**Министерство образования и науки Российской Федерации**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**
**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»**

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
## к выпускной квалификационной работе

# «Поиск подграфов социального графа, включающих заданное множество пользователей»

Автор: Филиппов Дмитрий Сергеевич _____
Направление подготовки (специальность): 01.03.02 Прикладная математика и информатика
Квалификация: Бакалавр
Руководитель: Фильченков А.А., канд. физ.-мат. наук _____

**К защите допустить**
Зав. кафедрой Васильев В.Н., докт. техн. наук, проф. _____

«___» _____ 20___ г.

Санкт-Петербург, 2017 г.

**Студент** Филиппов Д.С. **Группа** М3439 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования

**Направленность (профиль), специализация** Математические модели и алгоритмы в разработке программного обеспечения

**Консультанты**:

а) Коршунов А.В., канд. физ.-мат. наук, без звания _____

Квалификационная работа выполнена с оценкой _____

Дата защиты «___» _____ 20___ г.

Секретарь ГЭК *Павлова О.Н.* Принято: «___» _____ 20___ г.

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____

**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»**

**УТВЕРЖДАЮ**
Зав. каф. компьютерных технологий
докт. техн. наук, проф.
_____ Васильев В.Н.
«___» _____ 20___ г.

# ЗАДАНИЕ
## НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

**Студент** Филиппов Д.С. **Группа** М3439 **Кафедра** компьютерных технологий **Факультет** информационных технологий и программирования **Руководитель** Фильченков А.А., канд. физ.-мат. наук, доцент кафедры компьютерных технологий

**1 Наименование темы:** Поиск подграфов социального графа, включающих заданное множество пользователей

**Направление подготовки (специальность):** 01.03.02 Прикладная математика и информатика

**Направленность (профиль):** Математические модели и алгоритмы в разработке программного обеспечения

**Квалификация:** Бакалавр

**2 Срок сдачи студентом законченной работы:** «___» _____ 20___ г.

**3 Техническое задание и исходные данные к работе.**

По данному неориентированному невзвешенному графу социальной сети и выделенным в нем вершинам-запросам требуется найти плотный подграф, содержащий все или большинство данных вершин, то есть так называемое сообщество социальной сети.

**4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)**

Пояснительная записка должна демонстрировать новый подход к решению этой задачи, а также его плюсы и минусы по сравнению с предыдущими методами. Должно быть произведено сравнение оптимальности ответов, времени работы, а также других метрик для всех рассмотренных и приведенных алгоритмов.

**5 Перечень графического материала (с указанием обязательного материала)**

Не предусмотрено

**6 Исходные материалы и пособия**

а) Faloutsos C., McCurley K.S. and Tomkins A. Fast discovery of connection subgraphs;
б) Faloutsos C., Tong H. Center-Piece Subgraphs: Problem Definition and Fast Solutions;
в) Ruchansky N. et al. The Minimum Wiener Connector Problem;

г) Gionis A., Mathioudakis M. and Ukkonen A. Bump hunting in the dark: Local discrepancy maximization on graphs.

**7 Календарный план**

| №№ пп. | Наименование этапов выпускной квалификационной работы | Срок выполнения этапов работы | Отметка о выполнении, подпись руков. |
|---|---|---|---|
| 1 | Ознакомление с исходными статьями | 10.2016 | |
| 2 | Поиск новых статей | 11.2016 | |
| 3 | Изучение новых статей, выбор бейзлайна | 12.2016 | |
| 4 | Реализация бейзлайна | 03.2017 | |
| 5 | Исследование темы, предложения улучшений бейзлайна | 03.2017 | |
| 6 | Реализация улучшений, сравнение практических результатов с теоретическими | 04.2017 | |
| 7 | Написание пояснительной записки | 05.2017 | |

**8 Дата выдачи задания:** «___» _____ 20___ г.

Руководитель _____

Задание принял к исполнению _____ «___» _____ 20___ г.

**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»**

# АННОТАЦИЯ
## ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

**Студент:** Филиппов Дмитрий Сергеевич

**Наименование темы работы:** Поиск подграфов социального графа, включающих заданное множество пользователей

**Наименование организации, где выполнена работа:** Университет ИТМО

## ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

**1 Цель исследования:** В рамках работы необходить предложить улучшение существующих методов поиска сообщества по выделенным в графе социальной сети вершинам, предложить алгоритм отсеивания шума в запросе для получения более плотного подграфа в качестве ответа.

**2 Задачи, решаемые в работе:**

а) Провести исследование описанной задачи;
б) Выделить одно или несколько базовых решений;
в) Реализовать выбранные базовые решения и сравнить их результаты с результатами с статьях;
г) Разработать методы улучшения базовых решений, способные учитывать шум в запросах;
д) Реализовать разработанные методы и сравнить полученные результаты с результатами базовых решений.

**3 Число источников, использованных при составлении обзора:** 15

**4 Полное число источников, использованных в работе:** 17

**5 В том числе источников по годам**

| Отечественных | | | Иностранных | | |
|---|---|---|---|---|---|
| Последние 5 лет | От 5 до 10 лет | Более 10 лет | Последние 5 лет | От 5 до 10 лет | Более 10 лет |
| 0 | 0 | 0 | 9 | 3 | 5 |

**6 Использование информационных ресурсов Internet:** нет

**7 Использование современных пакетов компьютерных программ и технологий:** Для реализации алгоритмов был использован язык программирования *Java 1.8*. Также был использован фреймворк *Kryo* для быстрой и автоматической сериализации и десериализации большого объема данных. Для вычислений, использующих большой объем памяти, были использованы технологии *ssh* и *slurm* для подключения и работы на серверах кластера Университа ИТМО, имеющих 128, 256 и 496 Гб оперативной памяти.

**8 Краткая характеристика полученных результатов:** Результаты, полученные в статье, показывают, что на запросах, содержащих шум, предложенное решение работает оптимальнее всех предыдущих. На запросах без шума решение работает не хуже, а иногда даже лучше существующих решений.

**9 Гранты, полученные при выполнении работы:**

**10 Наличие публикаций и выступлений на конференциях по теме работы:**

Выпусник: Филиппов Д.С. _____

Руководитель: Фильченков А.А. _____

«___» _____ 20___ г.

## ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

Последнее время изучение и аналих социальных сетей представляет большой интерес. Один из примеров анализа социальных сетей — нахождение сообществ пользователей. Многие работы исследуют сообщества целого графа, в то время как большой интерес также представляет анализ сообществ, образованных только данным множеством вершин — по данным выделенным вершинам в социальном графе найти плотный подграф, содержащий их все. Эта задача также широко изучена, однако наложение условия на наличие всех вершин в итоговом подграфе не всегда оптимально для нахождения наиболее плотного подграфа, так как этот подход не учитывает возможный «шум» в запросе.

В этой работе представлен метод оптимизации текущих методов решения описанной задачи при условии возможного шума в запросе. Метод показал хорошие результаты на реальных графах социальных сетей, а также улучшил результаты предыдущих работ.

Актуальность исходной задачи (требующей наличие всех выделенных вершин в итоговом подграфе) проявляется во многих областях:

а) Полиция — зная нескольких подозреваемых, выяснить, кто еще мог быть соучастником преступления, участником банды или группировки;
б) Социальные сети — после добавления одного или нескольких друзей в социальной сети, также предлагается еще несколько, которые тесно связаны с недавно добавленными и которых вы вероятно всего тоже знаете;
в) Медицина — по нескольким заболевшим определить других наиболее вероятно инфицированных, используя социальный граф связей и знакомств;
г) Организация мероприятий — если на важное мероприятие требуется позвать несколько спикеров, также понять, кого еще, тесно связанного с этими людьми, хорошо было бы увидеть на этом мероприятии.

Актуальность нашей задачи (требующей наличие не всех, а только большинства выделенных вершин в итоговом подграфе) также проявлется в этих областях, причем, как можно заметить, наша задача более актуальна в реальной жизни:

а) Полиция — определить остальных соучастников преступления, участников банды или группировки, учитывая, что некоторые подозреваемые могли быть взяты только для этого дела и к группировке отношения не имеют;

б) Социальные сети — рекомендация друзей после недавнего добавления нескольких людей, учитывая то, что добавленные друзья могут быть из разных социальных сообществ и связь между ними может быть только через вас;
в) Медицина — определить наиболее вероятно инфицированных, учитывая тот фактор, что некоторые результаты могли оказаться ложноположительными;
г) Организация мероприятий — приглашение людей, наиболее связанных со спикерами, учитывая то, что спикеры могут быть слабо связаны друг с другом.

Практически все существующие по этой теме статьи рассматривают задачу поиска подграфа, содержащего все выделенные вершины. Наш же алгоритм вводит возможность взятия в итоговое подмножество не всех выделенных вершин, а только большинства из них, и показывает результаты лучше, чем предыдущие алгоритмы на запросах, содержащих шум.

В главе 1 приведен обзор темы: введены основные определения, используемые в статье и необходимые для понимания остальных терминов, приведен обзор существующих решений поставленной задачи с кратким описанием каждого решения. В конце главы уточнены требования к работе исходя из изложенной информации.

В главе 2 предложено подробное описание алгоритма, разбитое на фазы. В главе также приведены примеры работы алгоритма на небольших запросах с подробным описанием и иллюстрациями.

В главе 3 приведено описание проведенных экспериментов на реальных данных — какие данные были выбраны для анализа, как строились и проводились эксперименты. Также приведены диаграммы и таблицы, описывающие результаты экспериментов, показывающие и доказывающие улучшения, описанные в этой работе.

# ГЛАВА 1. ОБЗОР ПОНЯТИЙ И СУЩЕСТВУЮЩИХ РЕШЕНИЙ

В данный момент проблема автоматизации различных вещей, таких как поиск сообществ в социальных сетях, сильно актуальна и активно развивается. На данный момент разразботаны различные методики выделения сообществ во всем графе социальной сети [1–4], а также выделения плотного сообщества, содержащего все выделенные вершины в графе [5–8]. Также были предложены методы разбиения вершин в запросе на несколько сообществ [9] — алгоритм DOT2DOT. Однако, почти все эти методики не поддерживают шум в запросе и выдают неоптимальные подграфы на такие запросы. Именно эту проблему мы решаем в этой работе, предложив модификации существующих алгоритмов, позволяющие эффективно выделять искомые подграфы даже если в исходном запросе существует некий шум.

## 1.1. Термины и понятия

В данном разделе описаны основные термины, используемые в других частях представленной работы.

### 1.1.1. Вводные понятия теории графов

**Декартовым произведением** $A \times B$ двух множеств $A$ и $B$ называется множество всех пар $(a, b)$, таких, что $a \in A, b \in B$.

**Граф** — абстрактный математический объект, представляющий из себя множество *вершин*, некоторые пары из которых соединены *ребрами*. Здесь и далее, говоря «`граф`», мы будем иметь в виду «`неориентированный граф`», то есть граф, ребра которого неориентированы. Более формально, **граф** или **неориентированный граф** $G$ — это упорядоченная пара $G := (V, E)$, где $V$ — непустое множество вершин или узлов графа $G$, а $E$ — множество неупорядоченных пар вершин, называемых *ребрами* ($E \subset V \times V$).

Через **V(G)** будем обозначать множество вершин графа $G$, а через **E(G)** множество ребер графа $G$. Через **|V(G)|** будем обозначать количество вершин графа $G$, а через **|E(G)|** количество ребер графа $G$. Если в контексте понятно, о каком графе идет речь, будем просто писать $|V|$ и $|E|$ соответственно.

Через $N_G(v)$ будем обозначать множество соседей вершины $v$ в графе $G$, то есть множество вершин, напрямую соединенных с $v$ ребром: $N_G(v) = \{u | (v, u) \in E(G)\}$.

**Степенью вершины** $v$ графа $G$ называется количество ребер, исходящих из этой вершины, т.е. $deg(v) = |N_G(v)|$.

**Подграфом** графа $G$ называется граф $G' := (V', E')$, такой, что $V' \subseteq V(G)$ и $E' \subseteq E(G) \cap (V' \times V')$. Иными словами, это граф, порожденный подмножеством вершин исходного графа и содержащий только ребра исходного графа $G$ между вершинами этого подмножества.

$G[V]$ называется **порожденным подграфом** графа $G$ множеством вершин $V$, если $G[V] := (V, E[G, V])$, где $E[G, V]$ — подмножество ребер графа $G$, оба конца которых содержатся в $V$, то есть $E[G, V] = E(G) \cap (V \times V)$.

**k-truss** графа $G$ называется максимальным по количеству вершин подграф $G' \subseteq G$, такой, что для каждого его ребра $(v, u)$ количество вершин $w$, таких, что, в $G'$ существуют ребра $(w, v)$ и $(w, u)$, не меньше $k$. Другими словами, $k - truss$ — наибольший по размеру подграф $G'$, для каждого ребра $(v, u)$ которого $|N_{G'}(v) \cap N_{G'}(u)| >= k$.

**Простым путем** графа $G$ называется такой набор его вершин $v_1, v_2, v_3, \ldots, v_k$, что $\forall\ i, j : v_i \neq v_j$, а также $\forall\ 1 \leqslant i \leqslant k-1 : (v_{i-1}, v_i) \in E(G)$.

