Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования

Портной Александр Михайлович

Исследование и разработка методов вычисления распределения подграфов в графе

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
ассистент кафедры СП
Гомзин Андрей Геннадьевич

Научный консультант:
Дробышевский Михаил Дмитриевич

Москва, 2018

Аннотация

Исследование и разработка методов вычисления
распределения подграфов в графе

Портной Александр Михайлович

В данной работе исследуются и сравниваются существующие методы подсчета распределения подграфов в графе, даются рекомендации по их применению, а также предлагается усовершенствования метода для эффективного подсчета распределения подграфов размера 3 и 4. Проводится экспериментальное сравнение разработанного метода с существующими решениями на графах разной величины. Разработанный алгоритм работает быстрее на 10-20 % существующих при поиске небольших подграфов в больших графах (более 10^6 вершин).

Содержание

Введение

Распределение подграфов в графе часто используется в сложном анализе графов. Частота подграфов может использоваться для определения значимых и незначимых характеристик в социальных, биологических, химических и технологических графах, для исследования свойств генов и белков при поиске новых лекарств [1, 2]. Данная задача является подзадачей более обширной проблемы поиска мотивов - статистически значимых подграфов [3]. Подсчет можно осуществлять двумя разными способами:

- генерация неизоморфных подграфов размера k, и подсчет каждого из них по-отдельности
- определение класса подграфа после перечисления, то есть для каждого найденного подграфа его класс определяется отдельно

Задача подсчета подграфов является вычислительно сложной, так как включает в себя задачу проверки графов на изоморфность, для которой неизвестно полиномиального решения. С ростом размера входного графа n, количество подграфов заданного размера k растет экспоненциально. Количество неизоморфных подграфов также растет экспоненциально в зависимости от заданного k. В связи с этим при росте размера подграфа или размера входного графа необходимое время вычисления увеличивается.

Далее рассмотрены существующие методы для подсчета распределения подграфов, показаны их основные идеи, ограничения, достоинства и недостатки, а также проведено их сравнение на разных графах. На основе полученных результатов предложен новый метод, основанный на существующих идеях и специфике поставленной задачи.

1 Постановка задачи

В рамках выпускной квалификационной работы необходимо исследовать и разработать существующие методы вычисления распределения подграфов в графе. При этом требуется, чтобы разработанный метод был ориентирован на подсчет подграфов размера 3 и 4 в больших ориентированных графах ($|V| > 10^6$). Для этого необходимо:

- Провести обзор существующих методов подсчета подграфов в графе.
- Сравнить и выявить недостатки существующих методов в условиях поставленной задачи.
- Разработать метод для подсчета подграфов размера 3, 4 в графах большого размера.
- Провести экспериментальное сравнение с существующими методами.

2 Обзор существующих решений

В данной главе сначала приводятся необходимые определения, а затем описаны различные алгоритмы решения каждой из подзадач: задачи обхода графа и подсчета неизоморфных графов, а также кратко описаны существующие методы вычисления распределения подграфов.

2.1 Определения

Граф (ориентированный) G – это упорядоченная пара $G := (V, E)$, где V – это множество вершин, а E – множество (упорядоченных) пар вершин, называемых ребрами. $V(G)$ обозначает множество вершин графа, $E(G)$ – множество ребер. Размером подграфа обозначим количество вершин в графе.

Вершина v' смежна с вершиной v, если они соединены ребром. Для множества вершин $V' \subseteq V$ открытой окрестностью $N(V')$ называется множество всех вершин из $V - V'$, которые смежны с хотя бы одной вершиной из V'. Для вершины $v \in V - V'$ ее исключительной окрестностью относительно V' называется множество всех соседних с v вершин, не принадлежащих $V' \cup N(V')$ и обозначается $N_{excl}(v, V')$.

Ребра e и e' смежные, если у них есть общая вершина. Для множества вершин $V' \subseteq V$ ребро (v_1, v_2) является соседним, если либо $v_1 \in V'$ и $v_2 \notin V'$, либо $v_2 \in V'$ и $v_1 \notin V'$.

Граф H является подграфом графа G, если $V(H) \subseteq V(G)$ и $E(H) \subseteq E(G)$. Индуцированный подграф $G[N]$ - это подграф графа G, содержащий множество вершин N и все ребра между вершинами из N, которые входят в $E(G)$. В задаче подсчета подграфов считаются индуцированные подграфы.

Два графа G_1 и G_2 изоморфны, если существуют биекции $\phi : V(G_1) \leftrightarrow V(G_2)$ и $\theta : E(G_1) \leftrightarrow E(G_2)$ такие, что $(v_i, v_j) \in E(G_1) \Leftrightarrow \theta(v_i, v_j) = (\phi(v_i), \phi(v_j)) \in E(G_2)$

2.2 Изоморфизм графов

Для подсчета неизоморфных подграфов необходим механизм для определения изоморфных графов. Один из наиболее эффективных методов - это канонизация графа, которая заключается в сопоставлении графу его канонической метки. Канонические метки графов равны тогда и только тогда, когда графы изоморфны [4]. Существует несколько реализованных алгоритмов для нахождения канонической метки, например

nauty[4], *traces*[5], *bliss*[6] или *saucy*[7]. Однако, у всех этих решений в худшем случае сложность $O(s!)$, где s – размер графа, и на текущий момент неизвестно полиномиального решения для данной задачи.

