередного вышедшего из строя сенсора выбирается по распределению Гаусса, отцентрованному по центру сетки. Дисперсия равна $\left(\frac{N-1}{6}\right)^2$. Таким образом координата с высокой вероятностью (≈ 0.997) попадёт в сетку. Если координата в сетку не попала, выбираем значение снова, пока не попадёт. Затем координаты округляются, и мы получаем позицию сенсора.
- **Тип 3. Проблемы на границе сети.** Генерируем координаты сенсоров, как в проблемах второго типа (без округления), а затем отражаем каждую точку-позицию сенсора от центра сетки к её углу. То есть, если точка лежит в квадранте сетки $A_1A_2A_3A_4$, при чём A_1 — центр сетки, а A_3 — её угол, То отражаем точку симметрично относительно прямой A_2A_4. Теперь координаты округлим.

После удаления вершин, мы выбираем входы, для которых обязательный подграф имеет 2, 3 или 4 компоненты. Если вход имеет больше компонент, то для такого входа тяжело собрать хорошую статистику: время исполнения велико. Если во входе меньше компонент, то задача становится тривиальной.

Входы были сгенерированы для различных N от 10 до 100 с шагом 10. Значение K выбиралось в диапазоне $(0, 2]$ так, чтобы с хорошей вероятностью генерировать графы с подходящим значением C. Наконец, для каждого набора параметров случайный вход генерировался 10 раз.

3.4.2 Условия экспериментов

Для проведения экспериментов алгоритм 1 был реализован на языке C++ и скомпилирован компилятором GNU C++ 5.4.0 с уровнем оптимизации -O3. Эксперименты проведены на двухъядерном процессоре Intel Core i7-4600U, 15.6 Гб ОЗУ и 64-битной операционной системе Ubuntu 16.04.

В сравнении участвовали пять алгоритмов:

- **DP1**. Алгоритм 1 с двумя эвристиками, описанными в разделе 3.3.
- **DP2**. Алгоритм 1 без эвристик.
- **BF**. Простой алгоритм полного перебора.
- **EX1**, **EX2**. Линейные модели из [12], для решения которых применяется CPLEX.

Для **DP1** и **DP2** вероятность ошибки выбрана равной 0.1. Модели Монтеманни и Гамбарделлы из [12] были реализованы с предложенной ими предварительной обработкой данных и набором ограничений, наиболее эффективно показавшим себя в их исследованиях.

3.4.3 Результаты

На рисунке 1 видно, как сжимается вход после применения редукции данных из раздела 3.3. Видно, что каждая из редукций вносит значительный вклад в сжатие входа.

На рисунке 2 показано время работы алгоритмов для тестов типа 1 разного размера.

Алгоритм полного перебора **BF**, будучи самым быстрым при $C = 2$, показывает свою неэффективность уже при $C = 3$. Для $C = 4$ даже на входе с 340 вершинами работает дольше 90 минут, поэтому на графике не представлен.

DP1 обгоняет модели ЦЛП **EX1** и **EX2** для всех $C \in \{2, 3, 4\}$. Это говорит о том, что использование малых значений C значительно влияет на скорость алгоритма. Можно даже заметить, что преимущество **DP1** растёт с ростом C.

Также заметим, что **EX1** и **EX2** являются постановками задач ЦЛП, решаемых CPLEX с использованием многопоточности. Для наших экспериментов это 8 потоков. Наши реализации **DP1** и **DP2** работают лишь в один поток. Конечно, алгоритм можно легко параллелизовать — например, параллельно обрабатывать разные раскраски (алгоритм 1). Однако в наших

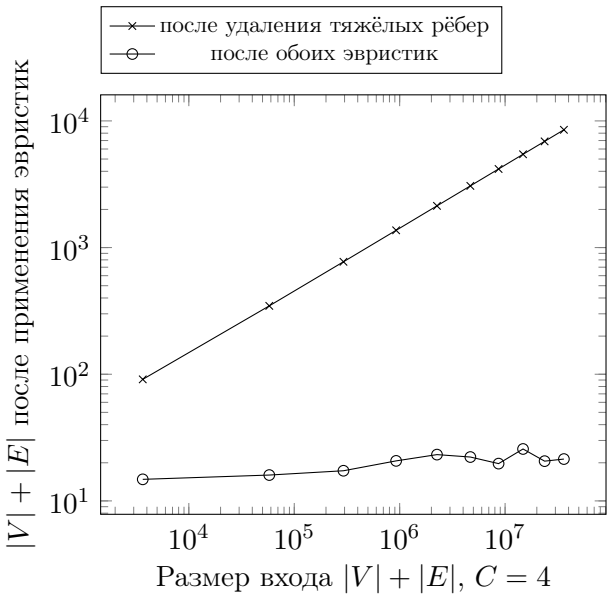


Рисунок 1: Эффект эвристической редукции данных на входах типа 1 с $C = 4$ обязательными компонентами

экспериментах выгода от параллелизации процесса перебора раскрасок незначительна из-за куда более тяжёлой стадии предварительной обработки. (Сравнение стадий предварительной обработки и динамического программирования показано на рисунке 5.)

При заявленной вероятности ошибки **DP1** и **DP2**, равной 0.1, реальная ошибка была значительно ниже. Для **DP1** — одна ошибка на 300 рассмотренных выше тестов, а для **DP2** — одна на 290, так для $C = 4$ при большом числе вершин некоторые тесты не удалось проверить: алгоритм потербляет чрезмерное колчичество памяти. (Ошибки **DP1** и **DP2** были допущены на разных входах.)

Алгоритмы **DP1** и **EX2** показали наилучшие результаты, и мы решили сравнить их более подробно на тестах разного типа. Выбрали C равным 4. Результаты — на рисунке 3. Одна точка на графике представляет один вход. Чем дальше точка от диагонали, тем больше отличаются времена работ алгоритмов. Если точка попала на пунктирную линию, то времена работ отличаются в 1.5 раза, а если на линию, нарисованную точками, то в 5 раз. Точки, стоящие выше диагонали, соответствуют входам, на которых **DP1** оказался быстрее **EX2**. Ниже диагонали — **EX2** оказался быстрее. Точки на графике можно визуально разбить на кластера, так как входы генерировались для нескольких $N \in \{10, 20, \ldots, 100\}$. На всех входах **DP1** быстрее **EX2** хотя бы в 1.53 раза.

На графиках видно, что редукция данных играет большую роль для скорости алгоритмов **DP1** и **DP2**. Для $C = 2$ алгоритм без редукции **DP2** оказался быстрее из-за того, что сама редукция данных занимает слишком много времени. Для $C = 3$ алгоритмы работают