**Кратчайшим путем** из вершины $v$ в вершину $u$ называется простой путь $v_1, v_2, \ldots, v_n$ минимальной длины, то есть с минимальным $n$. Длина кратчайшего пути из $v$ в $u$ в графе $G$ обозначается как $d_G(v, u)$.

**Диаметром** графа $G$ называется длина максимального по длине кратчайшего пути в $G$. Другими словами, $diam(G) = max_{v,u \in V(G)}\ d_G(v, u)$.

**k-core** графа $G$ называется максимальный по количеству вершин подграф $G' \subseteq G$, такой, что степень каждой его вершины не меньше $k$. Для фиксированного $k$, через $C_k$ будем обозначать *k-core*, а именно набор компонент связности, из которых он состоит. Таким образом, $C_k = \{H_i\}$, где $H_i$ — $i$-я компонента связности, в которой степень всех вершин хотя бы $k$. Число $k$ будем называть **порядком** *k-core*.

Через $\boldsymbol{\mu(G)}$будем обозначать минимальную степень вершин в графе $G$, т.е. $\mu(G) = \min_{v \in V(G)} deg(v)$.

**Core decomposition** или **ядерной декомпозицией** будем называть набор *k-core* для всех возможных $k$: $C = \{C_k\}_{k=1}^{k=k^*}$. Стоит отметить, что из определения *k-core* следует, что $C_1 \supseteq C_2 \supseteq C_3 \ldots \supseteq C_{k^*}$.

**Core index** или **ядерным индексом** вершины $v$ будем называть минимальный по размеру *k-core*, содержащий $v$, т.е. *k-core* с максимальным $k$: $c(v) = \max(k \in [0..k^*] | v \in C_k)$.

**$\gamma$-quasi-clique** графа $G$ называется любой такой подграф $G' \subseteq G$, что он «достаточно плотный», а именно: $\frac{2 \cdot |E(G')|}{|V(G')| \cdot (|V(G')|-1)} \geqslant \gamma$.

### 1.1.2. Социальные сети

**Сообществом** или **Сообществом социальной сети** называется набор вершин графа социальной сети $G$, где все вершины объединяет некоторое свойство или признак. Например, «сообщество любителей рока» или «сообщество акционеров Газпром».

**Социальной кликой** или **кликой** будем называть множество людей в социальной сети, где каждый человек знает всех остальных в этой клике, другими словами, между любой парой различных вершин в этой клике есть ребро.

**Социальной псевдокликой** или **псевдокликой** будем называть множество людей, где необязательно каждая пара различных вершин соединена ребром, однако множество людей все равно достаточно плотно связано. Оценка, насколько хорошо много связана будет зависеть от выбора типа псевдоклики и будет описана дальше в работе, однако во всех описаниях основную роль играет количество ребер в подграфе по отношению к количеству пар вершин ($\frac{2 \cdot |E(G)|}{|V(G)| \cdot (|V(G)|-1)}$).

**Free-rider effect** называется эффект, возникающий при получении ответа на сформулированную задачу (поиск плотного сообщества по набору вершин), который содержит в себе ненужные подграфы — подграфы, которые можно удалить без нарушения оптимальности ответа, тем самым, уменьшив размер итогового подграфа, что является одной из первоочередных целей нашей задачи.

### 1.1.3. Используемые сокращения

**RW** — Random Walks. Идея основана на перемещении из текущей вершины в соседнюю по ребру с вероятностью, пропорциональной весу ребра.

**RWR** — Random Walks with Restarts. Идея аналогична идее RW, однако в этом случае также существует вероятность пойти из текущей вершины в исходную, в которой все было начато.

**Smart-ST** — Smart Spanning Trees. Эвристика для решения задачи Штейнера, а также используемая в статье Gionis et al. [10].

**CSP** — Community Search Problem. Это задача нахождения сообщества в социальной сети, содержащего все выделенные вершины.

**NCSP** — Noising Community Search Problem. Это задача нахождения сообщества в социальной сети, содержащего большинство выделенных вершины (то есть отсеивая шум).

## 1.2. Обзор существующих решений

Задача, рассматриваемая в этой статье, формулируется следующим образом: по данному графу $G$ и набору выделенных вершин $Q \subset V(G)$ требуется решить задачу поиска сообщества, содержащего большинство, но не обязательно все вершины из $Q$. Выделенные вершины из множества $Q$ мы также иногда будем называть «вершинами-запросами», «запросом» или «вершинами из запроса».

### 1.2.1. Исходные решения

а) Задача поиска сообщества по выделенным вершинам рассматривается уже довольно давно. Еще в 2004 году Faloutsos et al. [11] предложили алгоритм нахождения плотного сообщества по двум выделенным в графе вершинам ($|Q| = 2$). В алгоритме показывается неоптимальность метрик плотности «кратчайший путь» и «максимальный поток» между двумя заданными вершинами. Вместо этого, исходный граф представляется в виде электрической сети и используется метрика «доставленный ток между вершинами» — устанавливая напряжение `+1` на первую вершину-запрос и `0` на вторую, находится подграф, который доставляет наибольший ток между вершинами из запроса. Приведенная метрика подходит только для $|Q| = 2$, однако этот алгоритм положил начало изучению этой темы. В дальнейшем многие авторы статей оптимизировали результаты этой статьи.

б) Авторы второй статьи, рассмотренной для изначального изучения [5], предлагают функцию метрики плотности на основе *random walks with restarts* (RWR), дословно переводящееся как *случайные прогулки с возвращениями* (однако мы не будем использовать этот термин), используемую на взвешенном графе. Они рассматривают $r(i,j)$ — вероятность того, что начав в $i$-й вершине-запросе $q_i$, с помощью RWR, где на каждом шаге из текущей вершины мы переходим в соседнюю по реб-

ру с вероятностью пропорциональной весу ребра, мы закончим в вершине $j$. Также вводится $r(Q, j)$, равная сумме $r(i, j)$ для всех вершин-запросов: $r(Q, j) = \sum_{i=1}^{i=|Q|} r(i, j)$. Метрика плотности вводится как $g(H) = \sum_{j \in H} r(Q, j)$. Этот метод показал достаточно хорошие результаты по сравнению со статьей 2004 года, так как расширил возможность поиска подграфа с $|Q| = 2$ до $2 \leqslant |Q| \leqslant |V(G)|$, а также ввел возможность поиска подграфа, содержащего не все вершины, а только хотя бы $k$ из них ($k$ — параметр, данный на входе). Это операция была названа $K\_softAND$ и была успешно реализована в статье. Последующие алгоритмы развивали эту тему, применяли другие метрики и улучшили результаты этого алгоритма, однако задача поиска подграфа, содержащего не обязательно все, а только часть вершин-запросов в ответе, больше практически не поднималась, улучшения операции $K\_softAND$ не рассматривались ввиду количества других возможных применений и улучшений поставленной задачи.

в) Авторы третьей статьи, рассмотренной для изначального изучения [6], предлагают в качестве метрики плотности брать *индекс Винера* — попарную сумму кратчайших расстояний между вершинами из запроса. Авторы статьи пытаются решить проблему большого графа в результате обрабатывания запроса, если вершины-запросы находятся в нескольких сообществах и слабо связаны между собой. Для решения этой проблемы авторы предлагают добавлять в запрос несколько «важных вершин», который будут связывать сообщества, пусть и не очень плотно. Результаты показали, что этот метод работает в несколько раз лучше большинства предыдущих [5, 12] и примерно так же, как методы, основанные на *проблеме Штейнера*. Однако в статье не рассмотрены более поздние методы на основе *проблемы Штейнера*, которые заметно улучшили результаты предыдущих, что делает этот метод менее приоритетным по сравнению с ними.

### 1.2.2. Нахождение оптимальных псевдоклик

Большу́ю часть всех алгоритмов для решения описанной задачи занимают алгоритмы, основанные на нахождении оптимальных псевдоклик с некоторыми дополнительными эвристиками. Рассмотренных в алгоритмах псевдоклик довольно много: например, *k-core* [8], *k-truss* [7], *$\gamma$-quasi-clique* [13] или просто алгоритмы, максимизирующие плотность ребер в полученном подграфе [14], что,

фактически, и является определением псевдоклики. Для каждой из этой псевдоклик алгоритмы развиваются, улучшают полученные результаты, используя новые эвристики. Сравнивать результаты алгоритмов, использующих разные псевдоклики, довольно сложно и вряд ли даст видимые результаты, потому что из-за разницы оптимизируемых функций полученные результаты сильно зависят от исходных графов и запросов. В некоторых случаях определенные псевдоклики будут давать результаты лучше других, в некоторых — хуже, поэтому имеет смысл сравнивать только средние показатели результатов, однако это тоже не дает полного понимания оптимальности или неоптимальности алгоритмов.

Рассмотрим несколько последних алгоритмов для наиболее популярных псевдоклик:

a) X. Huang et al. [7] в качестве псевдоклик выбирают *k-truss*. Однако, просто нахождение оптимального *k-truss* (то есть *k-truss*, содержащий в себе все вершины-запросы, с максимальным $k$) — не оптимально, а также это уже решенная за полиномимальное время задача. Поэтому авторы статьи предлагают находить *k-truss* с максимальным $k$ и минимальным диаметром, что, как они показывают в своей статье, уже NP-полная задача. Однако, эта идея должна показывать хорошие результаты, поэтому авторами была проведено исследование в попытке понять, насколько близкий ответ можно получить за полиномиальное время. Оказалось, что задачу нельзя решить с точностью лучше, чем в $(2 - \varepsilon)$ раз хуже для любого $\varepsilon > 0$ (под точностью подразумевается длина диаметра в полученном ответе). Однако, авторами был предложен эвристический алгоритм, который в худшем случае делает ошибку ровно в $2$ раза, что показывает, что он является оптимальным для решения поставленной авторами задачи. Алгоритм основывается на построении предполагаемого максимального *k-truss* с последующий итеративным удалением вершин, не ухудшающих ответ и уменьшающих длину диаметра. Результаты, полученные в статье также являются довольно неплохими по сравнению с предыдущими статьями [12, 14], однако, даже несмотря на то, что в статье доказана полная оптимальность разработанного алгоритма, оптимальным для решения всей задачи (нахождения плотного подмножества по заданному множеству вершин)

он не является, поскольку авторами был разработан алгоритм только для поставленной *ими* задачи.

б) N. Barbieri et al. [8] в качестве псевдоклик выбирают *k-core*. Однако, и здесь простое нахождение оптимального *k-core* (то есть *k-core*, содержащий в себе все вершины-запросы, с максимальным $k$) — не оптимально, а также уже решено за полиномиальное время [12]. Поэтому авторы статьи применяют эвристики, нацеленные на минимизацию размера итогового подграфа с сохранением его оптимальности. Эти эвристики позволяют свести задачу к поиску ответа в компоненте связности $H^* \subset G$, причем при этом гарантируется, что все возможные оптимальные ответы на исходную задачу лежат в $H^*$. После этого авторы статьи приводят несколько эвристик для минимизации полученного подграфа $H^*$, содержащего все решения задачи. Само утверждение, описанное авторами, не ново, однако выглядит интересным для дальнейших исследований, потому что добавляет в постановку задачи больше информации без потери решений. Результаты статьи показывают, что метод действительно работает лучше и быстрее предыдущих [12, 15], причем результаты улучшены примерно так же хорошо, как и у X. Huang et al. [7]. Исходя из информации, описанной выше, было сделано решение выбрать эту статью как основу для улучшений и расширений на нашу постановку задачи.

в) Y. Wu et al. [14] в качестве псевдоклик выбирают *query-biased* плотность ребер, основанную на *RW*. Авторы нацелены на устранение *free-rider effect*, который проявляется во многих имеющихся алгоритмах, поэтому формулируют новую задачу поиска подграфа, содержащего все вершины-запросы с максимальной *query-biased* плотностью. Результаты статьи показывают, что метод действительно оптимизует существующие решения и практически не проявляет в себе *free-rider effect*. Он показал результаты лучше существовавших на тот момент *k-core* решений [12], $\gamma$*-quasi-clique* решений [13], а также некоторых других решений. Однако, новые решения [7, 8] также практически лишены *free-rider effect*, поэтому по сравнению с ними этот алгоритм будет не оптимальнее.

### 1.2.3. Другие методики

Как уже было рассмотрено ранее, оптимизируемые функции бывают довольно разные. В предыдущей части были рассмотрены псевдоклики, здесь мы рассмотрим несколько других популярных оптимизируемых функций.

а) L. Akoglu et al. [9] немного меняют постановку задачи, пытаясь найти подграф, объединяющий не все вершины в запросе, а различные их группы. То есть идея фактически состоит в разбиении запроса на группы и построения ответа для каждой группы отдельно, чтобы в каждой группе вершины были плотно связаны и объединены каким-то общим свойством, а между собой группы были связаны не очень сильно. Это соответствует разбиению запроса на несколько сообществ. Результаты статьи показали, что этот метод довольно неплохо решает поставленную в статье задачу, однако, как уже было сказано выше, эта задача отличается от нашей и сравниваться с ней довольно сложно. Ее можно свести к нашей задаче, однако это сведение не равносильное и требует дополнительного исследования.