Большинство методов подсчета используют алгоритм *nauty* для канонизации графа. Для получения канонической метки данный метод представляет матрицу смежности графа, которая состоит из 0 и 1, в виде двоичного числа, путем конкатенации строк матрицы. К первой строке матрицы справа приписывается вторая, затем третья и так далее, пока все строки не будут записаны в одну. Получившуюся строку далее рассматривают в виде двоичного числа. Затем *nauty* перебирает перестановки вершин, и находит такую, при которой метка минимальна. Метка для соответствующей перестановки и является канонической для исходного графа.

2.3 Обход графов

2.3.1 Edge sampling algorithm

Алгоритм приближенного подсчета подграфов размера k. Сначала выбирается случайное ребро, и подграф расширяется, путем последовательного добавления случайного соседнего ребра, пока в подграфе не будет k вершин. Для выбора случайного ребра поддерживается список текущих соседних ребер, из которого алгоритм выбирает случайное ребро для расширения подграфа. Данный алгоритм не является однородным, то есть вероятности получить различные подграфы различаются и зависят от количества ребер в подграфе. Для исправления данного недостатка алгоритм подсчитывает концентрации подграфов C_i для определения их значимости [8]. Концентрация $C_i = \frac{S_i}{\sum\limits_{k=1}^{L} S_k}$, где S_i – количество подграфов типа i, L – количество типов подграфов.

2.3.2 ESU

Алгоритм обхода графа ESU (Enumerate Subgraphs) [9] предполагает, что все вершины помечены метками - целыми числами. Для каждой вершины v исходного графа G алгоритм выполняет обход, добавляя в множество $V_{extention}$ по одной вершине, для которых выполняется: их метки больше, чем метка вершины v и они смежны с последней добавленной в $V_{subgraph}$ вершиной и не смежны ни с какой вершиной из $V_{extention}$. Подграф $V_{subgraph}$ расширяется, пока не достигнет заранее заданного размера k. Каждый

подграф алгоритм находит один раз.

На основе данного алгоритма существует алгоритм RAND-ESU для приближенного подсчета подграфов. При работе алгоритма ESU образуется дерево обхода, на каждом уровне которого в текущий подграф $V_{subgraph}$ входит одинаковое количество вершин. Для каждого уровня задается вероятность, с которой продолжается построение подграфа, используя текущую вершину из $V_{extention}$. Таким образом все подграфы могут быть найдены с одинаковой вероятностью.

2.3.3 ESE

Алгоритм обхода графа ESE (Edge-based Subgraph Enumeration) [10] очень похож на алгоритм ESU. Основное отличие этих алгоритмов в том, что обход в ESE запускается для каждого ребра, а не для каждой вершины, как в алгоритме ESU. Данное изменение позволяет более сбалансированно распараллеливать вычисления, то есть средняя разница во времени работы потоков вычисления уменьшается.

2.4 Структура данных G-Tries

G-Tries[3] - структура данных для хранения коллекции графов. Данная структура концептуально похожа на префиксное дерево и хранит подграфы в соответствии с их структурами, где предок для своих потомков является подграфом. Каждый узел дерева содержит информацию об одной вершине графа и ребрах, связывающих ее с узлами-предками. Путь от корня до листа соответствует одному графу. Во время построения дерева графы, матрицы которых приведены к каноническому виду, вставляются по одному. В качестве канонической матрицы используется лексикографически максимальная матрица смежности для уменьшения размера дерева.

2.5 Классификация алгоритмов

Алгоритмы подсчета распределения подграфов в графе можно разделить на 2 категории: network-centric и motif-centric алгоритмы.

Network-centric алгоритмы совершают обход графа и определяют распределение подграфов заданного размера k с помощью проверки подграфов на изоморфность.

Motif-centric алгоритмы подсчитывают для каждого заданного подграфа, сколько раз он встречается в исходном графе. Для подсчета распределения всех подграфов перед обходом генерируются всевозможные подграфы заданного размера k.

Также некоторые алгоритмы могут производить как точный, так и приближенный подсчет распределения подграфов.

Таблица 1: Характеристики алгоритмов вычисления распределения подграфов

Алгоритм	Год	Прибл. вычисл.	Параллел. вычисл.
mfinder	2002	+	-
Fanmod	2005	+	-
Grochow-Kellis	2007	-	-
Kavosh	2009	-	-
MODA	2009	+	-
gtrieScanner	2010	-	+
NetMODE	2012	-	-
QuateXelero	2013	-	-
FaSE	2013	+	-
parallel FaSE	2013	-	+
Subenum	2015	-	+

2.5.1 Network-centric алгоритмы

mfinder [8]: один из первых инструментов для подсчета подграфов, который реализует, как точный, так и приближенный подсчет. Данный алгоритм основан на алгоритме обхода графа edge sampling algorithm и не подходит для подсчета подграфов большого размера.

Fanmod [9]: алгоритм, основанный на алгоритме поиска подграфов ESU и RAND-ESU и использующий утилиту *nauty* для канонизации подграфов. Алгоритм способен производить как точный, так и приближенный подсчет.