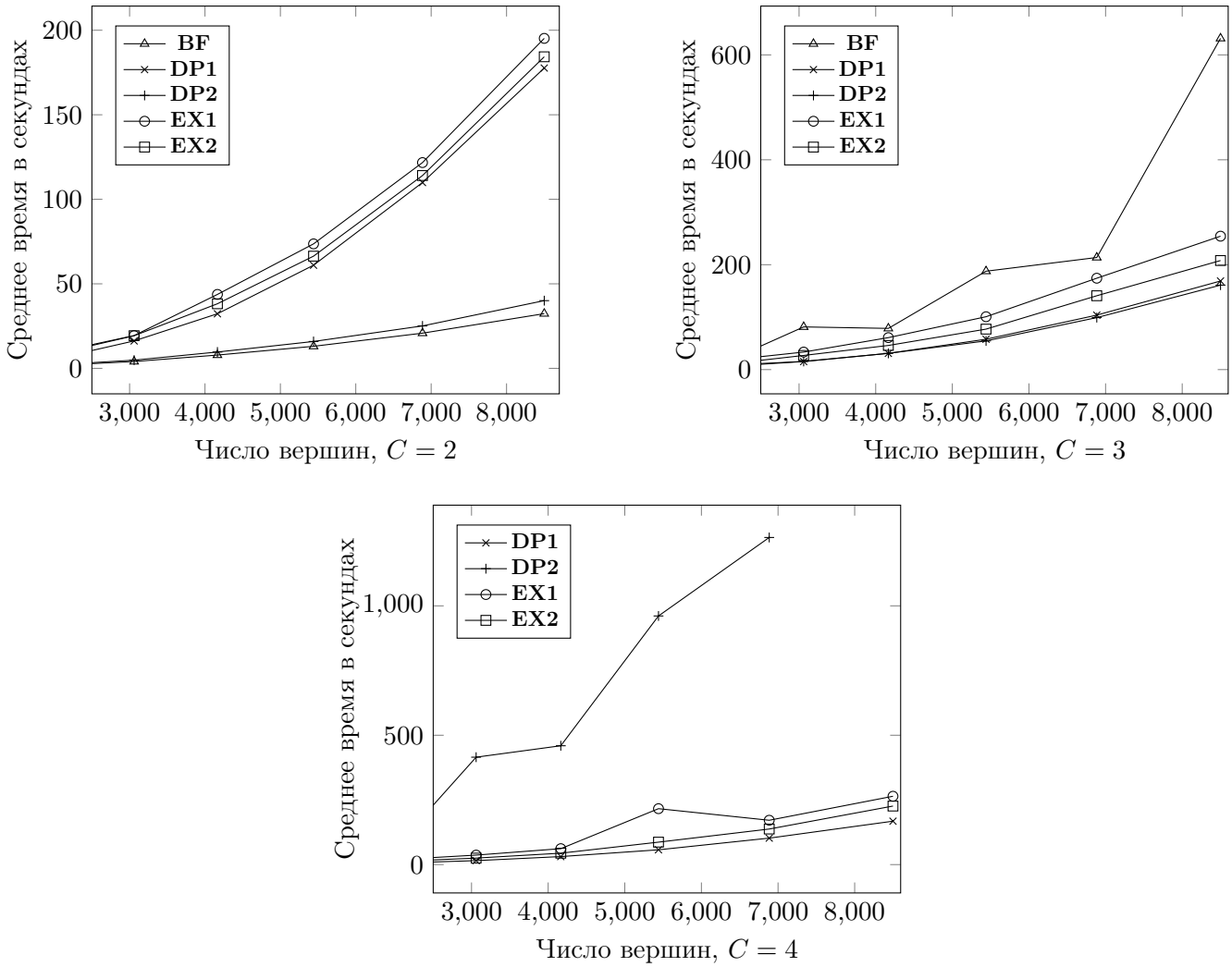


Рисунок 2: Экспериментально зафиксированное время работы алгоритмов **DP1**, **DP2**, **BF**, **EX1** и **EX2** на входах типа 1 с $C \in \{2, 3, 4\}$ обязательными компонентами.

примерно одно время, а значит выигрыш от редукции и время на её исполнение компенсируют друг друга. Для $C = 4$ ускорение, которое даёт редукция, уже значительно, и оно не только ускоряет весь алгоритм, но и позволяет его запускать на больших входах в случае ограниченной памяти. Для $C = 4$ алгоритм без редукции **DP2** не смог отработать на входах с 8500 вершинами из-за того что вышел за границы 15.6 Гб ОЗУ.

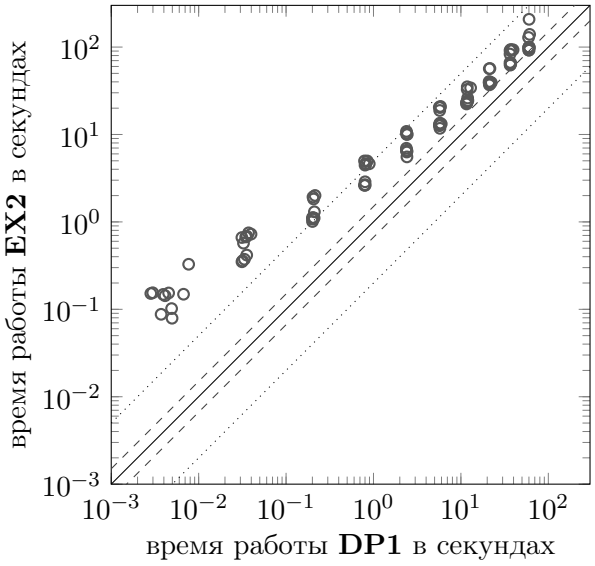
На рисунке 4 показано сравнение **DP1** и **EX2** аналогичным образом, но с другой функцией весов. Теперь вес ребра является кубом (а не квадратом) Евклидового расстояния между инцидентными ему вершинами. На всех таких тестах **DP1** быстрее **EX2** хотя бы в 1.81 раз. Эта серия экспериментов была проведена, чтобы показать, что **DP1** хорошо работает не только на одной функции весов.

Следующий рисунок 5 показывает, какую часть времени алгоритмы тратят на предварительную обработку, а какую — на само решение. Для **DP1** предварительная обработка — это эвристическая редукция данных, описанная в разделе 3.3, а решение — перебор раскрасок с динамическим программированием. Для **EX2** стадия предварительной обработки состоит из нахождения стоимости минимального остовного дерева, удаления дорогих рёбер и составление формулировки задачи ЦЛП.

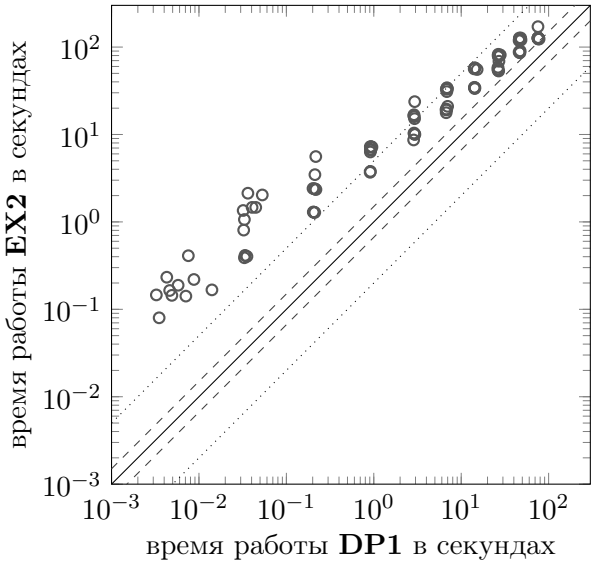
Оказывается, что для малых C время, которое **DP1** тратит на само решение задачи, незначительно по сравнению со временем, которое тратится на редукцию данных. Это объясняет, почему параллелизация процесса перебора раскрасок не смогла ускорить алгоритм. Также, предварительная обработка у обоих алгоритмов занимает приблизительно одно и то же время. Это ожидаемо, так как состоит она в основном из нахождения минимального остовного дерева.