б) A. Gionis et al. [10] в своей статье рассматривают метрику *линейное локальное несоответствие* (она же *linear local discrepancy*), которая равна взвешенной разнице количества вершин-запросов в итоговом подграфе-ответе и количества остальных вершин. Более точно, $g(C) = \alpha p_C - n_C$, где $p_C = |Q \cap V(C)|$, а $n_C = |V(C) \setminus Q|$. Алгоритм, максимизирующий эту функцию, основан на *проблеме Штейнера*, описанной ранее, а также *«Умных» минимальных остовных деревьяв* (они же *Smart-ST*). Отличительной чертой этого алгоритма является возможность решать задачу, используя *локальную модель доступа*, то есть модель, где весь граф изначально неизвестен (или он слишком большой, чтобы его полностью и быстро сохранить в оперативную память или на диск), и API позволяет только делать запросы на получение всех соседей вершины — $get - neighbours$. Эта модель позволяет решать задачу оптимально даже на очень больших графах социальных сетей, таких как *Twitter*, *Facebook*, а также многих других. Исходя из постановки задачи, в итоговом найденном подграфе вполне могут содержаться не все вершины, что совпадает с темой нашего исследования, однако метрика не очень подходит именно к нашей задаче — в ней не учитывается реберная плотность полученного подграфа, а учитывается только соотношение количества вершин. Также в ответ вполне может войти

довольно мало вершин из исходного запроса, нас такое тоже не устраивает.

### 1.3. Уточненные требования к работе

Подведем итог описанного выше обзора имеющихся решений. Большинство текущих рещений достаточно оптимально решают CSP — каждый использует свою метрику, но получает достаточно хорошие результаты. Однако, как мы можем заметить, что решение NCSP, то есть учитывание «шума», почти не поднимается в текущих решениях. Мы отметили рассмотрение задачи NCSP в статьях C. Faloutsos & H. Tong [5], а также A. Gionis et al. [10], однако там эта задача не является основной и цель сфокусировать результаты на этом не является первоочередной. Соответственно, целью нашей статьи будет разработка алгоритма, фокусирующегося на поиске плотного подграфа, не содержащего в себе исходной шум в запросе. Требования к нему будут следующие:

а) Алгоритм должен показывать результаты лучше текущих имеющихся решений [5, 8, 10];
б) Алгоритм должен быть достаточно оптимальным, желательно не проигрывающим по времени работы текущим аналогам;
в) Плюсом будет поддержка обратной совместимости — есть пользователь захочет все-таки найти подграф, содержащий все вершины в запросе, это должно быть возможно;

## Выводы по главе 1

В главе были описаны основные определения, требуемые для понимания статьи и ее терминов, а также приведен обзор текущих решений поставленной задачи. Обзор показал, что за 13 лет было изучено множество подходов к решению CSP, однако изучаемая в этой бакалаврской работе NCSP изучена не очень хорошо, поэтому ее усовершенствование более чем резонно.

В конце главы были приведены новые требования к работе, мотивирующие и показывающие возможность создания алгоритма, работающего лучше предыдущих исследований.

# ГЛАВА 2. ПРЕДЛОЖЕННЫЙ АЛГОРИТМ

Для решения нашей задачи нужно для начала разобраться с двумя вопросами:

а) Как оценивать плотность итогового подграфа?

б) Что такое «большинство вершин-запросов»?

Для сравнения подгафов, полученных разными алгоритмами мы будем использовать реберную плотность и размер итогового подграфа: $density(G) = \frac{2 \cdot |E(G)|}{|V(G)| \cdot (|V(G)|-1)}$, $size(G) = |V(G)|$. Действительно, если ребер в полученном подграфе достаточно много, можно считать, что он плотный. Однако также хочется, чтобы при этом он имел как можно меньший размер. Что же такое «большинство вершин-запросов» в подграфе-ответе мы обсудим чуть позже.

## 2.1. Описание идеи алгоритма

Так как алгоритмов, решающих именно нашу задачу, слишком мало, у нас есть два подхода к решению поставленной задачи — придумать полностью новый алгоритм или усовершенствовать один из существующих, требующих наличие всех вершин-запросов в ответе. Далее для описания задачи поиска сообщества, содержащего все выделенные вершины-запросы мы будем использовать сокращение *CSP*, а для нахождения сообщества, содержащего большинство вершин-запросов — *NCSP*.

Полностью новым алгоритмом будет, например, следующий: перебрать множество вершин, которое по нашему мнению является шумом, найти подграф, содержащий все остальные вершины одним из существующих алгоритмов, требующих наличие всех вершин-запросов, а затем обновить ответ. Это решение имеет место быть и даже будет выдавать наиболее оптимальный ответ, однако понятно, что это слишком долго — перебор множества игнорируемых вершин будет иметь экспоненциальную от их количества асимптотику, которая затем умножается еще и на асимптотику выбранного алгоритма, ищущего CSP для оставшихся вершин. Поэтому мы не будем использовать этот метод, а попытаемся предложить что-нибудь другое.

Более простым путем будет усовершенствование идеи одного из последних алгоритмов, решающих CSP. Действительно — идея (что именно оптимизировать) уже проверена и показала неплохие результаты, и если применить другие эвристики, которые позволят учитывать постановку нашей задачи, может

получиться неплохой результат. Так мы и сделаем: возьмем алгоритм Barbieri et al. [8] и попробуем его расширить на нашу задачу. Статьи Bogdanov et al. [16] и Cui et al. [15] показали, что максимизация минимальной степени вершины является эффективным способом нахождения оптимального сообщества, поэтому наша идея имеет место быть.

Идея алгоритма Barbieri et al. [8] заключается в нахождении в графе $G$ *k-core* с максимальным $k$, а также минимального размера, содержащим все вершины из $Q$. В статье авторы показывают, что это NP-полная задача, а следовательно для ее решения нужны эвристики. Это не является плохой стороной, потому что на данный момент не существует алгоритмов, в которых задача, оптимизируемая для поиска оптимального сообщества, является решаемой за полиномиальное время, а сами алгоритмы показывают хорошие результаты. Так, например, Sozio et al. [12] ищут *k-core* с максимальным $k$, не обращая внимание на минимальность его размера. Эта задача оказывается решаемой за линейное время от количества ребер в графе, однако, этот алгоритм выдает слишком большие подграфы. Введение же параметра $size$, ограничивающего размер итогового подграфа сразу делает задачу NP-полной, что авторы и показывают в статье.

Возвращаясь к статье Barbieri et al. [8], отметим главную идею их алгоритма. Алгоритм опирается на следующее утверждение: если взять ядерную декомпозицию $C = \{C_k\}_{k=1}^{k=k^*}$ графа $G$, а затем найти такое максимальное $k'$, что все вершины из запроса $Q$ лежат в одной и той же компоненте $H^* \in C_{k'}$, то все ответы на CSP находятся в этой компоненте $H^*$. Более формально: $k' = \max\{k | \exists H_i$ — компонента связности $C_{k'}$, т.ч. $\forall v \in Q : v \in H_i\}$. Найденный *k-core* мы будем обозначать $C_Q^*$, а компоненту, содержащую все вершины из $Q$ — $H^*$. Стоит отметить две вещи: во-первых, это утверждение позволяет сразу же перейти нам к более маленькому подграфу без потери решений, а во-вторых, это утверждение верно и для нашей задачи: раз $H^*$ содержит все оптимальные подграфы, содержащие все вершины из $Q$, то и подграфы, содержащие большинство вершин из $Q$, он тоже содержит (возможно, в нашем ответе *k-core* будет более высокого порядка, однако, ничто не мешает нам начать оптимизировать ответ с $H^*$).

Опишем итоговую идею нашего алгоритма: по данному графу $G$ и запросу $Q$, мы будем выделять *k-core* $C_Q^*$ и его компоненту связности $H^*$, содержащую все оптимальные ответы на NCSP. $H^*$ сам вполне подходит под ответ, пото-

му что содержит все вершины из $Q$, однако его размер в среднем получается довольно большим, что нас не устраивает (а также этот подграф содержит все вершины-запросы, а следовательно содержит и шум). Поэтому после нахождения $H^*$ мы предложим эвристики, позволяющие уменьшить его размер, сохраняющие условие на минимальную степень вершины, а следовательно и не ухудшающие плотность ответа.

## 2.2. Описание алгоритма

В этом разделе мы опишем сам алгоритм. Фактически, сейчас мы можем разделить алгоритм на две фазы:

а) Выделение по данному запросу $Q$ *k-core* $C^*_Q$ и его компоненты $H^*$;

б) Уменьшение размера $H^*$ с сохранением свойств на минимальную степень вершины.

На самом деле, в дальнейшем мы разделим вторую фазу еще на несколько, однако об этом будет написано позднее. Сейчас мы попробуем разобраться с каждой фазой по отдельности.

### 2.2.1. Фаза 1. Нахождение $C^*_Q$ и $H^*$

Фазу выделения $C^*_Q$ и $H^*$ мы позаимствуем из статьи Barbieri et al. [8], как и их идею о минимизации *k-core*. Несложно заметить, что ядерная декомпозиция не зависит от запроса, поэтому нет смысла строить ее каждый раз, так как это занимает достаточно много времени. Однако и сохранить ее в оперативной памяти перед всеми запросами мы тоже не можем, потому что ее размер слишком велик для этого. Поэтому мы сделаем предподсчет, который один раз построит ядерную декомпозицию и сохранит ее в сжатом виде на диск. Это будет так называемый индекс, который позволит по запросу $Q$ намного быстрее, чем раньше, находить $C^*_Q$ и $H^*$ из уже построенной ядерной декомпозиции.

В нашем индексе ядерной декомпозиции $C = \{C_k\}_{k=1}^{k=k^*}$ мы будем хранить следующую информацию:

а) Ядерные индексы всех вершин $c(v) = \max\{k \in [0..k^*] | v \in C_k\}$;

б) Для каждого *k-core* $C_k$ будем хранить набор его компонент $C_k = \{H_i\}$.

Описание вышесказанного иллюстрирует рисунок 1:

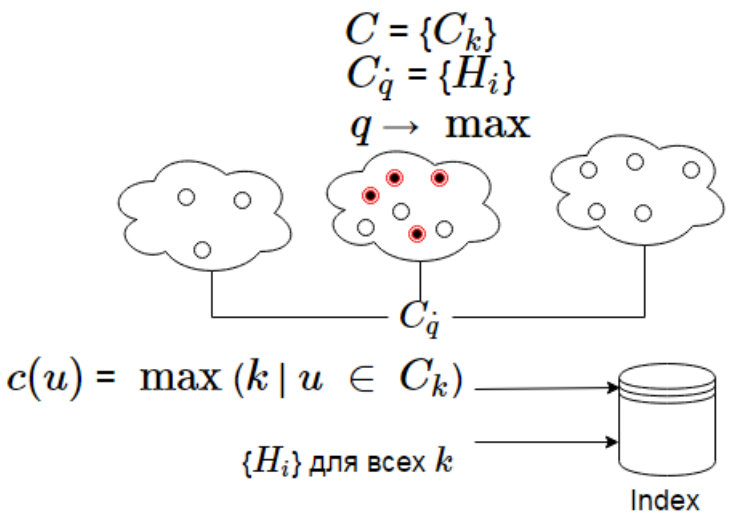



Рисунок 1 – Построение индекса

Стоит также сделать небольшое замечание, что некоторые соседние *k-core* равны: $C_k = C_{k+1}$. Нет смысла хранить их два раза, поэтому все повторяющиеся соседние *k-core* мы будем хранить только один раз. Сейчас это кажется не очень хорошей оптимизацией, однако на реальных данных количество различных *k-core* на несколько порядков меньше общего их количества, поэтому эта оптимизация имеет смысл. При описании дальнейших действий мы будем использовать переменную $h$ — она означает количество различных *k-core* в нашей ядерной декомпозиции, то есть количество *k-core*, которые мы будем хранить в индексе.

Хранить ядерные индексы и набор компонент мы будем в соответствующих структурах данных (хеш-таблицах), позволяющих получать доступ к $c(v)$, набору всех компонент фиксированного *k-core*, а также к компоненте фиксированного *k-core*, в которой находится выбранная вершина $v$, за $O(1)$.

Однако мы до сих пор не сказали, как построить этот индекс. Строить его мы будем довольно просто: сначала построим набор компонент для $C_{k^*}$, а все остальные *k-core* будем строить следующим образом: если уже построены *k-core* порядков $k^*, k^* - 1, \ldots, k$, то для построения $(k-1)$-*core* мы по одной добавим все вершины из $C_{k-1} \setminus C_k$ и ребра, инцидентные им (благодаря свойству, что $C_{k-1} \supseteq C_k$, этого достаточно), а затем обновим компоненты связности, которые благодаря новым вершинам могут объединиться в новые, более большие компоненты.

Эта фаза занимает $O(h \cdot |V(G)| + |E(G)|)$ времени, или же по-простому $O(hn + m)$, где $h$, напомним, количество различных *k-core* в $G$. Однако эту асимптотику мы не будем учитывать в итоговом подсчете, потому что для каждого графа эта операция производится один раз, а затем построенный индекс используется для нахождения $C^*_Q$ и $H^*$.