Kavosh [11]: алгоритм, обходящий граф следующим образом. Для каждой вершины графа v рассматриваются всевозможные разложения $k-1$ на слагаемые $k_1+...+k_i+...+k_m$, которые обозначают кол-во вершин берущихся из окрестности порядка i вершины

v, причем рассматриваются только те вершины, метки которых больше, чем у v. Для каждого найденного подграфа с помощью утилиты nauty ищется каноническая метка и считается общее количество подграфов. Для быстрого перебора всевозможных вершин в окрестности определенного порядка используется алгоритм resolving door [12].

gtrieScanner [3]: первый алгоритм, использующий g-tries для подсчета подграфов. Спускаясь по gtrie, алгоритм сопостовляет узлам gtrie подходящие вершины графа, сравнивая соответствующую строку матрицы смежности для вершины графа и информации о ребрах в узле gtrie. В ходе построения gtrie и поиска подходящих вершин применяется алгоритм Symmetry breaking conditions [3] для генерации и применения условий для отсечения одинаковых подграфов.

NetMODE [13]: алгоритм, работающий для подграфов, размер которых не больше 6 и основанный на алгоритме обхода графа Kavosh. В данном алгоритме применена следующая идея: канонические метки для всех подграфов заданного размера заранее вычисляются и помещаются в оперативную память, что позволяет не тратить время на нахождение метки во время обхода. В отличие от алгоритма Kavosh авторы не используют алгоритм resolving door[12].

QuateXelero [14]: алгоритм, использующий метод обхода графа ESU и одновременное построение Quaternary Tree, с помощью которого уменьшается количество вызовов алгоритма вычисления канонической метки графа *nauty*. Quaternary Tree представляет собой четверичное дерево, где каждое из ребер дерева соответствует одному из 4 вариантов направления ребер между новой вершиной и одним из предков: вершины не связаны, вершины связаны одним направленным ребром (2 варианта) или двумя противоположно направленными ребрами. В отличие от g-trie каждому подграфу размера k в дереве соответствует путь от корня к листу длиной $k * (k-1)/2$

FaSE [15]: алгоритм, основанный на алгоритме обхода графа ESU и немного измененной структурой хранения графов g-trie. G-Trie изменен следующим образом: перед вставлением графов в g-trie не производится их канонизация, что приводит к более "широкому"дереву, то есть в листьях дерева могут находится изоморфные графы. При обходе графа полученный подграф вставляется в g-trie, не изменяясь. После обхода всего графа для каждого листа дерева g-trie находится его каноническая форма, и производится итоговый подсчет подграфов. Заметим, что количество вызовов функции получения канонической метки графа значительно уменьшилось по сравнению с

Fanmod, так как данная функция вызывается не для всех графов, а только различающихся матрицами смежности. Так как данный алгоритм основан на алгоритме ESU, то он также может осуществлять приближенный подсчет.

Subenum [10]: алгоритм, основанный на алгоритме обхода графа ESE (Edge-based Subgraph Enumeration). Также в данном алгоритме применяется следующая эвристика: вместо вычисления канонической метки, вершины подграфа сортируются по степени и переименовываются в соответствии с порядком в отсортированном массиве. После чего каждой полученной метке сопоставляется каноническая метка. Таким образом алгоритм позволяет уменьшить количество вызовов алгоритма подсчета канонических меток *nauty*.

Таблица 2: Network-centric алгоритмы вычисления распределения подграфов

Алгоритм	Обход графа	Структура данных	Каноническая метка
mfinder	Edge sampling	-	-
Fanmod	ESU	-	nauty для каждого подграфа
Kavosh	Kavosh	-	nauty для каждого подграфа
gtrieScanner	$\sim$ ESU	g-trie	-
NetMODE	Kavosh	-	предподсчет в память
QuateXelero	ESU	quaternary-tree	nauty для листьев
FaSE	ESU	g-trie "широкий"	nauty для листьев
Subenum	ESE	-	Ordering labeling + nauty

2.5.2 Motif-centric алгоритмы

Grochow-Kellis [16]: алгоритм, разработанный для подсчета подграфов размера больше 8 и использующий *nauty*, subgraph symmetries и элементы хэширования. Он отображает заданный шаблон в граф всеми возможными способами для проверки графов на изоморфность. Данный алгоритм может быть распараллелен.

MODA [17]: алгоритм, разработанный для более эффективного подсчета подграфов большого размера и использующий информацию на основе ранее найденных графов запросов. Сохранение информации о ранее найденных сопоставлениях помогает сократить время вычислений. Кроме того, MODA может производить приближенный

подсчет, который может быть использован для уменьшения времени выполнения.

2.6 Параллельный вариант алгоритма FaSE

В алгоритме обхода ESU подсчет запускается от каждой вершины, и, таким образом, состоит из независимых веток вычисления, что позволяет параллельно запускать подсчет по вершинам графа. Но данный подход сильно несбалансирован, вычисления для одной из вершин могут занимать до 40% времени. В связи с этим авторами алгоритма FaSE был предложен сбалансированный метод динамического распределения работы между потоками [18].

2.6.1 Parallel FaSE Algorithm

```
1: procedure ParallelFaSE(G, T, k)
2:     T ← ∅
3:     W ← ∅
4:     i, j ← thread_id
5:     while i ≤ |V(G)| do
6:         v ← V(G)_i
7:         if workRequest(P) then
8:             W.addWork()
9:             (W_Q, W_P) ← splitWork(W)
10:            giveWork(W_P, P)
11:            resumeWork(W_Q)
12:        enumerate({v}, {u ∈ N(v) : u > v}, T.root)
13:        i ← i + num_threads
14:    while j ≤ |T.leaves()| do
15:        l ← T.leaves()_j
16:        frequency[canonicalLabel(l.Graph)] += l.count
17:        j ← j + num_threads
```

Рис. 1: Алгоритм параллельного подсчета FaSE

На рис.1 показан параллельный алгоритм FaSE. Вычисление начинается с первоначально пустой gtrie T и очередей работы W для каждого потока. Каждому потоку выделяется $|V(G)|/num_{threads}$ вершин для подсчета методом $round - robin$. Такое разделение

не обязательно является сбалансированным, но нахождение наилучшего разделения достаточно трудозатратно. Когда поток Q получает запрос работы от P, ему нужно остановить свое вычисление, добавить оставшуюся работу в W, разделить работу, дать половину P и возобновить вычисление. После завершения фазы подсчета листья также распределяются между нитями, а вычисление канонических меток для каждого случая выполняется параллельно.