Наконец, на рисунке 6 показано, насколько варьируется время выполнения алгоритмов **DP1** и **EX2**. Для каждого числа вершин алгоритмы запускались на 10 входах. Таким образом, для каждого числа вершин и алгоритма мы получили выборку из 10 результатов. Верхняя и нижняя границы свечки показывают максимальное и минимальное время работы в этих выборках соответственно, а черта между ними — медиану выборки. Графики показывают, что **DP1** работает не только быстрее, но и показывает большую стабильность во времени работы.

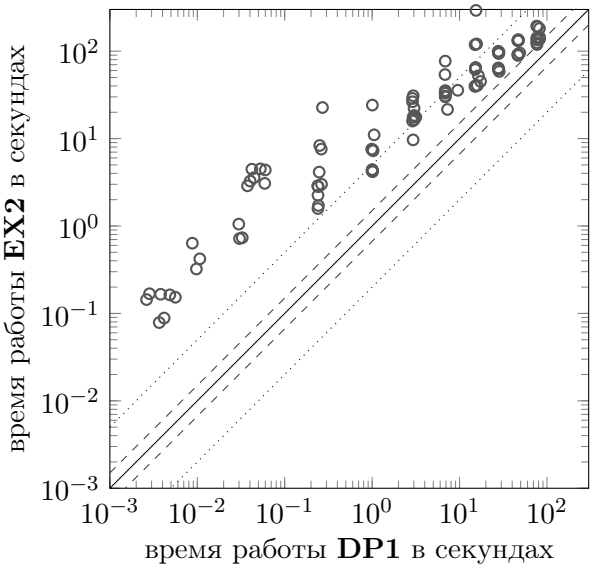
В итоге можно сказать, что для малого числа обязательных компонент лучше всего использовать алгоритм **DP1**.



(a) тесты типа 1



(b) тесты типа 2



(c) тесты типа 3

Рисунок 3: Сравнение **DP1** и **EX2**.

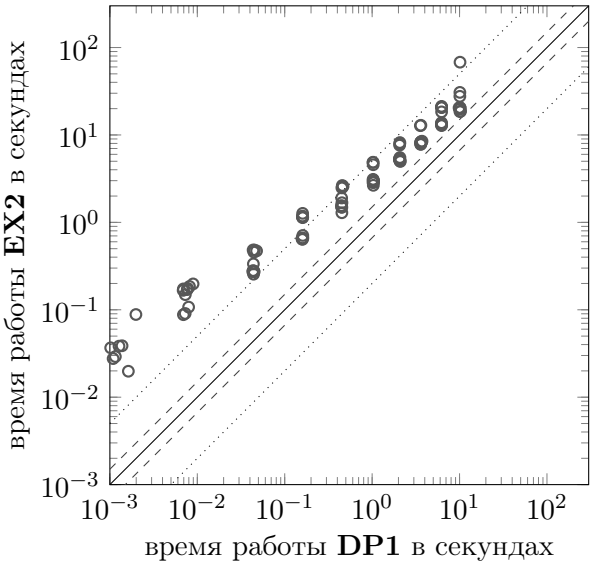


Рисунок 4: Сравнение **DP1** и **EX2**. Тесты типа 1, $w(\{u, v\}) = \|u - v\|^3$.

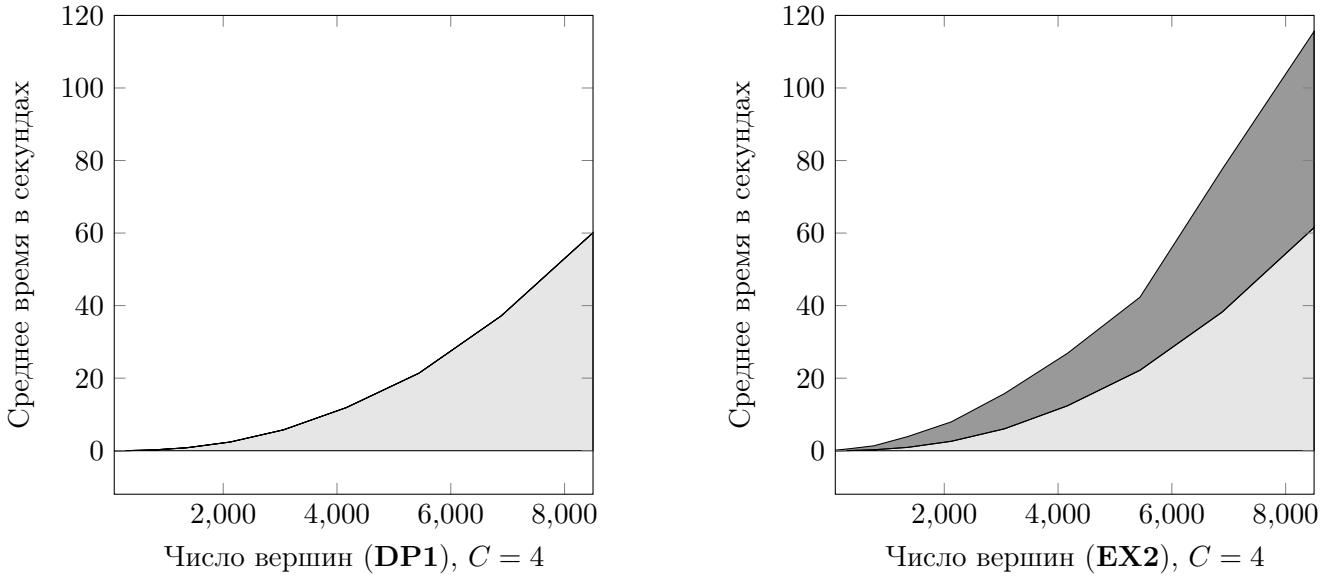
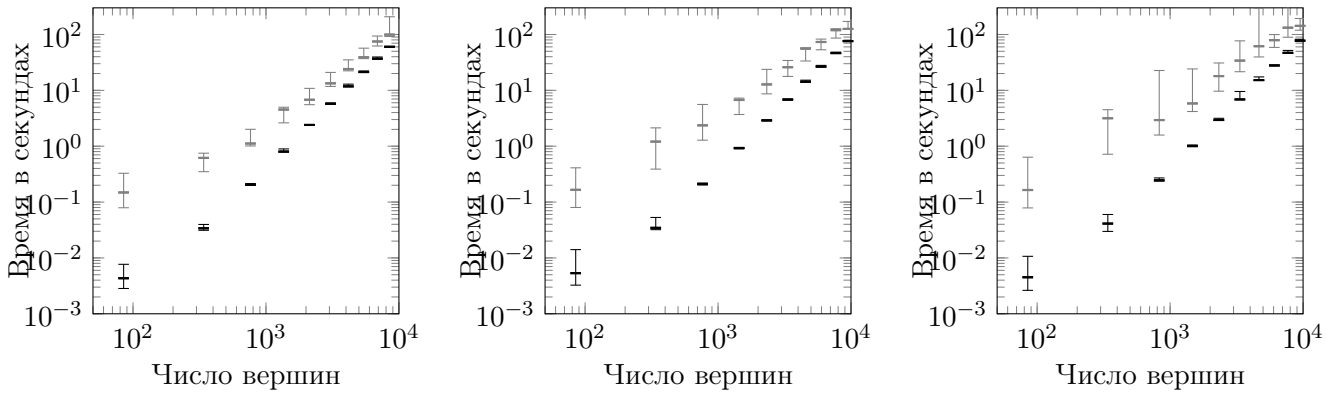


Рисунок 5: Сравнение времени, потраченного на предварительную обработку (светлым), и времени, потраченного на само решение (тёмным).