Так как же найти $C^*_Q$ и $H^*$ по построенному индексу? На самом деле, очень просто. Так как $C_1 \supseteq C_2 \ldots \supseteq C_{k^*}$, то для нахождения *k-core* наибольшего порядка, содержащего все вершины $Q$ в одной своей компоненте, мы можем использовать двоичный поиск по порядку *k-core*. Чтобы проверить для фиксированного $C_k$, правда ли, что все вершины из $Q$ находятся в одной его компоненте, мы достанем из индекса информацию о компоненте для каждой вершины-запроса (напомним, что эта операция выполняется за $O(1)$), а затем проверим, что все эти числа получились одинаковые. Таким образом, проверка шага двоичного поиска происходит за $O(|Q|)$.

Суммируя все вышесказанное, получаем, что шаг нахождения $C^*_Q$ и $H^*$ выполняется за $O(|Q| \cdot \log(h))$.

### 2.2.2. Фаза 2. Уменьшение размера $H^*$

После выделения из графа $G$ *k-core* $C^*_Q$ и его компоненты связности $H^*$, содержащей все вершины из $Q$, как мы уже отмечали, можно заметить, что $H^*$ — ответ на нашу задачу, ведь степени всех его вершин хотя бы $k$ (то есть это *k-core*), причем это $k$ — максимально возможное (по построению $H^*$). Однако, не соблюдается одно оставшееся условие в постановке задачи — этот *k-core* не минимального размера. Это мы и попытаемся исправить в этой и 3-й фазах. Именно в этой фазе мы ставим перед собой следующую задачу: сейчас $H^*$ содержит все вершины-запросы, однако мы знаем, что среди них может быть шум. Задача этой фазы — выделить подграф, не содержащий шумовые вершины.

Напомним, что задача нахождения минимального по размеру *k-core* — NP-полная. Какие эвристики мы можем применить? Первая мысль, которая приходит в голову — удалять слабо связанные вершины или подграфы из $H^*$, тем самым уменьшая его размер. Однако, так как степени всех вершин хотя бы $k$, сложно понять, какие вершины слабо связаны и удаление каких подграфов не нарушит свойство на степени вершины, то есть оставит подграф плотным.

Нами был выбран обратный подход — добавление вершин. Будем строить итоговый подграф $H_{min}$ следующим образом: возьмем все вершины-запросы $Q$,

удалим остальные вершины и ребра. Затем будем добавлять по одной вершине с некоторым приоритетом, тем самым постепенно увеличивая наш текущий подграф $H_{min}$. В каждый момент времени, когда текущий граф будет удовлетворять ответу на нашу задачу (то есть что компонента $H_{min}$, содержащая наибольшее количество вершин-запросов, содержит «большинство вершин из $Q$», а также она довольно «плотная»), мы будем обновлять ответ. Понятно, что процесс не бесконечный — количество вершин в текущем ответе постоянно увеличивается, но подграф больше, чем $H^*$, мы не получим. Однако перед нами встает три глобальных вопроса:

а) Какой выбрать приоритет для добавления вершин?

б) Когда именно обновлять ответ (что такое «большинство вершин из $Q$», что такое «компонента довольно плотная»)?

в) Когда останавливать добавление вершин?

**а. Какой приоритет использовать для добавления очередной вершины?**

Вариантов приоритета может быть множество. Однако давайте подумаем, что для нас важно в первую очередь. Во-первых, мы хотим, чтобы вершины, образующие сообщество (то есть набор вершин-запросов без шума), как можно быстрее объединились в связную компоненту и начали наращивать ее плотность. Для этого первым приоритетом сделаем количество компонент, которое объединяет вершина при добавлении. То есть, $p_1(v) = |A'| - |A|$, где $A$ — множество компонент до добавления, а $A'$ — после. Однако довольно часто при добавлении очередной вершины количество компонент остается неизменным. Что делать в этом случае? Так как для нас также важны степени вершин в итоговом подграфе, давайте акцентируем на этом внимание. При добавлении вершины $v$ степени вершин ее соседей, уже содержащихся в текущем подграфе, увеличиваются на 1. Нам бы хотелось максимизировать это количество. Однако, нужно также учесть только что добавленную вершину и ее степень. Сделаем вторым приоритетом $p_2(v) = |N_{H_{min}}(v) \cap \{v \in V(H_{min}) | deg(v) < \mu(H^*)\}| - \max(0, \mu(H^*) - |N_{H_{min}}(v)|)$, где $N_{H_{min}}(v)$ — множество вершин из $H_{min}$, соединенных с $v$ ребром. Другими словами, мы учитываем со знаком плюс количество вершин-соседей вершины $v$ из $H_{min}$, степень вершин которых еще меньше требуемой $\mu(H^*)$, а со знаком

минус учитываем количество ребер, которое не хватает только что добавленной вершине до требуемой степени $\mu(H^*)$.

**b. Что же такое «большинство вершин из $Q$», что значит «компонента довольно плотная»?**

Мы будем говорить, что подграф $H \subset G$ содержит большинство вершин из $Q$, если количество вершин-запросов в нем хотя бы $\alpha(|Q|) \cdot |Q|$. $\alpha(|Q|) \in (0, 1]$ или просто $\alpha$ — некий коэффициент, зависящий от количества вершин-запросов. Какие же значения должно принимать $\alpha$? С одной стороны, мы хотим, чтобы «большинство вершин» было действительно большинством, поэтому мы будем считать, что шума в запросе меньше $\frac{|Q|}{2}$ (то есть $\alpha \geqslant 0.5$). С другой стороны, даже $\frac{|Q|}{2}$ вершин почти всегда слишком много для шума.

Нас также интересует, что значит «компонента довольно плотная». Будем говорить, что компонента довольно плотная, если в ней достаточно много ребер (то есть и плотность достаточно большая). Для выбора границы количества ребер было попробовано несколько функций, оптимальной оказалось следующая: $|E(H_{min})| \geqslant (V(H_{min}) - |Q|) \cdot \mu(H^*) + \beta(|Q|) \cdot |Q|)$, то есть у всех вершин-запросов степень вершин берется равной $\beta$ (еще один параметр), а у остальных она должна быть хотя бы $\mu(H^*)$ (на самом деле, если в запросе есть шум, степень вершины должна быть больше, однако, если шума нет, это неправда).

На самом деле, условие на плотность компоненты можно не учитывать, а просто проверять, что текущий подграф содержит «большинство вершин из $Q$» — ведь в зависимости от этих условий мы только решаем, обновлять ли ответ. Однако, условие на плотность компоненты было добавлено с целью уменьшения количества обновлений ответа и уменьшения времени работы алгоритма.

Для выбора оптимальных значений параметров $\alpha$ и $\beta$ по фиксированному $|Q|$ было решено провести исследование. Исследование проводилось на тех же экспериментах, что и сравнение различных алгоритмов (об этом подробнее в главе 3). Результаты исследований приведены в таблице ниже ($k$ — порядок *k-core*, то есть $k = \mu(H^*)$).

Таблица 1 – Оптимальные значения параметров $\alpha$ и $\beta$ для различных $|Q|$

| $\|Q\|$ | $\alpha$ | $\beta$ |
|---|---|---|
| 2 | 1 | 1 |
| 3 | 2 / 3 | 1 |
| 4 | 1 / 2 | 1 |
| 5 | 3 / 5 | 2 |
| 6 | 2 / 3 | 2 |
| 7 | 4 / 7 | 3 |
| 8 | 3 / 4 | 3 |
| $> 8$ | 7 / 10 | 4 |

**c. Когда останавливать алгоритм добавления?**

Понятно, что остановить его можно, когда мы например добавили все вершины из $H^*$. Однако, так как размер $H^*$ может быть достаточно большим, это не самое оптимальное решение — после того, как наш подграф станет связным, он уже будет содержать все вершины-запросы, а следовательно и весь шум, в то время как наша задача как раз от него избавится. Поэтому будем останавливать алгоритм, когда все компоненты объединились в одну, то есть текущий подграф стал связным.

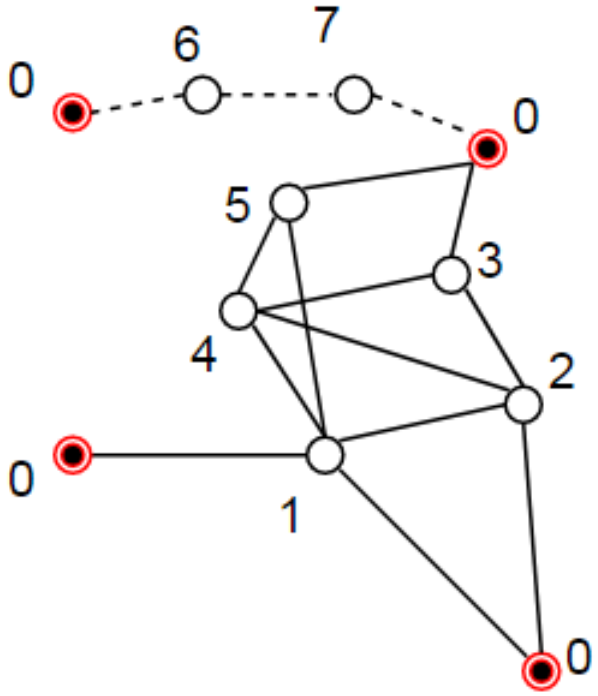



Рисунок 2 – Пример работы фазы 2

Например, пусть $H^*$ выглядит так, как на рисунке 2 (таким он взят для упрощения и более простого объяснения и понимания написанного выше текста). Вершины-запросы в нем отмечены красным, их мы возьмем первыми (шаг $0$). Затем будем добавлять вершины по одной, в порядке, как показано на рисунке: сначала добавляем вершину $1$, потому что она объединяет две компоненты, затем $2$, потому что у нее максимальный второй приоритет, и т.д. Закончим мы, когда $H_{min}$ станет связным, то есть в данном случае, когда мы добавим все его ребра. Однако, оптимальным ответом будет подграф, не содержащий пунктирные ребра, потому что он более плотным. И, как мы видим, действительно, не добавленная вершина — явный шум, добавлять ее в ответ не стоит.

### 2.2.3. Фаза 3. Восстановление условия на степень вершин

После того, как мы выполнили фазу $2$, мы нашли подграф, который будем обозначать $H^*_{min}$. Этот подграф можно было бы вернуть как ответ, но, как мы только что видели на рисунке 2, вполне может оказаться, что $H^*_{min}$ далек от идеального ответа. Почему? Ведь мы добавляли вершины по одной с, как нам казалось, оптимальным приоритетом. На самом деле, так как в первую очередь мы стремимся добавлять вершины, которые лучше всего объединяют наши текущие компоненты, и следим за степенями вершины только во вторую очередь (при этом фактически мы только пытаемся удовлетворить условие на степени вершин, но это не факт, что точно получится), ответ может получиться неоптимальным. Поэтому с $H^*_{min}$ надо что-то сделать, чтобы удовлетворить условие на степени вершин.

Сейчас может показаться, что фаза $2$ была вообще лишней, ведь мы из *k-core*, который достаточно плотно связан благодаря условиям на степени вершин, сделали что-то непонятное, казалось бы только хуже. Это не так, потому что во-первых исходный $H^*$ содержал все вершины-запросы, в том числе шум, а подграф-ответ может быть намного плотнее. Во-вторых, это только временно, сейчас мы применим несколько эвристик и подграф станет плотным *k-core* бóльшего порядка.

Для начала давайте удалим слабо связанные вершины. Так как теперь наш подграф не *k-core*, мы можем предложить логичные условия для этого. Вспомним про наш введенный параметр $\beta$, который отвечал за минимальные степени вершин-запросов. Давайте удалим все вершины-запросы, имеющие степень меньше $\beta$, то есть не подходящие под наше описание (такие вершины могли

остаться, потому что мы смотрели только на общее количество ребер подграфа, а не на степени вершин).

После удаления слабо связанных вершин-запросов выполним еще раз фазу 1 уже на оставшихся запросах. На самом деле, всю фазу нам выполнять не нужно, от этой фазы нам нужно только новое значение $k^{opt}$ — наибольший порядок *k-core*, в котором все оставшиеся вершины-запросы лежат в одной компоненте связности.

Теперь мы сделаем главный шаг этой завершающей фазы. Его идея заключается в следующем: возьмем все оставшиеся выделенные вершины и найдем минимальное количество не выделенных, которые их соединяют. То есть, взяв множество оставшихся выделенных вершин $Q' \subseteq Q$, мы хотим найти такое множество вершин $V_{opt} \subseteq V(H^*_{min}) \setminus Q'$ минимального размера, что $G[V_{opt} \cup Q']$ — связен (напомним, $G[V]$ — подграф, порожденный множеством вершин $V$). Это даст нам «каркас» подграфа $H^*_{min}$, который мы сможем после этого нарастить, почти как в фазе 2.

То, что мы описали выше — ничто иное как задача Штейнера. Оригинальная задача Штейнера звучит так: по данному взвешенному графу $G$ и выделенным в нем множеству вершин $Q$, найти минимальное остовное дерево на этих вершинах. Занимательный факт состоит в том, что если $|Q| = 2$, то задача сводится к кратчайшему пути в графе, если $|Q| = |V(G)|$, то задачу сводится к построению минимального остовного дерева в графе, но если оба этих условия не выполняются, задача становится NP-полной. Можно отметить, что в нашей задаче граф не взвешенный, однако задача Штейнера остается NP-полной даже в случае, когда все веса в графе одинаковые. Задача Штейнера была давно изучена, и в статье Kou et al. [17] был представлен оптимальный алгоритм ее решения с аппрокцимацией $2 - \frac{2}{|Q|}$ и доказательством того, что за полиномиальное время задачу Штейнера нельзя решить лучше. Мы воспользуемся решением из этой статьи и применим его к нашей задаче.