2.6.2 Запрос работы

Когда поток P завершил назначенную ему работу, он запрашивает работу у случайного потока Q. Случайный опрос был установлен как эффективная эвристика для динамической балансировки нагрузки [19], и, кроме того, в нашем случае сложно точно предсказать, сколько вычислений осталось у потока. Если Q делится с P работой, тогда P вычисляет данную ему работу. Если у Q не было работы для обмена, P пытается опросить другой случайный поток. Когда все потоки пытаются получить работу, и все вычисления произведены, подсчет заканчивается.

2.6.3 Разделение работы

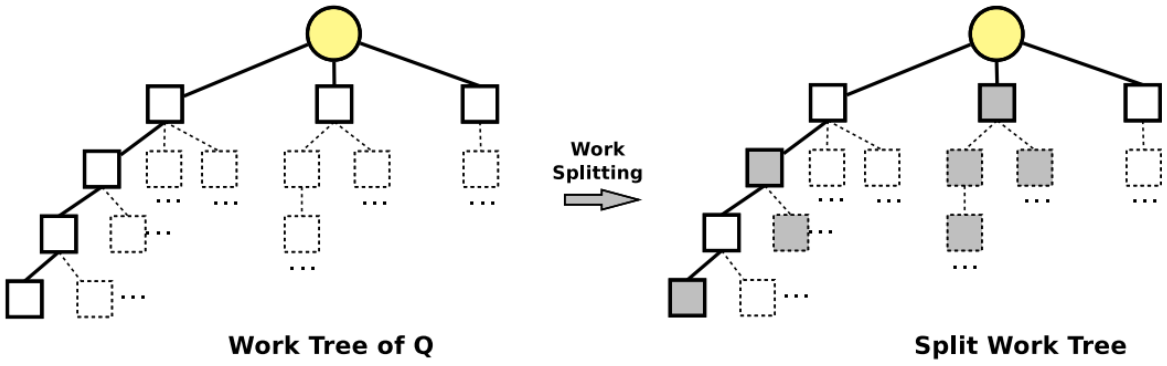

Рис. 2: Дерево выполнения для потока Q, и его разделение после получения запроса работы

Когда поток получает запрос на работу, выполнение подсчета приостанавливается и выполняется разделение работы. На рис. 2 показано дерево выполнения потока Q и его разделение с потоком P. Дерево работы создается рекурсивными вызовами $addWork()$.

Во время разделения один уровень работ выделяются одному потоку, кроме 0-го уровня, где работа выделяется попеременно. В данном примере Q получает уровень 3 и 1, а P получает 2 и 4. Самый верхний уровень полностью разделен.

2.6.4 Продолжение подсчета

```
 1: procedure RESUMEWORK(W)
 2:     ORDERBYLOWEST(W)
 3:     for all level L of W do
 4:         if WORKREQUEST(P) then
 5:             (W_Q, W_P) ← SPLITWORK(W)
 6:             GIVEWORK(W_P, P)
 7:             RESUMEWORK(W_Q)
 8:             return
 9:         if L.depth = 0 then
10:             for all vertex v of L.V_ext do
11:                 ENUMERATE({v}, {u ∈ N(v) : u > v}, T.root)
12:         else
13:             ENUMERATE(L.V_s, L.V_ext, L.current)
14:     ASKFORWORK()
```

Рис. 3: Алгоритм продолжения подсчета после разделения работы

Уровни работ упорядочиваются сверху вниз. Если получен запрос на работу, то выполняется разделение работы. Если вычисляемый уровень является верхним, то вершины вычисляются индивидуально, иначе сохраненные значения используется для продолжения ранее остановленного вычисления. Если поток заканчивает свою работу, он отправляет запрос работы.

3 Исследование и построение решения задачи

В данной главе описывается разработанный метод подсчета распределения подграфов в графе, этапы построения решения и выводы. При конструировании метода были использованы идеи, позаимствованные из уже существующих методов, которые подходят под специфику поставленной задачи. Изначально было разработано базовое решение, после чего в него были добавлены возможность приближенного и параллельного подсчета.

В силу полученных результатов тестирования существующих алгоритмов было решено взять за основу метод подсчета FaSE, который работает достаточно быстро на больших графах, и улучшить его, исходя из специфики поставленной задачи. Далее описываются 3 ключевые составляющие решения: обход графа, канонизация графа и внутреннее представление графа.

3.1 Обход графа

Был выбран метод обхода графа Rand-ESU [9], который позволяет также осуществлять обход не всего графа, а только его части с учетом заданных вероятностей. Основное достоинство данного приближенного подхода заключается в том, что он является сбалансированным, то есть вероятность найти каждый подграф одинаковая.

Также для данного метода обхода существует динамический способ разделения вычислений, предложенный авторами алгоритма FaSE, что позволяет достаточно хорошо распараллеливать подсчет.