(a) Тесты типа 1 (b) Тесты типа 2 (c) Тесты типа 3

Рисунок 6: Вариативность времени работы (**DP1** чёрным, **EX2** серым).

4 Редукция данных с доказуемыми оценками результативности

В разделе 3.4.3 на рисунке 1 мы видели, как редукции данных сжимают вход до очень малых размеров, практически не зависящих от размера входа. Это поднимает вопрос о наличии редукции, способной сжимать вход до доказуемо малых размеров. Попробуем найти такую редукцию хотя бы для более простой задачи MinPSC(t). (Напомним, в MinPSC(t) не требуется найти оптимум, но лишь сравнить его с t.)

Известно, что для задачи распознавания MinPSC(t), параметризованной по числу обязательных компонент, существует кернелизация, то есть вход задачи можно за полиномиальное время свести к ядру, размер которого зависит только от параметра. Однако зависимость эта произвольна и может быть экспоненциальной.

Кернелизация используется, как легковесная редукция данных, за которой следует решение задачи ресурсоёмким алгоритмом. Значит, от эффективности редукции данных может значительно зависеть скорость дальнейшего решения задачи. Интересным является вопрос существования ядра полиномиального от параметра размера. Этим вопросом мы и задаёмся в этом разделе. Мы докажем, что у задачи MinPSC(t), параметризованной по числу обязательных компонент, не может существовать ядра полиномиального размера, если не схлопывается полиномиальная иерархия.

4.1 Трудность редукции данных

В этом разделе мы докажем следующую теорему.

Теорема 4.1. *Задача MinPSC(t), параметризованная по числу обязательных компонент, является WK[1]-трудной. Это верно даже для метрической задачи MinPSC(t).*

Доказывать теорему будем через сведение задачи Set Cover(t) к задаче MinPSC(t).

Задача 4.2 (Set Cover(t))**.**
Вход: Множество элементов X, набор множеств $s_i \subseteq X, i \in I$ и число $t \in \mathbb{N}$.
Требуется: Определить, существует ли множество $I' \in I$, или *покрытие*, такое, что $|I'| = t$ и $\bigcup_{i \in I'} s_i = X$.

Известно, что задача Set Cover(t), параметризованная по числу элементов $|X|$, является WK[1]-полной [9]. Будем показывать WK[1]-трудность задачи MinPSC(t') через сведение к ней задачи Set Cover(t).

В [5] показано, что любой вход задачи Set Cover(t) можно полиномиально свести ко входу задачи MinPSC. Мы сделаем похожее сведение, но сводить будем Set Cover(t) к

метрическому случаю MinPSC(t'), т. е. к случаю, в котором для любой пары смежных рёбер выполняется неравенство теугольника.

Покажем сведение. Рассмотрим некоторый вход задачи Set Cover(t) с элементами $x \in X$ и множествами $s_i, i \in I$. Множества и элементы удовлетворяют равенству $X = \bigcup_{i \in I} s_i$. Вход задачи MinPSC($t'$) строится следующим образом. Берём граф с множеством вершин $V = \{r\} \cup \{v_i\}_{i \in I} \cup \{u_x\}_{x \in X}$. Будем добавлять в него рёбра. Сначала мы добавим по ребру веса 1 между вершиной r и каждой вершиной $v_i, i \in I$. Теперь для каждого $i \in I$ добавим рёбра веса 2 между v_i и всеми $u_x, x \in X$ такими, что $x \in s_i$. Назовём этот незавершённый граф G_1. Пусть $d(a, b)$ — длина кратчайшего взвешенного пути между вершинами a и b в этом графе. Наконец, сделаем граф полным, добавив ребро веса $d(a, b)$ между каждой парой вершин $\{a, b\}$. Этот полный граф обозначим G_2. Он удовлетворяет неравенству треугольника.

Лемма 4.3. *Любое оптимальное решение p^* задачи MinPSC для графа G_2, описанного выше, имеет следующую структуру:*

(i) $p^*(r) = 1$,

(ii) $\forall i \in I : p^*(v_i) \in \{1, 2\}$,

(iii) $\forall x \in X : p^*(u_x) = 2$.

Доказательство. Сначала оценим нижние границы вершин l. Нетрудно видеть, что $l(r) = 1$, $l(v_i) = 1$ для $i \in I$, и $l(u_x) = 2$ для $x \in X$. Рассмотрим оптимальное решение p^*. Множество I можно разделить на два непересекающихся подмножества I_1 и I_2 таких, что $p^*(v_i) = 1$ для $i \in I_1$ и $p^*(v_i) \geq 2$ для $v \in I_2$. Рассмотрим также множества $X_2 = \bigcup_{i \in I_2} s_i$ и $X_1 = X \backslash X_2$.

Докажем, что $X_1 = \emptyset$. Для получения противоречия допустим $X_1 \neq \emptyset$. Оценим стоимость оптимального решения снизу. Сложим найденные ранее нижние границы вершин l для r, I_2 и X_2. Для I_1 мы в точности знаем мощности, а для каждой вершины из X_1 мощность не меньше 3, так как у каждой из них все соседи на расстоянии не больше 2 находятся в I_1, но любая вершина в I_1 имеет мощность 1, а значит ребра длиной 2 обеспечить не может. При этом так как $X_1 \neq \emptyset$, то хотя бы одна из вершин $\{r\} \cup I \cup X_2$ должна иметь мощность не меньше 3, чтобы соединиться с какой-нибудь вершиной из X_1. Иначе X_1 не будет связано с остальным графом. Так как для каждой из этих вершин мы взяли в сумму лишь нижние границы, не превосходящие 2, то мы можем добавить 1 к сумму для некоторой вершины.

$$\begin{aligned} &\sum_{v \in V} p^*(v) \geq 1 \cdot |\{r\}| + 1 \cdot |I_1| + 2 \cdot |I_2| + 2 \cdot |X_2| + 3 \cdot |X_1| + 1 = \\ &= 1 + |I| + 2|X| + |I_2| + |X_1| + 1 \end{aligned} \tag{6}$$

Теперь дадим вершинам новые мощности p особым образом, получив решение со стоимостью меньше нижней оценки, полученной под предположением $X_1 \neq \emptyset$. Тем самым мы

покажем противоречие предположению $X_1 \neq \emptyset$. Пусть $p(r) = 2$. Для $i \in I_2$ пусть $p(v_i) = 2$ и для $x \in X$ пусть $p(u_x) = 2$. Для $x \in X_1$ найдём любое множество $k(x) \in I$ такое, что $x \in s_{k(x)}$. Ясно что $k(x) \notin I_2$, ведь тогда x лежало бы в X_2. Значит $k \in I_1$. Пусть $K = \bigcup_{x \in X_1} k(x) \subseteq I_1$. Заметим, что $|K| \leq |X_1|$. Для всех $i \in K$ сделаем $p(i)$ равным 2 и для всех $i \in I_1 \backslash K$ сделаем $p(i)$ равным 1.