Найдя искомый «каркас», применим к нему слегка измененную фазу 2. Во-первых, каркас уже связен и нам не нужен приоритет на количество объединенных компонент. Во-вторых, хотелось бы усовершенствовать эту фазу, чтобы строго гарантировать плотность итогового подграфа. Поэтому мы меняем фазу 2 следующим образом:

а) Добавлять вершины будем только по второму приоритету (на самом деле, можно оставить и первый, просто он не будет нести никакого смысла);

б) Останавливаться будем, когда минимальная степень вершины станет хотя бы $k^{opt}$, ведь мы точно знаем, что на оставшихся вершинах-запросах можно построить $k^{opt}$-*core*;

в) Мы не будем по ходу добавления вершин обновлять ответ, ответом будет просто финальный подграф.

Изменив фазу 2 описанным выше образом, мы запускаем ее на $H^*_{min}$ с удаленными слабо связанными вершинами-запросами и получаем итоговый ответ на задачу.

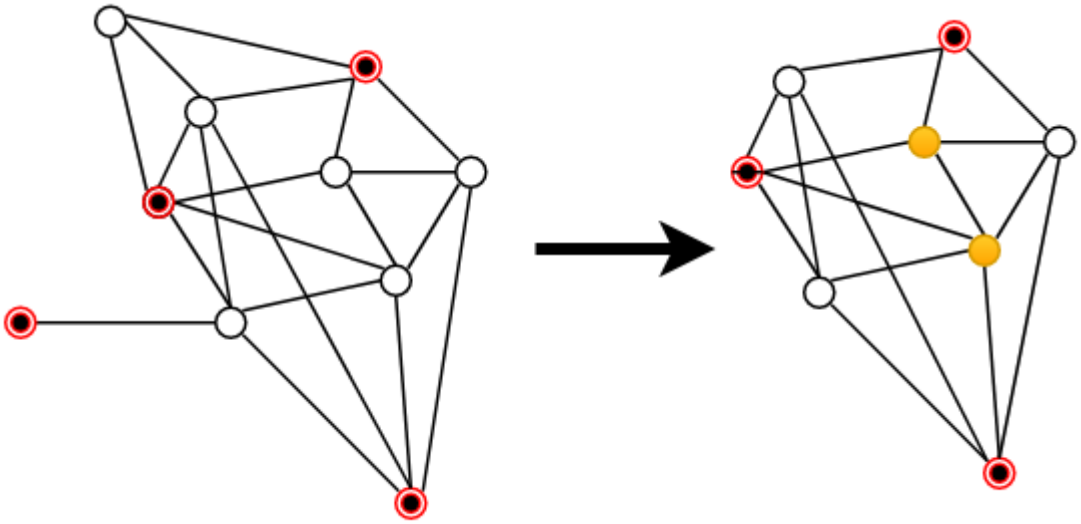


Рисунок 3 – Пример работы фазы 3

Работа этой фазы представлена на рисунке 3, где исходный подграф $H^*_{min}$ (который до фазы 2 являлся *3-core*)находится слева, а полученный после обработки — справа. Сначала мы удалили явно слабо связанную вершину-запрос, а затем выделили с помощью задачи Штейнера две желтые вершины, связывающие 3 оставшихся запроса и образующих тем самым «каркас» из 5 вершин. После этого мы запускаем измененную фазу 2 и получаем ответ, представленный справа на рисунке. Как видно, он получился меньше и плотнее, а также является *4-core* вместо *3-core*.

### 2.3. Теоретическая оценка качества

Сложно оценивать качество алгоритма без проверки на практических данных. Однако в этом разделе мы отметим несколько явных преимуществ по сравнению с имеющимися алгоритмами.

— Наш алгоритм учитывает шум в запросе, если он есть и его меньше половины, а также требует наличие всех вершин в запросе, если явного шума нет;

— Наш алгоритм должен работать не хуже Barbieri et al. [8], потому что мы в итоге находим *k-core* порядка хотя бы $\mu(H^*)$. Наш алгоритм может работать хуже из-за отличающейся эвристики, однако на запросах с шумом мы должны находить *k-core* с бóльшим порядком, а следовательно плотнее;

## Выводы по главе 2

В этой главе был подробно описан предложенный алгоритм решения поставленной задачи. Алгоритм состоял из трех фаз, последовательно оптимизирующих ответ на задачу. Несмотря на то, что задача NP-полная и невозможно предложить алгоритм, решаюший ее за полиномиальное время, наши эвристики позволяют, хоть и в теории, но судить о том, что результаты должны быть лучше, чем у существующих сейчас алгоритмов, однако подтверждение этому можно будет увидеть только после проведения практических результатов.

Также заметим, что наш алгоритм может поддерживать обратную совместимость — несложно будет добавить настраиваемый параметр $minSize$, который будет означать минимальное количество вершин-запросов, которое должно быть в ответе. Тогда в фазе 2 мы не будем прекращать наращивать $H^*_{min}$, пока количество вершин-запросов в нем не станет хотя бы $minSize$, а в фазе 3 не будем удалять слабо связанные вершины-запросы. Тем самым, фактически, мы можем все еще решать CSP, а не NCSP, если это потребуется.

# ГЛАВА 3. ПРАКТИЧЕСКИЕ ЭКСПЕРИМЕНТЫ И РЕЗУЛЬТАТЫ

В данной главе будут описаны практические исследования и полученные результаты нашего алгоритма. Сначала будет описано, на каких данных проводились результаты и как были устроены эксперименты, а после этого будут приведены диаграммы, описывающие сравнение нашего решения с существующими и подробное описание, объяснение и обсуждение результатов.

## 3.1. Описание экспериментов

Эксперименты были проведены на двух реальных датасетах: *DBLP* — граф авторой статей, где ребро между авторами соответствует наличию у них общей статьи и *Youtube* — граф пользователей социальной сети *Youtube*, где ребро между пользователями означает обоюдную подписку. Для экспериментов мы взяли $50$ случайных запросов без шума, $50$ запросов с шумом и $5$ запросов с абсолютным шумом, то есть с вершинами, которые практически никак между собой не связаны. Подробнее об этом дальше.

а) В качестве первого исследования мы взяли $50$ случайных запросов на графе, каждый из которых содержал вершины, довольно сильно коррелирующие между собой. В качестве запроса мы выбирали *k-core* с достаточно большим $k$, а затем распределяли вершины-запросы в нем случайно, однако, следя за тем, чтобы они не были сильно далеко друг от друга, потому что иначе размеры подграфов будут довольно большими (это исследование рассматривается в статье Barbieri et al. [8], у них размеры подграфов получаются примерно в $10$ раз больше, чем у нас. Это обусловлено только выбором вершин для запросов, наш выбор вершин больше соответствует актуальности нашей задачи). Как результат мы считали среднее результатов по этим $50$ экспериментам.

б) В качестве второго исследования мы брали $50$ случайных запросов на графе, каждый из которых содержал вершины, часть из которых довольно сильно коррелирует между собой, а часть (небольшая) — шум и плохо коррелирует с остальными вершинами. Фактически, первая часть вершин соответствует первому исследованию, а вторая часть вершин — просто добавленный шум. В качестве запроса мы выбирали *k-core* с достаточно большим $k$ (то есть запрос, сгенерированный первым исследованием) с добавленными несколькими вершинами из *k-core* с другими $k$, которые,

соответственно, слабо коррелируют с остальными вершинами. Как результат мы считали среднее результатов по этим $50$ экспериментам.

в) В качестве третьего исследования, чтобы оценить поведение нашего алгоритма на практических абсолютно шумных запросах, мы выбираем несколько вершин из совсем разных *k-core*, которые очень слабо или совсем не коррелируют между собой. Исследование показало, что на таких запросах наше решение строит достаточно большие по размеру итоговые подграфы, однако по максимуму отсеивая шум, что является довольно логичным эффектом в таком типе запросов.

После каждого запуска алгоритм на вершинах-запросах мы также проводим валидацию ответа — что в полученном подграфе содержится большинство ($\max(\frac{|Q|}{2}, |Q| \cdot \alpha - \epsilon)$) исходных вершин-запросов, где $\epsilon$ — небольшое число, показывающее, что после шага $2$ мы также могли удалить несколько слабо связанных вершин на третьей фазе, которое для наших исследований мы брали равным $4$.

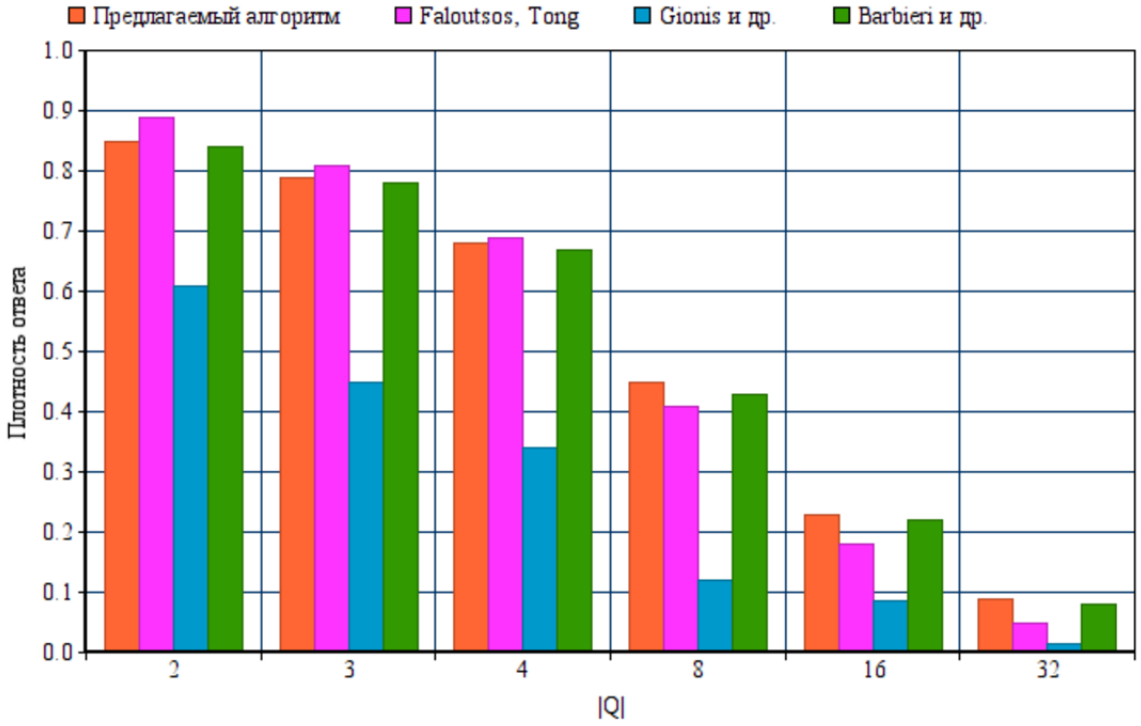
### 3.2. Результаты экспериментов

Здесь мы приведем результаты экспериментов, сравнивающие предложенный алгоритм, алгоритм Faloutsos & Tong [5], Gionis et al. [10] и Barbieri et al. [8]. Для визуализации результатов будут приведены диаграммы с краткими описаниями. Для каждого из двух датасетов будет приведено $6$ диаграм, сравнивающие плотности и размеры полученных рассмотренными алгоритмам подграфов, а также время работы этих алгоритмов (без учета времени построения индекса). Диаграммы строятся для $2$ типов запросов — без шума и с ним соответственно. В каждой диаграмме горизонтальная ось соответствует различным размерам выбранных запросов $|Q|$ (рассмотренные значения $|Q|$: $1$, $2$, $3$, $4$, $8$, $16$ и $32$), а вертикальная ось соответствует плотности полученного подграфа, его размеру или же времени работы алгоритма.