3.2 Канонизация графа

Идея избежать многократного подсчета канонических меток взята из алгоритма NetMODE [13], авторы которого предложили совершить предварительный подсчет канонических меток всех подграфов и поместить их в память. В поставленной задаче требуется искать подграфы маленького размера, а значит сопоставление меток подграфам займет мало места (менее 1 Мб) и поместится в оперативную память.

3.3 Внутреннее представление графа

Существуют следующие способы хранить графы:

- Матрица смежности - граф хранится в виде матрицы размера $|V|^2$, где на пересечении строк i и j стоит 1, если в E входит ребро (i, j), и 0 иначе.
- Список смежности - граф хранится в виде списка списков, где для каждой вершины хранятся список номеров вершин, из которых приходят ребра, и список вершин, в которые ребра выходят из данной.
- Список ребер - граф хранится в виде списка ребер.

Для больших графов не подходит матрица смежности, так как она потребляет много памяти. Список ребер также не подходит, так как нам необходимо часто определять связанные с текущей вершины, а в списке ребер данная операция не выполняется за $O(1)$. Таким образом, для поставленной задачи предпочтительно использовать список смежности.

3.4 Выводы

Для построения решения принималось во внимание то, что разработанный метод должен работать на больших графах, а значит быть эффективен по потреблению ЦП и памяти.Также алгоритм должен использоваться для подсчета небольших подграфов, что позволяет использовать в решении методы, неприменимые для больших подграфов.

В разработанном методе используются следующие идеи и алгоритмы:

- Алгоритм обхода графа Rand-ESU [9].
- Внутреннее представление графа в виде списка смежности.
- Идея предподсчета канонических меток из алгоритма NetMODE [13].
- Метод распараллеливания из параллельной реализации алгоритма FaSE [18].

Стоит отметить, что в полученном методе есть возможность параллельного приближенного подсчета распределения подграфов, что позволяет быстро получить приблизительный ответ для больших графов. На данный момент нет методов, предоставляющих такую возможность.

4 Описание практической части

Выбор алгоритмов для тестирования осуществлялся на основе доступности исходного кода и результатов последних обзоров [20, 21], исходя из чего в тестирование попало 8 network-centric алгоритмов. Все алгоритмы, кроме Fanmod, тестировались на ОС Ubuntu. Fanmod тестировался под ОС Windows. Оцениваемые параметры: время исполнения и переполнение памяти. В конце пункта 4.4 будут даны рекомендации по использованию протестированных инструментов.

4.1 Данные для тестирования

Проводилось тестирование поиска подграфов размера 3 и 4 на следующих ориентированных графах [1]:

Таблица 3: Графы для тестирования

Название	Домен	Кол-во вершин	Кол-во ребер	Особенности
DNC emails	Communication	2029	39264	есть кратные ребра
Cora citation	Citation	23166	91500	
FOLDOC	Hyperlink	13356	125207	
Gnutella	Computer	62586	147892	
Google+	Social	23628	39242	
Amazon (TWEB)	Misc	403394	3387388	
Digg friends	Social	279630	1731653	
Pokec	Social	1632803	30622564	
TREC WT10g	Hyperlink	1601787	8063026	
Berkeley/Stanford	Hyperlink	685230	7600595	

На каждом графе алгоритм тестировался от 1 до 5 раз в зависимости от времени работы T. При $T < 10$сек 5 раз, при $t < 400$сек 3 раза, иначе - 1 раз

4.2 Характеристики оборудования

Тестирование проводилось на ноутбуке со следующими характеристиками:

[1] http://konect.uni-koblenz.de/networks/

- Процессор — 8 core Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz.
- Оперативная память — 8Gb, 1600MHz.
- Система —Ubuntu 14.04, Windows 7 (для Fanmod).

4.3 Результаты тестирования

4.3.1 Полный подсчет

Таблица 4: Время работы алгоритмов на графах в сек. (-t – время работы дольше наблюдаемого (1 часа), -m – переполнение по памяти, - – ошибка при чтении; в скобках указано количество вершин и ребер в графе)

	размер подграфа	mfinder	Fanmod	Kavosh	gtrieScanner	FaSE	Subenum	QuateXelero	NetMODE	New algorithm
DNC emails	3	-	0,279	1,122	0,014	0,015	0,172	**0,007**	0,019	0,009
(2029,39264)	4	-	40,8	20,3	1,07	1,35	2,38	**0,62**	0,9	0,85
Cora citation	3	146,8	1,276	1,050	0,169	0,195	1,394	**0,086**	0,16	0,16
(23166,91500)	4	6840	79	71	3,6	5,7	16	**2,6**	3,4	4,7
FOLDOC	3	99	2,71	1,42	0,18	0,2	0,78	**0,08**	0,14	0,14
(13356,125207)	4	16704	596	201	9	17,1	28,8	**6,1**	8,5	13,3
Gnutella	3	184,8	1,01	0,98	0,19	0,23	4,2	**0,11**	0,26	0,16
(62586,147892)	4	5040	17	18,2	1,9	3,6	17,5	**1,7**	2,15	2,7
Google+	3	236	9,3	7,9	0,37	0,66	2,79	**0,25**	0,42	0,55
(23628,39242)	4	-t	-t	6030	205	318	828	**125**	201	258
Amazon (TWEB)	3	-t	41	-m	-m	6,8	662	-m	-m	**5,8**
(403394,3387388)	4	-t	-t	-m	-m	1123	-t	-m	-m	**911**
Digg friends	3	-t	1628	-m	-m	81	339	-m	-m	**68**
(279630,1731653)	4	-t	-t	-m	-m	-t	-t	-m	-m	-t
Pokec	3	-t	3592	-m	-m	369	6013	-m	-m	**329**
(1632803,30622564)	4	-t	-t	-m	-m	-t	-t	-m	-m	-t
TREC WT10g	3	-t	-t	-m	-m	295	7097	-m	-m	**233**
(1601787,8063026)	4	-t	-t	-m	-m	-t	-t	-m	-m	-t
Berkeley/Stanford	3	-t	-t	-m	-m	1895	-m	-m	-m	**1708**
(685230,7600595)	4	-t	-t	-m	-m	-t	-m	-m	-m	-t