Теперь вершина r соединена с каждой из v_i, где $i \in I$. Также все вершины u_x, где $x \in X_1$, соединены с некоторыми вершинами из I_1 и вершины u_x, где $x \in X_2$, соединены с некоторыми вершинами из I_2. То есть, граф связен. Посчитаем стоимость такого решения.

$$\sum_{v \in V} p(v) = 1 \cdot |\{r\}| + 1 \cdot |I_1 \backslash K| + 2 \cdot |K| + 2 \cdot |I_2| + 2 \cdot |X|$$

Так как $|K| \leq |X_1|$, можно получить следующее:

$$\sum_{v \in V} p(v) \leq 1 \cdot |\{r\}| + 1 \cdot |I_1| + 1 \cdot |X_1| + 2 \cdot |I_2| + 2 \cdot |X| =$$
$$= 1 + |I| + 2|X| + |I_2| + |X_1|$$

Теперь, используя неравенство 6, получаем $\sum_{v \in V} p^*(v) \geq \sum_{v \in V} p(v) + 1$. Раз p^* оптимальное решение, то получаем противоречие предположению $X_1 \neq \emptyset$. Тогда $X = X_2$ и $X = \bigcup_{i \in I_2} s_i$.

Теперь мы можем доказать, что оптимальное решение p^* удовлетворяет всем трём правилам из утверждения леммы. Мы предположили, что I_2 — подмножество I, в котором стоимость каждой вершины хотя бы 2. Придерживаясь этого предположения, мы можем выдать каждой вершине в графе минимальную возможную для неё стоимость. Вершинам r и $X \cup I_1$ дадим стоимость, равную их нижним границам l, а каждой вершине из I_2 дадим стоимость 2. Конечно, это единственное минимальное по сумме распределение стоимостей, удовлетворяющее нашим предположениям. Тогда для любого оптимального решения p^*, мы показали, что оно должно удовлетворять правилам из утверждения леммы i, ii и iii. □

Покажем эквивалентность примеров Set Cover(t) и MinPSC(t') для некоторого t'. Поставим в соответствие выбранным в покрытие множествам из задачи Set Cover(t) вершины из I, мощность которых равна 2 (вершины I_2). Тогда, взяв любое покрытие, получим допустимое решение задачи MinPSC, так как вершины из I_2 свяжут $I \cup \{r\}$ с X. Также для допустимого решения MinPSC получаем корректное покрытие, так как именно вершины из I_2 связывают $I \cup \{r\}$ с X, а значит соответствуют множествам, дающим в объединении X. Ответ для задачи MinPSC равен $1 + |I| + 2|X| + |I_2|$. То есть восстановить $|I_2|$ из него нетрудно. Также $|I_2|$ — это число множеств в покрытии. То есть в задаче MinPSC минимизируется именно оно. Тогда для сведения Set Cover(t) к MinPSC(t') сделаем t' равным $1 + |I| + 2|X| + t$.

Теперь покажем, что при сведении Set Cover(t) к MinPSC(t') параметр преобразуется полиномиально. Тогда мы покажем, что MinPSC(t') WK[1]-трудна с интересующим нас параметром. Найдём нижние границы стоимостей для вершин полученного графа. Соседи вершины r имеют расстояние 1 до вершины r: это вершины $v_i, i \in I$. Значит, по лемме, вершины из $C_0 = \{r\} \cup \{v_i\}_{i \in I}$ связаны, и образуют обязательную компоненту. Остальные вершины $\{u_i\}_{i \in X}$ образуют не более, чем $|\{u_i\}_{i \in X}| = |X|$ обязательных компонент. Видно, что исследуемый параметр, число обязательных компонент не превышает $|X| + 1$, т. е. полинома от $|X|$. Это мы и хотели получить.

Как утверждается в [9], задача Set Cover(t), параметризованная по числу элементов является WK[1]-полной. Мы показали сведение её к MinPSC(t'), с полиномиальным преобразование параметра, где параметр — число обязательных компонент в графе, т. е. показали сведение с ППП. По определению WK-иерархии [9] это означает, что MinPSC(t'), параметризованная по числу обязательных компонент, является WK[1]-трудной. Таким образом, доказана теорема 4.1.

Из свойств WK[1]-трудности следует [9]:

Следствие 4.4. *Если не схлопывается полиномиальная иерархия, то задача MinPSC(t) не имеет полиномиального ядра с числом обязательных компонент в качестве параметра. Это верно даже в метрическом случае задачи MinPSC(t).*

4.2 Приближённая редукция данных

Не всегда удаётся добиться редукции данных с низкой оценкой на размер ядра. Однако существуют методы сжимать входы задач эффективно, жертвуя стоимостью решения [11]. Отрицательный результат об отсутствии полиномиального ядра у задачи распознавания всё ещё позволяет разработать эффективную редукцию данных для исходной задачи оптимизации. Для такой редукции мы пожертвуем оптимальностью решения. При этом для наперёд заданной ошибки $\epsilon > 0$ мы будем требовать, чтобы из любого α-приближённого решения редуцированного входа можно было восстановить $(1 + \epsilon)\alpha$-приближённое решение исходной задачи при $\alpha \geq 1$. Заметим, что задача MinPSC является APX-трудной, и без параметризации поиск $(1 + \epsilon)$-приближённого решения для произвольного $\epsilon > 0$ является задачей NP-трудной [5].

Введём вспомогательную задачу:

Задача 4.5 (Preconnected MinPSC)**.**

Вход: Четвёрка (G, w, l, f): граф $G = (V, E)$ с функцией весов $w : F \to \mathbb{N}$, отображением $f : V \to \mathbb{N}$ и функцией нижних границ $l : V \to \mathbb{N}$.

Требуется: Найти функцию $p : V \to \mathbb{N}$ такую, что $p(v) \geq l(v)$ для любой вершины $v \in V$, минимизирующую $\sum_{v \in V} p(v)$ при условии связности графа $G_p = (V, E_p)$, где E_p — множество всех рёбер $\{u, v\} \in E$ таких, для которых $\min\{p(u), p(v)\} \geq w(\{u, v\})$, а также рёбер $\{u, v\} \subseteq V$ таких, что $f(u) = f(v)$.

Эта задача отличается от MinPSC нижними границами мощностей l и множеством заведомо связных компонент, задаваемых отображением f: вершины u и v лежат в одной связной компоненте, если $f(u) = f(v)$. Будем считать, что $f(u)$ — это номер компоненты. Заметим, что эта задача более общая, чем MinPSC: вход для Preconnected MinPSC можно получить из входа MinPSC, занулив все нижние границы l и сделав различными значения f на множестве V.

Обязательные компоненты для новой задачи определим аналогично старой: это компоненты связности в остовном подграфе G, состоящем из всех рёбер, которые обязательно войдут в оптимальное решение. Также необходимо, чтобы рёбра, получаемые условием $f(u) = f(v)$ являлись обязательными. Параметризуем задачу опять же по числу обязательных компонент C.

Заметим, что для новой задачи всё ещё существует параметризованных алгоритм: алгоритм из раздела 3 сам сводит MinPSC к Preconnected MinPSC для решения.

Оказывается, для метрического случая задачи MinPSC можно построить эффективную приближённую редукцию данных, результатом которой является вход Preconnected MinPSC. Мы применим метод, схожий с редукцией данных из [2]. Ниже приведён алгоритм сведения. Идея алгоритма — проредить вершины так, чтобы для каждой удалённой вершины осталась достаточно близкая к ней неудалённая вершина. Численно значение такой близости задаётся числом $\gamma > 0$, которое даётся на вход алгоритму, помимо входа задачи MinPSC и нижних

границ вершин.

Алгоритм 3: приближённая редукция данных для метрической MinPSC

Вход: Граф $G = (V, E)$, функция весов $w : E \to \mathbb{N}$, нижние границы вершин $l \colon V \to \mathbb{N}$ и вещественное число $\gamma > 0$.

Выход: Четвёрка (G', w', l', f), являющаяся входом задачи 4.5.