#### 3.2.1. Датасет DBLP

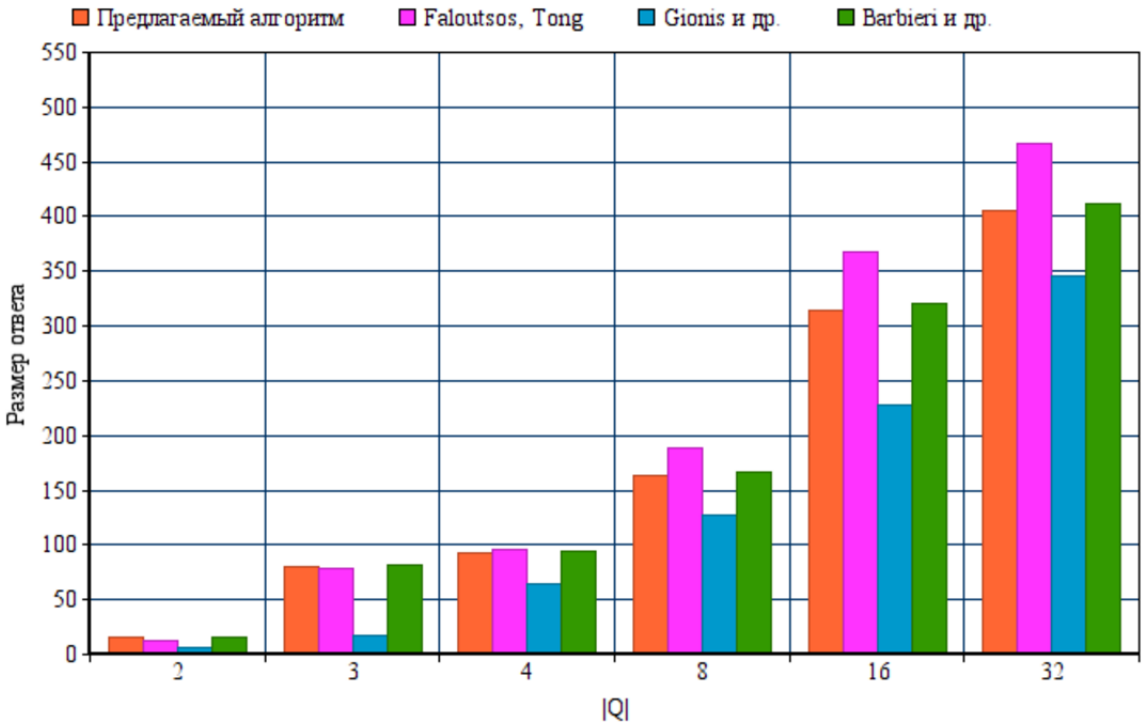
В датасете *DBLP* $317080$ вершин и $1049866$ ребер. Датасет состоит из авторов различных статей, а ребро между авторами соответствует наличию у них общей статьи.

#### 3.2.1.1. Запросы без шума. Плотность подграфа



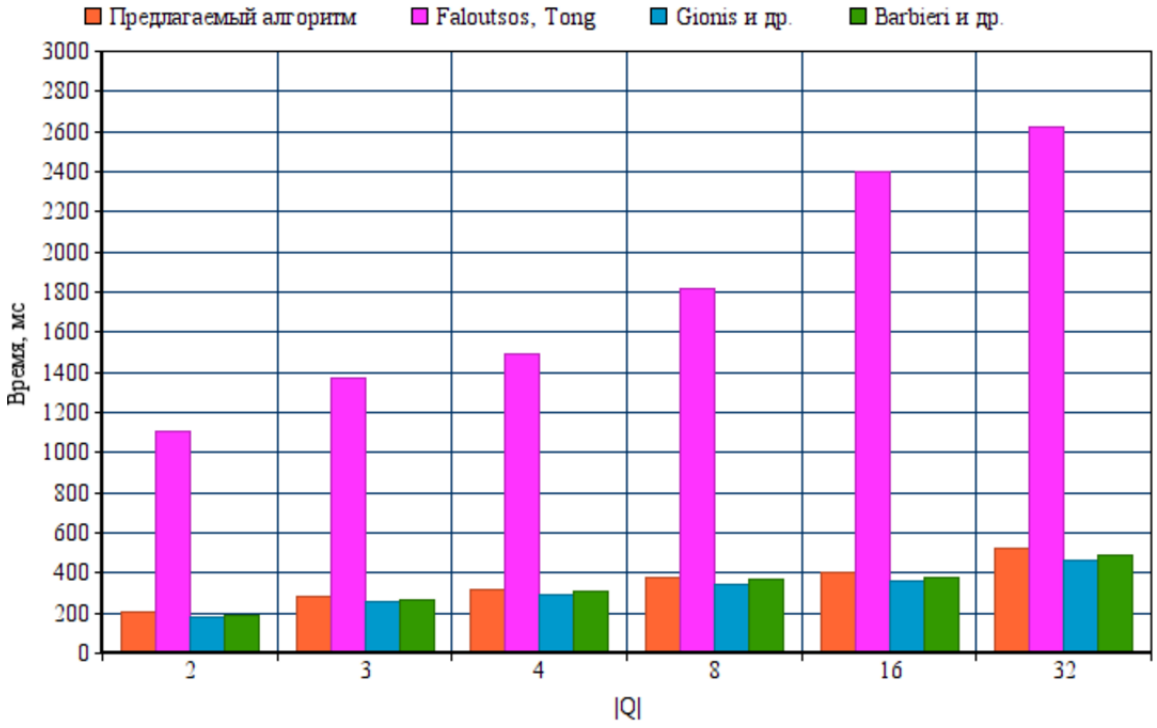

Как видно из диаграммы, на случайных запросах без шума плотность нашего полученного подграфа меньше, хоть и совсем немного, плотности подграфа, полученного статьей Barbieri et al. [8], а значит наш ответ оптимальнее. Также плотность нашего подграфа в среднем больше плотности Faloutsos & Tong [5] (она совсем немного меньше при небольших значениях $|Q|$ из-за структуры их алгоритма, однако при бóльших значениях $|Q|$ мы начинаем сильно выигрывать по плотности) и намного больше плотности Gionis et al. [10], потому что статья этих авторов, хоть и умеет решать поставленную нами задачу, плотность полученного подграфа — не ее первоочередная цель.

### 3.2.1.2. Запросы без шума. Размер подграфа



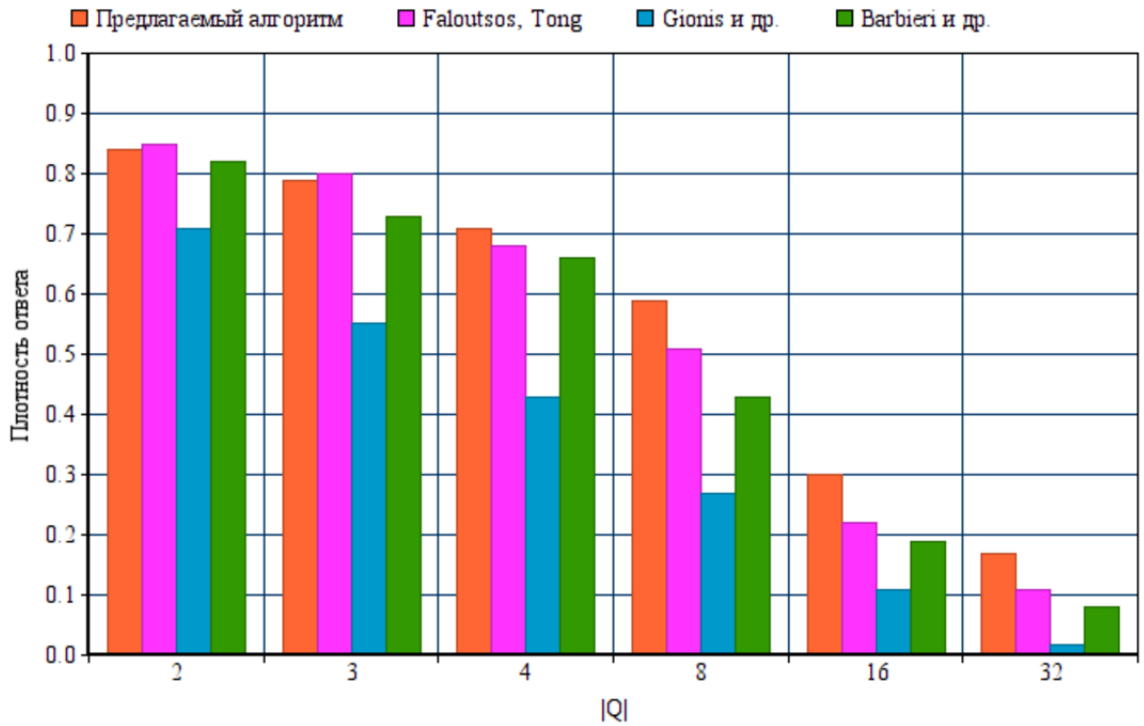

Как видно из диаграммы, на случайных запросах без шума размер нашего полученного подграфа также немного, но меньше размера подграфа, полученного статьей Barbieri et al. [8], из чего становится логичным, откуда получено улучшение по плотности. Размер нашего подграфа в среднем меньше размера подграфа Faloutsos & Tong [5], хоть и для небольших $|Q|$ их решение работает лучше из-за его построения, но меньше размера Gionis et al. [10], потому что в этом как раз и состоит предназначение их статьи.

### 3.2.1.3. Запросы без шума. Время работы



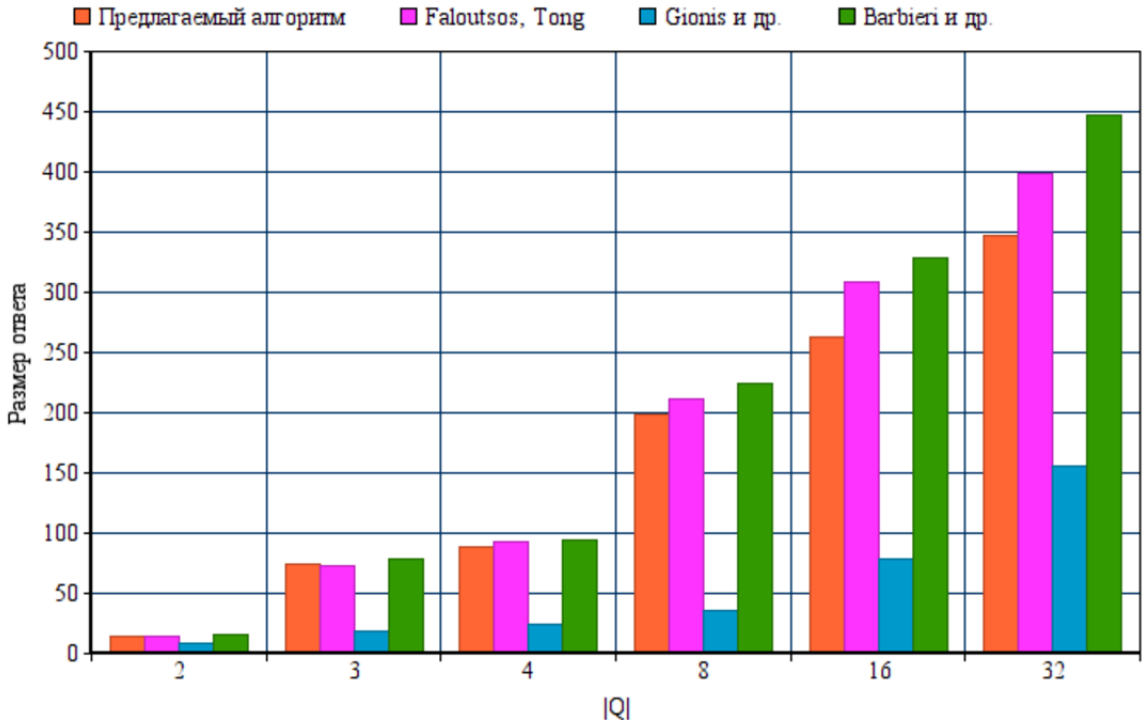

Как видно из диаграммы, по времени мы немного проигрываем статье Barbieri et al. [8], однако этот проигрыш совсем незначительный. Faloutsos & Tong [5] работают сильно дольше нас, а Gionis et al. [10] быстрее, но это не является нашим минусом, посколько плотность подграфов, получаемых этих алгоритмом намного меньше нашей.

### 3.2.1.4. Запросы с шумом. Плотность подграфа



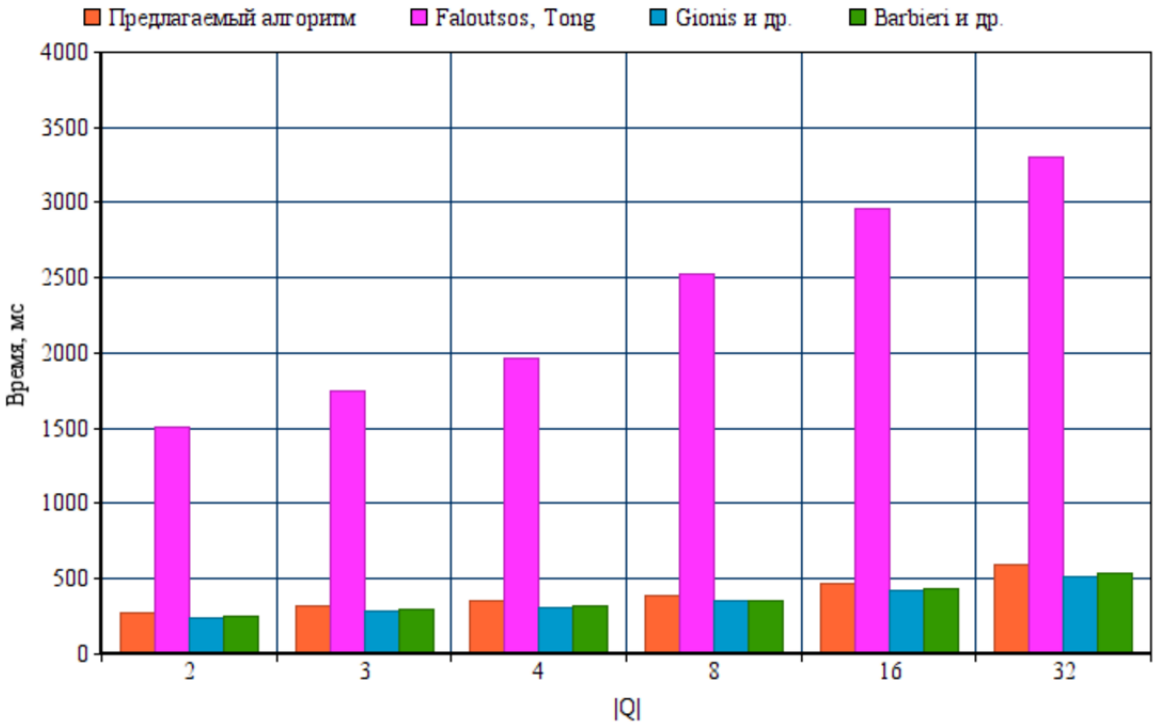
Если в запросе появляется шум, наш алгоритм начинает работать сильно оптимальней. Это видно из представленной выше диаграммы: при небольших значениях $|Q|$ наш выигрыш мал, что логично, потому что шума в таких запросах практически быть не может, однако, при бóльших значениях $|Q|$ мы начинаем сильно выигрывать по плотности и размеру итогового подграфа (см. следующую диаграму).