4.3.2 Параллельный подсчет

Далее на графиках показаны ускорения времени работы алгоритмов в зависимости от количества используемых потоков. Тестирование проводилась на графах Cora citation и Gnutella для подграфов размера 3 и 4.

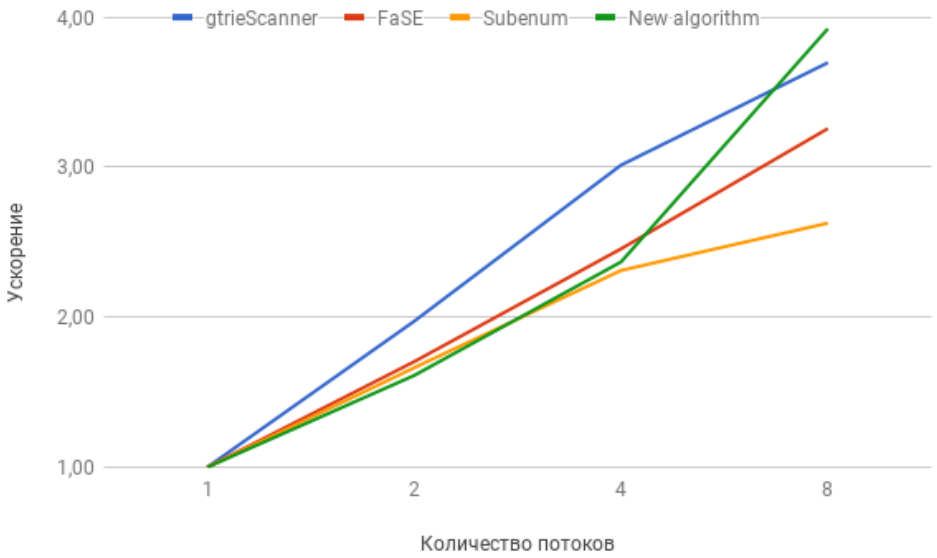

Рис. 4: Ускорения алгоритмов в графе Cora citation для подграфов размера 3

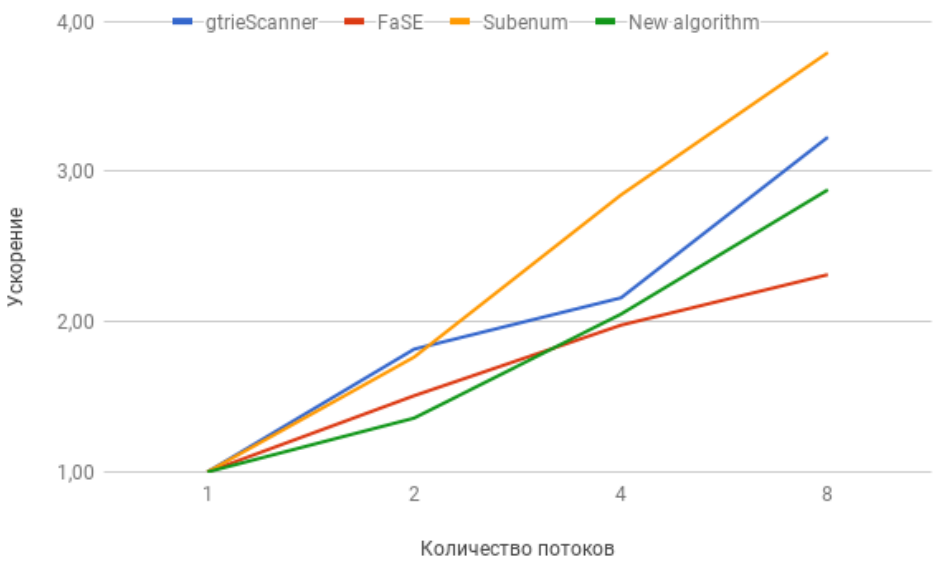

Рис. 5: Ускорения алгоритмов в графе Cora citation для подграфов размера 4

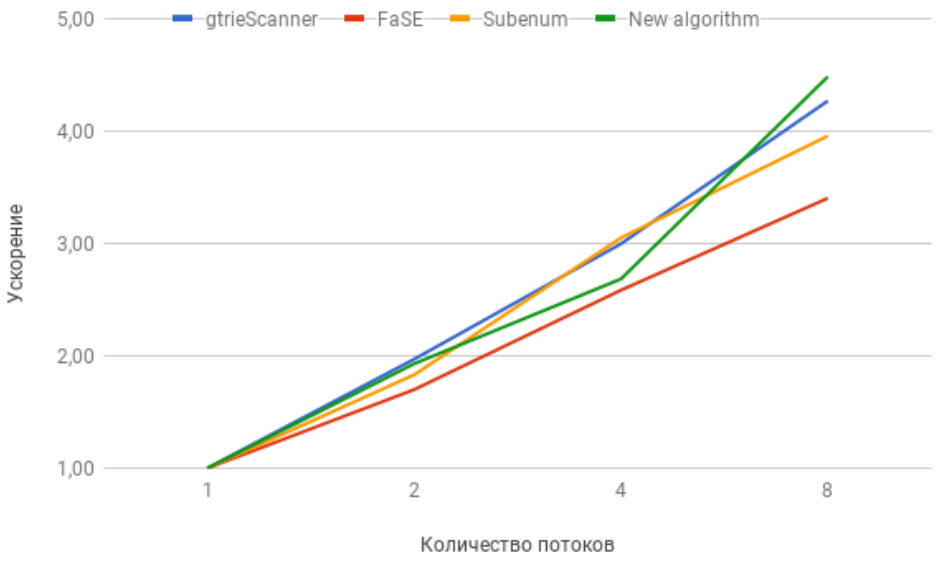

Рис. 6: Ускорения алгоритмов в графе Gnutella для подграфов размера 3

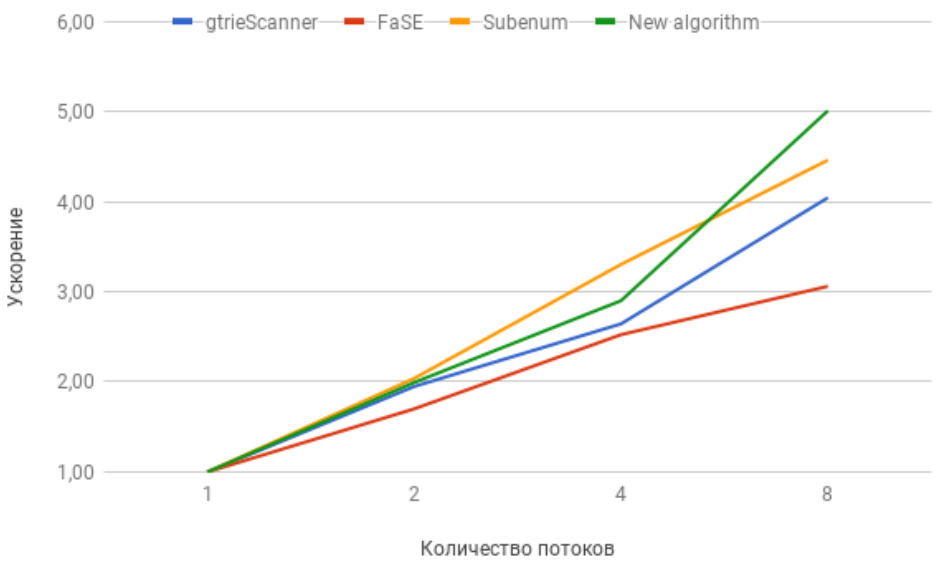

Рис. 7: Ускорения алгоритмов в графе Gnutella для подграфов размера 4

4.4 Результаты и рекомендации для существующих алгоритмов

Результаты тестирования в табл. 4 показывают, что для точного подсчета на небольших графах следует использовать алгоритмы QuateXelero, gtrieScanner и FaSE, причем на всех протестированных графах QuateXelero работает быстрее. При этом у алгоритмов gtrieScanner и FaSE есть параллельные версии, с помощью которых можно уменьшить время выполнения в несколько раз. Алгоритм FaSE позволяет посчитать распределение подграфов в больших графах. Основываясь на этих данных, можно сделать вывод, что при наличии многоядерного компьютера на небольших сетях эффективнее использовать алгоритм gtrieScanner, а в тех случаях, когда происходит переполнение по памяти - алгоритм FaSE. При использовании для подсчета одного ядра на небольших сетях эффективнее показал себя алгоритм QuateXelero.