1. найти G_l, обязательный подграф G
2. найти $G_1 = (V_1, E_1), \ldots, G_C = (V_C, E_C)$, компоненты связности G_l
3. $f \leftarrow$ пустой ассоциативный массив
4. $B \leftarrow$ пустой ассоциативный массив
5. **для** $i \in \{1, \ldots, C\}$ **выполнять**
6. $\quad V_i' \leftarrow \emptyset$
7. $\quad S_i \leftarrow$ сортировка вершин V_i по невозрастанию значения l
8. $\quad$ **для** $u \in S_i$ в порядке **выполнять**
9. $\quad\quad A_u \leftarrow \{v \in V_i' \mid w(\{u, v\}) < \gamma\}$
10. $\quad\quad$ **если** $A_u = \emptyset$ **тогда**
11. $\quad\quad\quad V_i' \leftarrow V_i' \cup \{u\}$
12. $\quad\quad\quad f[u] \leftarrow i$
13. $\quad\quad\quad B[u] \leftarrow \{u\}$
14. $\quad\quad$ **иначе**
15. $\quad\quad\quad B[v] \leftarrow B[v] \cup \{u\}$ для произвольного $v \in A_u$
16. $G' \leftarrow G[\bigcup_{i=1}^{C} V_i']$; пусть $G' = (V', E')$
17. $w' \leftarrow$ сужение w на область определения E'
18. $l' \leftarrow$ сужение l на область определения V'
19. **вернуть** (G', w', l', f)

Как видно в строке 5 алгоритм выполняется для каждой обязательной компоненты независимо, а затем прореженные компоненты объединяются в строке 16. Для каждой компоненты с множеством вершин V_i ищется множество $V_i' \subseteq V_i$, которое войдёт в сжатый граф. Также для каждой компоненты нетрудно вычислить функцию f. Веса рёбер и нижние границы для сжатого графа получаются просто сужением области определения функций w и l. Можно заметить, что массив B никак не используется для вычисления результата алгоритма. Действительно, строки 4, 13, 14 и 15 можно убрать, и алгоритм останется корректным. Эти строки добавлены для удобства доказательства корректности алгоритма.

Сначала сделаем проведём небольшой анализ множества вершин, которые мы взяли в сжатый граф. Рассмотрим произвольную вершину $u \in V'$. Раз такая вершина попала в V', то для неё должно быть выполнено условие в строке 10. Значит существует непустое множество $B[u]$. Будем называть это множество *шаром* с *центром* u.

Лемма 4.6. *Для шаров верны следующие утверждения.*

(i) $B[u] \cap B[v] = \emptyset, u \neq v$

(ii) $V = \bigcup\limits_{u \in V'} B[u]$

(iii) $\forall u \in V' \forall v \in B[v] : l(u) \geq l(v)$

(iv) $\forall u \in V' \forall v \in B[v] : w(\{u, v\}) < \gamma$

(v) $\forall u, v \in V' \cap V_i$, *где* V_i — *вершины некоторой обязательной компоненты, выполняется* $u \neq v \implies w(\{u, v\}) \geq \gamma$

Доказательство. Каждую вершину мы добавили в какой-либо шар ровно один раз после того, как проверили для неё условие в строке 10. Значит верны утверждения i и ii. Рассмотрим произвольный шар $B[u]$. Помимо центра u в шаре лежат вершины $B[u]\backslash\{u\}$. Это вершины, которые не попали в V' из-за вершины u, т. е. такие вершины, для которых условие в строке 10 не выполнилось, и они присоединились к шару $B[u]$. Так как вершины из $B[u]\backslash\{u\}$ обрабатываются после самой вершины u, а мы предварительно произвели сортировку вершин в строке 7, то u имеет максимальную нижнюю границу, задаваемую l, среди вершин из $B[u]$. Это утверждение iii леммы. Вершины из $B[u]\backslash\{u\}$ находятся от u на расстоянии меньше γ опять же из-за условия в строке 10. Также верно, что $w(\{u, u\}) = 0$ по метричности w. Значит верно утверждение iv леммы.

Осталось показать утверждение v. Рассмотрим две различные вершины $u \in V'$ и $v \in V'$ из одной обязательной компоненты с вершинами V_i. Без ограничения общности будем считать, что v была обработана алгоритмом 3 раньше, чем u. Предположим, что $w(\{u, v\}) < \gamma$. Если это так, то при обработке вершины u вершина v попадёт в множество A_u в строке 9 алгоритма 3. Значит, вершина u не попадёт во множество V_i' по условию в строке 10, а значит не попадёт и во множество V'. Но вершина u взята из множества V'. Из данного противоречия следует ложность предположения $w(\{u, v\}) < \gamma$. Тогда верно утверждение v. □

Теперь докажем корректность алгоритма 3. и оценим ошибку, получаемую при восстановлении решения из произвольного решения на сжатом входе. Для этого нам понадобится следующая лемма.

Лемма 4.7. *Пусть* $G = (V, E)$ *с функцией весов* w — *вход задачи MinPSC. Пусть* l — *нижние границы вершин, а* C — *число обязательных компонент. Пусть алгоритм 3, выполненый для заданного значения* $\gamma > 0$, *сжал множество вершин до* V'. *Тогда существует решение* p *задачи MinPSC, удовлетворяющее следующим условиям.*

(i) Обязательные компоненты можно связать между собой рёберами, соединяющими только вершины из V'. *Иначе говоря, только вершины из* V' *имеют мощности больше нижних границ вершин.*

(ii) Стоимость решения p превышает стоимость оптимального решения не больше, чем на $4\gamma(C-1)$.

Доказательство. Рассмотрим любое оптимальное решение задачи p^*. Нам неважно, как связаны внутри обязательные компоненты, но поскольку решение оптимальное, то в нём присутствуют обязательные ребра, связывающие каждую обязательную компоненту. Для того, чтобы связать компоненты между собой, необходимо $C-1$ рёбер, связывающих компоненты структурой дерева. Значит, эти $C-1$ *ключевых* рёбер есть в решении p^*. Никаких других рёбер добавлять в граф и не требуется, значит p^* повышает стоимость вершин относительно нижних границ только у вершин, инцидентных ключевым рёбрам.

Предположим, что вершинам выданы мощности в соответствии с оптимальным решением p^*. Рассмотрим множество вершин K, инцидентных хотя бы одному ключевому ребру. Всего таких не больше $2(C-1)$. Для получения искомого решения p выполним следующие две операции.

1. Мощность каждой вершины $u \in K$ понизим с $p^*(u)$ до её нижней границы $l(u)$. При этом некоторые ключевые рёбра могут перестать удовлетворять условию покрытия.

2. Будем пользоваться шарами B, найденными алгоритмом 3. По очереди для каждой вершины u из K найдём то множество $B[v]$, в котором находится u. Вершина v может оказаться вершиной u, это значения не имеет. Повысим текущую мощность t вершины v до $\max\{t, p^*(u)+2\gamma\}$. Скажем, что "переместили обязанность" с вершины u на вершину v.