### 3.2.1.5. Запросы с шумом. Размер подграфа




Как можно видеть из диаграммы, при наличии шума, размер нашего подграфа также меньше размера подграфов остальных решений (кроме Gionis et al. [10], но так и должно быть), что объясняет выигрыш по плотности в предыдущей диаграмме.

### 3.2.1.6. Запросы с шумом. Время работы



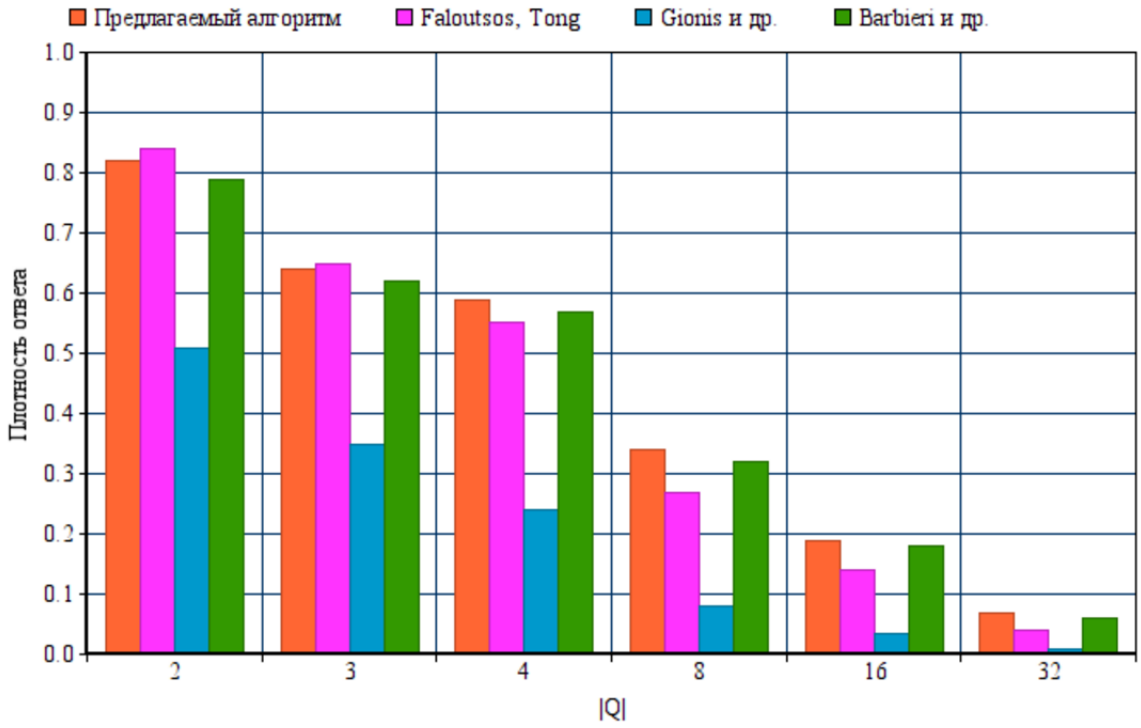

Время работы предложенного алгоритма на запросах с шумом также немного больше времени работы Barbieri et al. [8], однако проигрыш по времени совсем незначительный по сравнению с улучшением результатов плотности и размера.

### 3.2.2. Датасет Youtube

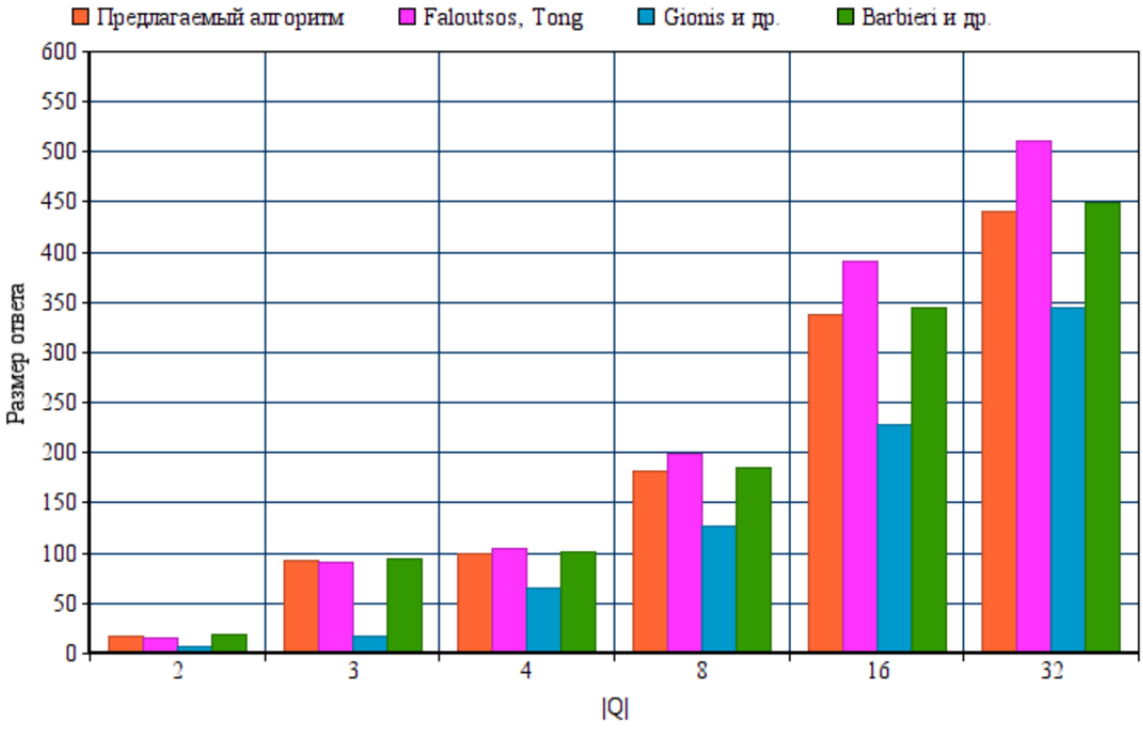
В датасете *Youtube* 1134890 вершин и 2987624 ребер. Датасет состоит из пользователей социальной сети *Youtube*, ребро между пользователями существует, если они друг у друга в друзьях.

Результаты практически совпадают с результатами на датасете *DBLP*, за исключением глобальных изменений во времени работы из-за размера датасета, поэтому комментарии к диаграммам будут приведены только в случае необходимости.

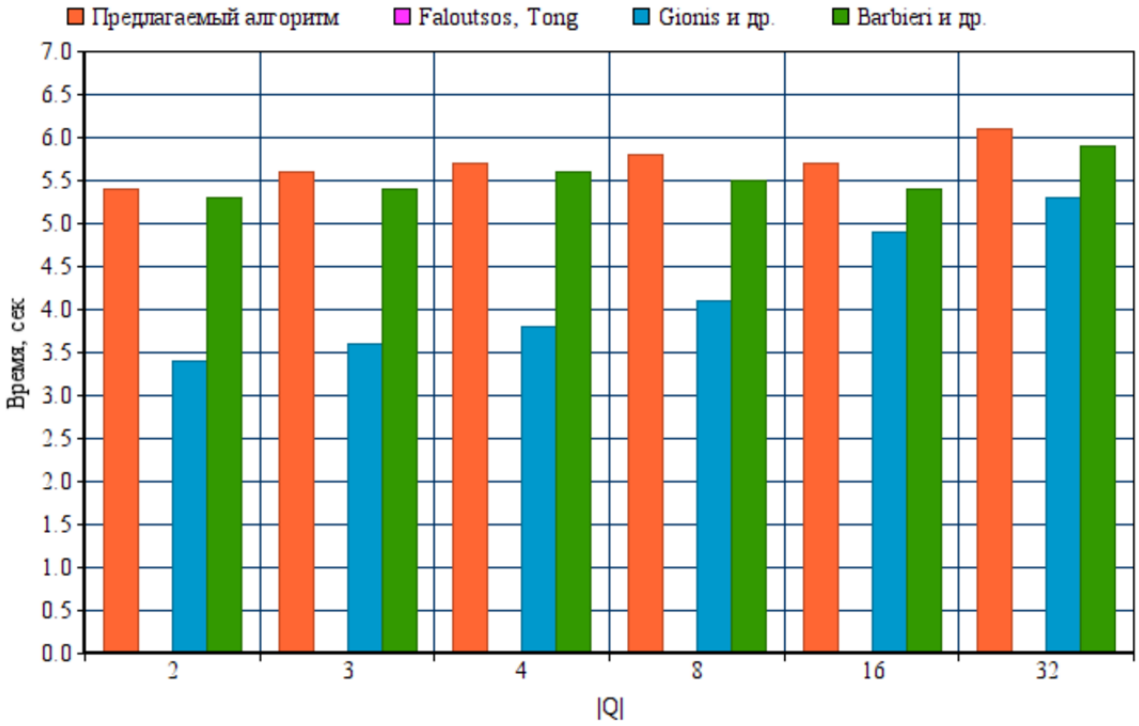
### 3.2.2.1. Запросы без шума. Плотность подграфа