Для алгоритма Fanmod реализован удобный пользовательский интерфейс, что позволяет пользователям ПК использовать данный алгоритм без особых проблем, причем интерфейс запускается только из-под операционной системы Windows. Для метода FaSE опубликован неокончательный код, из-за чего данный алгоритм некорректно обрабатывает сети с циклами и кратными ребрами.

Резюмируя вышесказанное, для большинства задач подойдут методы gtrieScanner и

QuateXelero, но когда они неприменимы из-за нехватки памяти, следует использовать метод FaSE.

4.5 Результаты тестирования разработанного метода

Результаты тестирования показывают, что новый метод работает быстрее, чем алгоритм FaSE, который был взят за основу. На маленьких графах разработанный метод уступает QuateXelero и работает примерно также, как и алгоритмы gtrieScanner и NetMODE, но надо учитывать то, что при этом данные методы не работают на больших графах.

Результаты сравнения параллельных версий алгоритмов показали, что разработанный метод показывает чуть лучшее ускорение, чем алгоритм FaSE.

В итоге был получен алгоритм, работающий лучше остальных на больших графах, способный осуществлять как точный, так и приближенный подсчет, а также одновременно производить приближенные вычисления параллельно.

Заключение

По итогам данной работы было проведено исследование существующих методов вычисления распределения подграфов в графе и разработан собственный метод. Были выполнены следующие подзадачи:

1. Исследованы существующие методы вычисления распределения подграфов в графе, и даны рекомендации по использованию методов.

2. Реализован собственный метод вычисления распределения подграфов, и проведено сравнение методов подсчета подграфов размера 3 и 4 на графах разной величины.