Докажем, что для такого решения p выполняются условия леммы. Сначала докажем условие i. Для любого ключевого рёбра $\{u_1, u_2\}$ из графа-решения G_{p^*}, поскольку оно имеет вес $w(\{u_1,u_2\}) \le \min\{p^*(u_1), p^*(u_2)\}$, в подграфе-решении G_p есть ребро $\{v_1, v_2\}$, где v_1 и v_2 — вершины, на которые мы "переместили обязанность" с u_1 и u_2 соответственно. Действительно, по неравенству треугольника вес такого ребра

$$\begin{aligned} w(\{v_1,v_2\}) &\le w(\{v_1,u_1\}) + w(\{u_1,u_2\}) + w(\{u_1,v_2\}) \\ &\le \gamma + \min\{p^*(u_1), p^*(u_2)\} + \gamma = \min\{p^*(u_1)+2\gamma, p^*(u_2)+2\gamma\}. \end{aligned}$$

Заметим, что $p(v_1)$ не меньше первого числа в min, а $p(v_2)$ не меньше второго числа в min. Новое ребро теперь связывает обязательные компоненты, в которых находятся вершины u_1 и u_2. Тогда любая пара обязательных компонент, которая была связана ключевым ребром в решении p^*, теперь тоже связана ребром в решении p. Утверждение i леммы доказано.

Докажем утверждение ii леммы. Поскольку мы перемещали обязанность с некоторой вершины u на вершину v в шаре $B[v]$, в котором у v максимальная нижняя граница (по лемме

4.6), то решение ухудшилось при таком перемещении не больше, чем на 2γ. Действительно, если $u = v$, то стоимость просто увеличиалась на 2γ. Иначе, до перемещения стоимости вершин были равны некоторому t и $p^*(v)$ соответственно (стоимость v не менялась, т. к. она не центр шара). В сумме $s_{\text{before}} = t + p^*(v)$. Если $t \geq p^*(v) + 2\gamma$, то стоимости стали t и $l(v)$ соответственно, что в сумме $s_{\text{after}} = t + l(v) \leq t + p^*(v) = s_{\text{before}}$. Если же $t < p^*(v) + 2\gamma$, то стоимости стали $p^*(v) + 2\gamma$ и $l(v)$ соответственно, что в сумме $s_{\text{after}} = p^*(v) + 2\gamma + l(v) \leq p^*(v) + 2\gamma + l(u) = s_{\text{before}} + 2\gamma$.

Так как $|K| \leq 2(C-1)$, то после всех перемещений обязанности стоимость решения изменилась не больше, чем на $4\gamma(C-1)$. Это и есть утверждение ii леммы. □

Лемма 4.8. *Пусть $G = (V, E)$ с функцией весов w — вход задачи MinPSC. Пусть l — нижние границы вершин, а C — число обязательных компонент. Пусть алгоритм 3, выполненый для заданного значения $\gamma > 0$, сжал вход ко входу (G', w', l', f) задачи 4.5, $G' = (V', E')$. Тогда для любого α-приближённого решения ($\alpha \geq 1$) задачи 4.5 на этом входе можно за полиномиальное время найти решение исходной задачи, не превышающее по стоимости $\alpha(R + 4\gamma(C-1))$, где R — стоимость оптимального решения исходной задачи.*

Доказательство. Сначала получим верхнюю оценку на размер оптимального решения задачи 4.5 для сжатого алгоритмом 3 входа задачи MinPSC. Рассмотрим любое оптимальное решение p^* задачи MinPSC. Его стоимость $R = \sum_{v \in V} p^*(v)$. По лемме 4.7 существует решение p, превышающее по стоимости R не больше, чем на $4\gamma(C-1)$. При этом, по утверждению i леммы 4.7 при удалении вершин $V \backslash V'$ из графа остатки обязательных компонент всё ещё будут связаны между собой. Внутри компонент связность не нужна по условию задачи 4.5. Тогда после применения алгоритма 3 для сжатого входа задачи 4.5 существует решение стоимости $R' = R + 4\gamma(C-1) - \sum_{v \in V \backslash V'} p^*(v)$. А поскольку только для $v \in V'$ мы имеем $p^*(v) > l(v)$, то $R' = R + 4\gamma(C-1) - \sum_{v \in V \backslash V'} l(v)$. Это и будет верхней оценкой на размер оптимального решения задачи 4.5 для сжатого входа.

Предположим, что мы нашли α-приближённое решение $\hat{p}'$ задачи для сжатого входа (G', w', l', f). Пусть оно будет иметь стоимость $\hat{R}'$. Покажем, что такое решение можно превратить в решение $\hat{p}$ исходной задачи. Для этого просто возьмём $\hat{p}$ равным $\hat{p}'$ в вершинах V', и равным l в вершинах $V \backslash V'$. Заметим, что для вершин $v \in V'$ значение $\hat{p}'(v)$ не меньше $l(v)$ по тому, как мы выбрали l'. Тогда раз все мощности не меньше нижних границ, то для каждого i от 1 до C вершины i-й компоненты V_i связаны. Между собой компоненты тоже связаны, так как есть ключевые рёбра, добавленные при решении сжатого входа. Значит решение допустимое.

Оценим стоимость полученного решения:

$$\hat{R} = \hat{R}' + \sum_{v \in V \setminus V'} l(v) \leq \alpha R' + \sum_{v \in V \setminus V'} l(v) \leq$$
$$\leq \alpha(R + 4\gamma(C-1) - \sum_{v \in V \setminus V'} l(v)) + \sum_{v \in V \setminus V'} l(v) \leq$$
$$\leq \alpha(R + 4\gamma(C-1))$$

Это и есть утверждение леммы. □

Осталось оценить размер полученного сжатого примера. Оценим число вершин.

Лемма 4.9. *Пусть $G = (V, E)$ и w — вход задачи MinPSC. Пусть l — нижние границы вершин, а C — число обязательных компонент. После применения алгоритма 3 для фиксированного $\gamma > 0$ число вершин в выходном графе $G' = (V', E')$ не превышает $\frac{2R}{\gamma} + C$, где R — стоимость оптимального решения задачи MinPSC.*

Доказательство. Рассмотрим каждую обязательную компоненту V_i графа G отдельно. Нижние границы связывают V_i, а значит в любом оптимальном решении порождённый подграф $G[V_i]$ связен. Стоимость этой связности, или сумму нижних границ, назовём R_i. Сразу заметим, что $\sum_{i=1}^{C} R_i \leq R$.

Вес минимального остовного дерева в порождённом подграфе $G[V_i']$ обозначим T_i'. Раз у каждого ребра между вершинами из V_i' вес хотя бы γ по утверждению v леммы 4.6, то $T_i' \geq \gamma(|V_i'| - 1)$, так как остовное дерево состоит из $|V_i'| - 1$ рёбер.

Вес минимального дерева Штейнера с минимальным весом в графе $G[V_i]$ с терминалами V_i' назовём S_i. Рассмотрим любое дерево Штейнера минимального веса. Совершим обход дерева, пройдя по каждому ребру дважды. (Это можно сделать обходом дерева в глубину.) Возьмём из обхода вершины V_i' в порядке их первого вхождения. Соединим их рёбрами в путь в таком порядке и получим остовное дерево в $G[V_i']$, по весу не превышающее сумму весов рёбер обхода (по транзитивности w), то есть $2S_i$. Тогда вес минимального остовного дерева в $G[V_i']$ тем более не превышает по $2S_i$. Итого $2S_i \geq T_i'$.

Если мы увеличим множество терминалов в задаче о дереве Штейнера с V_i' до V_i, то мы получим задачу о минимальном остовном дереве в $G[V_i]$. Вес его обозначим T_i. Конечно, вес дерева не уменьшится, ведь мы лишь увеличили множество терминалов. То есть $T_i \geq S_i$.