### 3.2.2.2. Запросы без шума. Размер подграфа

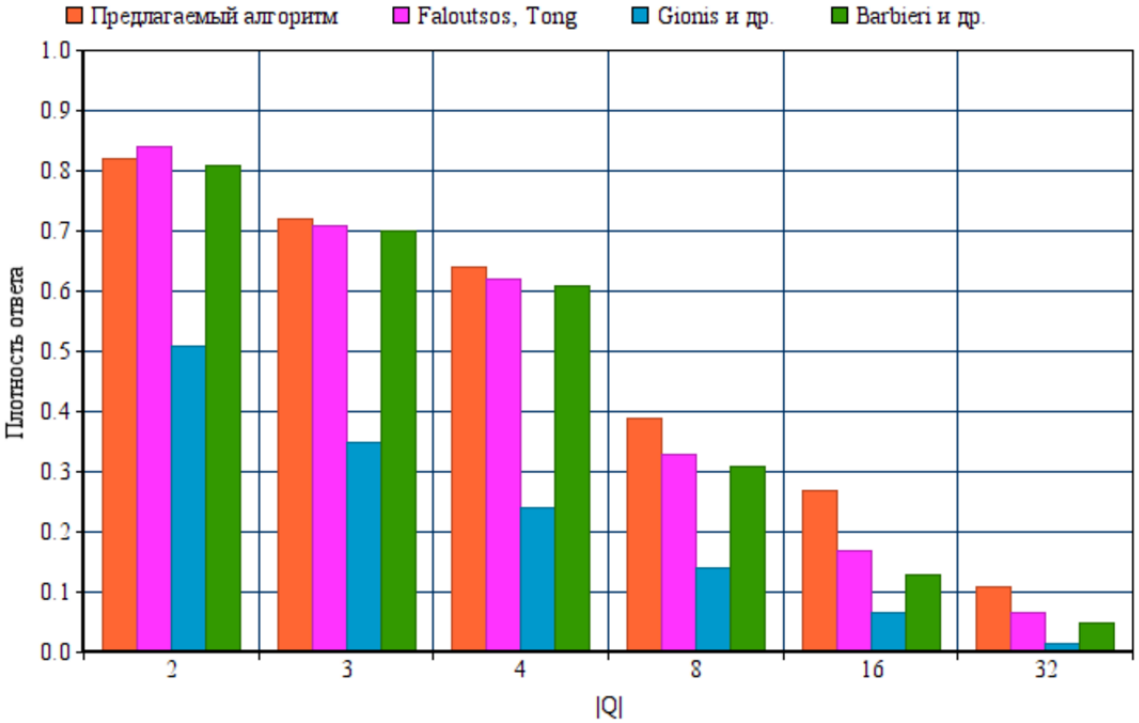


### 3.2.2.3. Запросы без шума. Время работы

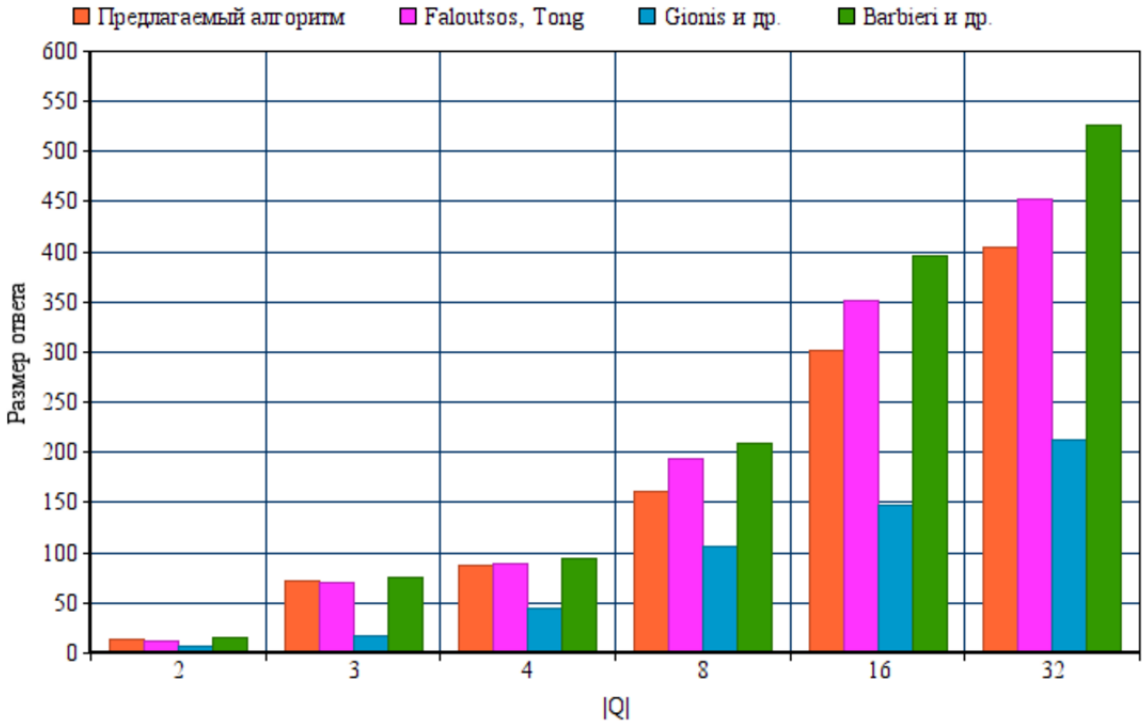



Здесь стоит отметить, что алгоритм Faloutsos & Tong [5] работает слишком долго и чтобы не нарушать визуальность относительности времени работы остальных алгоритмов, этот алгоритм в диаграмме опущен. Также стоит отметить, что время работы изменяется в секундах в отличие от предыдущего датасета из-за значительной разницы в их размерах.

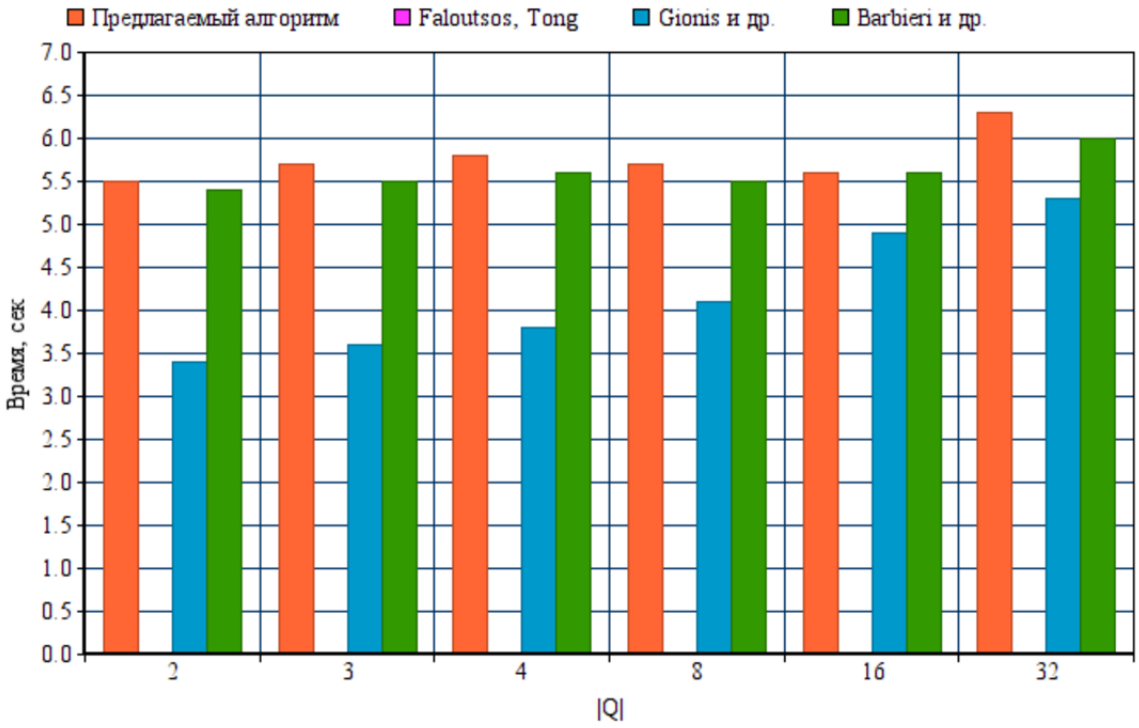
### 3.2.2.4. Запросы с шумом. Плотность подграфа



### 3.2.2.5. Запросы с шумом. Размер подграфа




### 3.2.2.6. Запросы с шумом. Время работы




Здесь время работы алгоритма Faloutsos & Tong [5] не указано по той же причине, что и в диаграмме с запросами без шума.

### 3.2.3. Итоговые результаты

Соберем результаты приведенных диаграмм в таблицах 2 и 3:

В таблицах указано, как соотносится наше решение с тремя рассмотренными. В каждой ячейке через записаны средний результат и максимальный выигрыш соответственно. Так, например, $18.1\%/54.6\%$ в плотности с шумом означает, что мы в среднем выиграли $18.1\%$ по плотности, а максимум по всем экспериментам — $54.6\%$. В размере и времени же $-4.7\%/-14.6\%$ означает, что наш размер на $4.7\%$ меньше в среднем и на $14.6\%$ меньше в максимальном случае соответственно.

Таблица 2 – DBLP - сравнение предлагаемого алгоритма с существующими решениями

| – | Faloutsos & Tong | Gionis et al. | Barbieri et al. |
|---|---|---|---|
| Плотность | +18.2% / +80.0% | +200.6% / +542.9% | 4.2% / +12.5% |
| Размер | -4.5% / -14.4% | +97.4% / +17.7% | -2.9% / -6.25% |
| Время | -80.3% / -83.3% | +11.7% / +7.7% | +5.2% / +2.7% |
| Плотность (шум) | +18.1% / +54.6% | +210.4% / +844.4% | +37.6% / +112.5% |
| Размер (шум) | -4.7% / -14.6% | +247.9% / +87.5% | -11.6% / -22.4% |
| Время (шум) | -82.8% / -84.6% | +12.1% / +10.3% | +8.4% / +6.6% |

Таблица 3 – Youtube - сравнение предлагаемого алгоритма с существующими решениями

| – | Faloutsos & Tong | Gionis et al. | Barbieri et al. |
|---|---|---|---|
| Плотность | +23.3% / +75.0% | +305.4% / +775.0% | +6.5% / +16.7% |
| Размер | -4.6% / -13.7% | +123.3% / +27.9% | -2.6% / -5.3% |
| Время | -320.8% / -325.1% | +39.6% / +15.1% | +3.6% / +1.8% |
| Плотность (шум) | +24.7% / +69.2% | +252.1% / +685.7% | +45.3% / +129.2% |
| Размер (шум) | -4.3% / -17.0% | +223.3% / +27.8% | -14.8% / -23.7% |
| Время (шум) | -323.4% / -331.1% | +40.8% / +14.2% | +2.9% / +0.0% |

Также нас интересует еще одна вещь — не может ли быть так, что на запросе без шума мы выкинем лишние вершины? Проведенные эксперименты показали, что на самом деле такое возможно, и наш алгоритм на запросах без шума может выкинуть несколько вершин, решив, что это шум. Однако, так как на самом деле шума нету и вершины довольно плотно связаны, таких вершин будет мало или вообще не будет. Результаты показали, что мы выкидываем не больше $2-3$ вершин из $Q$ (при $|Q|=32$), что вполне нормально. Напомним, что этого можно избежать, воспользовавшись обратной совместимостью, описанной в конце главы 2 — мы можем немного изменить наш алгоритм, чтобы он решал исходную задачу SCP, тогда все вершины-запросы будут включены в ответ.

## Выводы по главе 3

В главе были описаны методы проведения экспериментов на двух датасетах *DBLP* и *Youtube*, полученные результаты показывают, что наш алгоритм на случайных запросах с коррелирующими вершинами работает даже чуть лучше предыдущего решения [8], однако немного проигрывая ему по времени. Однако, на запросах, содержащих шум, наш алгоритм работает намного лучше существующих и выдает подграфы по плотности и размеру сильно улучшающие имеющиеся решения.

## ЗАКЛЮЧЕНИЕ

По данному графу $G$ и набору вершин $Q \subset V(G)$ в этой бакалаврской работе мы решаем задачу поиска сообщества, содержащего все или большинство вершин из $Q$. Задача является актуальной и применимой во многих областях, что мотивирует улучшать ее текущие решения.

В нашей работе мы пытаемся найти *k-core* с максимальным $k$ минимального размера, содержащий большинство вершин из $Q$. Мы оговариваем, что эта задача NP-полная и что для ее решения требуется эвристическое решение. Мы предлагаем решение этой задачи, основанное на идее алгоритма Barbieri et al. [8] о поиске ответа в компоненте максимального *k-core*, и предлагаем несколько эвристик, оптимизирующих это решение.

Результаты экспериментов, проведенных на реальных данных, показывают, что на запросах без шума предложенное решение работает примерно так же, как Barbieri et al. [8], однако с добавлением шума решение авторов этой статьи начинает находить неоптимальные подграфы, в то время как наше решение продолжает находить плотные оптимальные подграфы.

Также предложенный алгоритм поддерживает возможность обратной совместимости — мы можем ввести параметр $minSize$, который будет означать минимальное количество вершин-запросов, которое надо включить в ответ, от этого наш алгоритм практически не изменится. Тем самым, мы также можем точно так же решать задачу поиска поиска подграфа, содержащего все вершины-запросы.

В будущем планируется распространить предложенную идею на взвешенные подграфы, а также на другие типы графов — динамические графы, мультиграфы и т.д.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1 *Newman M.* Fast algorithm for detecting community structure in networks. — 2004.

2 *Newman M.* Finding community structure in networks using the eigenvectors of matrices // Phys. Rev. E 74. — 2006. — Т. 74.

3 *Fortunato S.* Community detection in graphs. — 2010.

4 Online search of overlapping communities / W. Cui [и др.] // Proceeding SIGMOD '13 Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. — 2013. — Т. 978. — С. 277–288.

5 *Faloutsos C.*, *Tong H.* Center-Piece Subgraphs: Problem Definition and Fast Solutions // Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and Data Mining. — 2006. — Т. 1. — С. 404–413.

6 The Minimum Wiener Connector Problem / N. Ruchansky [и др.] // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. — 2015. — Т. 1. — С. 1587–1602.

7 Approximate closest community search in networks / X. Huang [и др.] // Proceedings of the VLDB Endowment. — 2015. — Т. 9. — С. 276–287.

8 Efficient and effective community search / N. Barbieri [и др.] // Data Mining and Knowledge Discovery. — 2015. — Т. 29. — С. 1406–1433.

9 Mining Connection Pathways for Marked Nodes in Large Graphs / L. Akoglu [и др.]. — 2013.

10 *Gionis A.*, *Mathioudakis M.*, *Ukkonen A.* Bump hunting in the dark: Local discrepancy maximization on graphs // 2015 IEEE 31st International Conference on Data Engineering (ICDE). — 2015. — Т. 978. — С. 1155–1166.

11 *Faloutsos C.*, *McCurley K. S.*, *Tomkins A.* Fast discovery of connection subgraphs // Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. — 2004. — Т. 1. — С. 118–127.

12 *Sozio M.*, *Gionis A.* The community-search problem and how to plan a successful cocktail party // Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. — 2010. — Т. 978. — С. 939–948.

13 *Zhu L.*, *Ng W. K.*, *Cheng J.* Structure and attribute index for approximate graph matching in large graphs // Journal Information Systems. — 2011. — Т. 36. — С. 958–972.

14 Robust local community detection: on free rider effect and its elimination / Y. Wu [и др.] // Journal Proceedings of the VLDB Endowment. — 2015. — Т. 8. — С. 798–809.

15 Local search of communities in large graphs / W. Cui [и др.] // Proceeding SIGMOD '14 Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. — 2014. — Т. 978. — С. 991–1002.

16 As Strong as the Weakest Link: Mining Diverse Cliques in Weighted Graphs / P. Bogdanov [и др.] // ECML PKDD 2013 Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases. — 2013. — Т. 8188. — С. 525–540.

17 *Kou L.*, *Markowsky G.*, *Berman L.* A fast algorithm for Steiner trees // Journal Acta Informatica. — 1981. — Т. 15. — С. 141–145.