Разработанный метод работает быстрее на 10-20% существующих алгоритмов для маленьких подграфов (размера $3, 4$) на графах большого размера ($|V| > 10^6$), тем самым выполнил поставленную задачу. На небольших графах есть методы, работающие быстрее, однако это выходит за рамки поставленной задачи. Разработанный метод также позволяет осуществлять параллельный приближенный подсчет, чего не было до этого реализовано в других методах.

Список литературы

[1] *Kim, Wooyoung.* Network motif detection: Algorithms, parallel and cloud computing, and related tools / Wooyoung Kim, Martin Diko, Keith Rawson // *Tsinghua Science and Technology.* — 2013. — Vol. 18, no. 5. — Pp. 469–489.

[2] *Zabet, Nicolae Radu.* Negative feedback and physical limits of genes / Nicolae Radu Zabet // *Journal of theoretical biology.* — 2011. — Vol. 284, no. 1. — Pp. 82–91.

[3] *Ribeiro, Pedro.* G-tries: an efficient data structure for discovering network motifs / Pedro Ribeiro, Fernando Silva // Proceedings of the 2010 ACM Symposium on Applied Computing / ACM. — 2010. — Pp. 1559–1566.

[4] *Babai, László.* Canonical labeling of graphs / László Babai, Eugene M Luks // Proceedings of the fifteenth annual ACM symposium on Theory of computing / ACM. — 1983. — Pp. 171–183.

[5] *Katebi, Hadi.* Conflict anticipation in the search for graph automorphisms / Hadi Katebi, Karem A Sakallah, Igor L Markov // International Conference on Logic for Programming Artificial Intelligence and Reasoning / Springer. — 2012. — Pp. 243–257.

[6] *Junttila, Tommi.* Engineering an efficient canonical labeling tool for large and sparse graphs / Tommi Junttila, Petteri Kaski // 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX) / SIAM. — 2007. — Pp. 135–149.

[7] *Darga, Paul T.* Faster symmetry discovery using sparsity of symmetries / Paul T Darga, Karem A Sakallah, Igor L Markov // Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE / IEEE. — 2008. — Pp. 149–154.

[8] Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs / Nadav Kashtan, Shalev Itzkovitz, Ron Milo, Uri Alon // *Bioinformatics.* — 2004. — Vol. 20, no. 11. — Pp. 1746–1758.

[9] *Wernicke, Sebastian.* Efficient detection of network motifs / Sebastian Wernicke // *IEEE/ACM Transactions on Computational Biology and Bioinformatics.* — 2006. — Vol. 3, no. 4.

[10] *Shahrivari, Saeed.* Fast parallel all-subgraph enumeration using multicore machines / Saeed Shahrivari, Saeed Jalili // *Scientific Programming.* — 2015. — Vol. 2015. — P. 6.

[11] Kavosh: a new algorithm for finding network motifs / Zahra Razaghi Moghadam Kashani, Hayedeh Ahrabian, Elahe Elahi et al. // *BMC bioinformatics.* — 2009. — Vol. 10, no. 1. — P. 318.

[12] *Kreher, Donald L.* Combinatorial algorithms: generation, enumeration, and search / Donald L Kreher, Douglas R Stinson. — CRC press, 1998. — Vol. 7.

[13] Netmode: Network motif detection without nauty / Xin Li, Douglas S Stones, Haidong Wang et al. // *PloS one.* — 2012. — Vol. 7, no. 12. — P. e50093.

[14] QuateXelero: an accelerated exact network motif detection algorithm / Sahand Khakabimamaghani, Iman Sharafuddin, Norbert Dichter et al. // *PloS one.* — 2013. — Vol. 8, no. 7. — P. e68073.

[15] *Paredes, Pedro.* Towards a faster network-centric subgraph census / Pedro Paredes, Pedro Ribeiro // Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on / IEEE. — 2013. — Pp. 264–271.

[16] *Grochow, Joshua A.* Network motif discovery using subgraph enumeration and symmetry-breaking / Joshua A Grochow, Manolis Kellis // Annual International Conference on Research in Computational Molecular Biology / Springer. — 2007. — Pp. 92–106.

[17] *Omidi, Saeed.* MODA: an efficient algorithm for network motif discovery in biological networks / Saeed Omidi, Falk Schreiber, Ali Masoudi-Nejad // *Genes & genetic systems.* — 2009. — Vol. 84, no. 5. — Pp. 385–395.

[18] *Aparicio, David.* A scalable parallel approach for subgraph census computation / David Aparicio, Pedro Paredes, Pedro Ribeiro // European Conference on Parallel Processing / Springer. — 2014. — Pp. 194–205.

[19] *Slota, George M.* Fast approximate subgraph counting and enumeration / George M Slota, Kamesh Madduri // Parallel Processing (ICPP), 2013 42nd International Conference on / IEEE. — 2013. — Pp. 210–219.

[20] Current innovations and future challenges of network motif detection / Ngoc Tam L Tran, Sominder Mohan, Zhuoqing Xu, Chun-Hsi Huang // *Briefings in bioinformatics*. — 2014. — Vol. 16, no. 3. — Pp. 497–525.

[21] A Review of Motif Discovery Algorithms as the Main Units of the Complex Networks / Mohammad Ali Salari, Jamshid Tashk, Hossein Bobarshad, Sahand Khakabimamaghani.

[22] *McKay, Brendan D.* Practical graph isomorphism / Brendan D McKay et al. — 1981.