Наконец, вспомним, что нижняя оценка для MinPSC в графе $G[V_i]$ равна T_i, а значит $R_i \geq T_i$. Сократив полученную серию неравенств, получим:

$$2R_i \geq \gamma(|V'_i| - 1)$$
$$2R \geq \sum_{i=1}^{C} 2R_i \geq \gamma \left(\sum_{i=1}^{C} (|V'_i| - 1) \right) = \gamma(|V'| - C)$$
$$\frac{2R}{\gamma} + C \geq |V'|$$

□

Наконец, показав малость ошибки и эффективность редукции при сжатии алгоритмом 3, можно сформулировать теорему.

Теорема 4.10. *Для любого $\epsilon > 0$ вход (G, w) метрической задачи MinPSC с C обязательными компонентами в графе может быть за полиномиальное время сведён к такому входу (G', w', l', f) задачи Preconnected MinPSC с $O(\frac{C}{\epsilon})$ вершинами, что любое α-приближённое решение ($\alpha \geq 1$) задачи Preconnected MinPSC на полученном входе может быть преобразовано за полиномиальное время к $(1 + \epsilon)\alpha$-приближённому решению исходной задачи.*

Доказательство. Сведение будем проводить алгоритмом 3. Пусть стоимость оптимального решения равна R. Мы можем за полиномиальное время найти стоимость минимального остовного дерева R'. Тогда $R' \leq R \leq 2R'$. Выберем γ равным $\frac{\epsilon R'}{4(C-1)}$ и, применив алгоритм 3, получим вход (G', w', l', f) задачи Preconnected MinPSC. Тогда по лемме 4.8 любое α-приближённое решение задачи Preconnected MinPSC можно свести к решению исходной задачи стоимости не больше

$$\alpha(R + 4\gamma(C - 1)) = \alpha \left(1 + \epsilon \frac{R'}{R}\right) R \leq \alpha(1 + \epsilon)R,$$

т. к. $R' \leq R$. По лемме 4.9 число вершин в графе G' не превышает

$$\frac{2R}{\gamma} + C = \frac{8(C-1)R}{\epsilon R'} + C \leq \frac{16(C-1)}{\epsilon} + C,$$

т. к. $R \leq 2R'$. □

Размер входа после редукции действительно стал полиномиальным от параметра. Число вершин зависит от параметра линейно, а значит число рёбер — квадратично. Таким образом, мы показали эффективную редукцию данных для метрического случая задачи MinPSC. Обратим внимание на то, что в условии теоремы α может быть равно единице, то есть необязательно искать приближённое решение. Можно, например, воспользоваться параметризованным алгоритмом из раздела 3. Вообще говоря, решать можно как угодно, хоть эвристикой: априорная оценка качества не требуется.

5 Заключение

В ходе работы была исследована параметризация задачи MinPSC по числу обязательных компонент. Был оптимизирован параметризованный алгоритм, что позволило превзойти существующие алгоритмы решения, использующие формулировки ЦЛП. Было проведено экспериментальное исследование алгоритма, показавшее превосходство параметризованного алгоритма на входах с малым значением параметра. а также эффективность эвристической редукции данных. Была опровергнута гипотеза о возможности существования полниомиального ядра с числом обязательных компонент в качестве параметра. Несмотря на это была показана приближённая редукция данных ко входу вспомогательной задачи полиномиального от параметра размера.

Можно сделать следующий вывод: при возникновении задачи MinPSC нужно попытаться найти нижние границы вершин и найти значение параметра: это нетрудно как с точки зрения реализации, так и с точки зрения ресурсоёмкости. Если параметр оказался большим, то предпочтение нужно отдать одной из формулировок ЦЛП. Если параметр оказался достаточно мал, то можно использовать параметризованный алгоритм. Наконец, в метрическом случае задачи, если параметризованный алгоритм оказывается слишком ресурсоёмким, то можно применить приближённой редукции данных перед запуском параметризованного алгоритма. Ответ может потерять оптимальность, но время работы сократится.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Смирнов Павел Владимирович _____

“31” мая 2020

Список литературы

[1] Bentert M. *Parameterized Algorithms for Power-Efficient Connected Symmetric Wireless Sensor Networks* / M. Bentert, R. van Bevern, A. Nichterlein, R. Niedermeier // Algorithms for Sensor Systems. – Springer, 2017. – P. 26–40.

[2] van Bevern R. *On approximate data reduction for the Rural Postman Problem: Theory and experiments* / R. van Bevern, T. Fluschnik, O. Yu. Tsidulko // Networks. – to appear.

[3] Clementi A. E. F. *On the Power Assignment Problem in Radio Networks* / A. E. F. Clementi, P. Penna, R.Silvestri // Mobile Networks and Applications. – Springer, 2004. – Vol. 9(2). – P. 125–140.

[4] Cygan M. *Parameterized Algorithms* / M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh – Springer, 2015.

[5] Erzin A. I. *On some polynomially solvable cases and approximate algorithms in the optimal communication tree construction problem* / A. I. Erzin, R. V. Plotnikov, Yu. V. Shamardin // Journal of Applied and Industrial Mathematics. – Pleiades Publishing Ltd, 2013. – Vol. 7(2). – P. 142–152.

[6] Flum J. *Parameterized Complexity Theory* / J. Flum, M. Grohe – Springer, 2006.

[7] Fomin F. V. *Kernelization* / F. V. Fomin, D. Lokshtanov, S. Saurabh, M. Zehavi – Cambridge University Press, 2018.

[8] Garey M. R. *Computers and intractability : a guide to the theory of NP-completeness* / M. R. Garey, D. S. Johnson – Freeman, 1979.

[9] Hermelin D. *A Completeness Theory for Polynomial (Turing) Kernelization* / D. Hermelin, S. Kratsch, K. Sołtys, M. Wahlström, X. Wu // Algorithmica. – Springer, 2014. – Vol. 71. – P. 702–730.

[10] Kirousis L. M. *Power consumption in packet radio networks* / L. M. Kirousis, E. Kranakis, D. Krizanc, A. Pelc // Theoretical Computer Science. – Elsevier, 2000. – Vol. 243(1-2). – P. 289–305.

[11] Lokshtanov D. *Lossy kernelization* / D. Lokshtanov, F. Panolan, M. S. Ramanujan, S. Saurabh // Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing - STOC 2017. – ACM Press, 2017. – P. 224–237.

[12] Montemanni R. *Exact algorithms for the minimum power symmetric connectivity problem in wireless networks* / R. Montemanni, L. M. Gambardella // Computers & Operations Research. – Elsevier, 2005. – Vol. 32(11). – P. 2891–2904.

[13] Prim R. C. *Shortest Connection Networks And Some Generalizations* / R. C. Prim // Bell System Technical Journal. – Institute of Electrical and Electronics Engineers (IEEE), 1957. – Vol. 36(6). – P. 1389–1401.

[14] Rappaport T. S. *Effects of radio propagation path loss on DS-CDMA cellular frequency reuse efficiency for the reverse channel* / T. S. Rappaport, L. B. Milstein // IEEE Transactions on Vehicular Technology. Institute of Electrical and Electronics Engineers (IEEE), 1992. – Vol. 41(3). – P. 231–242.

[15] Zalyubovskiy V. *Energy-efficient Area Coverage by Sensors with Adjustable Ranges* / V. Zalyubovskiy, A. Erzin, S. Astrakov, H. Choo // Sensors. – MDPI AG, 2009. – Vol. 9(4). – P. 2446–2460.

[16] Zhang H. *Maintaining Sensing Coverage and Connectivity in Large Sensor Networks* / H. Zhang, J. Hou // Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks. – Auerbach Publications, 2005.