Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра системного программирования

Дипломная работа

Исследование и разработка распределённых алгоритмов для поиска сообществ пользователей в социальных сетях

Выполнил:
студент 528 группы
Рябов Сергей Викторович

Научные руководители:
ак. РАН, проф. Иванников Виктор Петрович
Коршунов Антон Викторович

Москва 2012

Содержание

Аннотация 4

Введение 5

Сообщества в социальных сетях 6

1 Постановка задачи 9

2 Обзор существующих решений 10

2.1 Методы иерархической кластеризации 10

2.2 Максимизация целевой функции 11

2.3 Линейные графы 11

2.4 Локальные методы 12

2.4.1 Перколяция клик 12

2.4.2 Жадный алгоритм расширения клик 13

3 Исследование и построение решения задачи 15

3.1 Исследование методов 15

3.2 Модель MapReduce 16

3.3 Построение решения задачи 17

3.3.1 Алгоритм поиска максимальных клик в графе 17

3.3.2 Расширение клик 20

3.3.3 Отсев потенциальных дубликатов 20

3.3.4 Модель данных 21

4 Описание практической части 22

4.1 Обоснование выбранного инструментария 22

4.2 Общая схема работы 22

4.2.1 Конвертация входных данных 23

4.2.2 Поиск максимальных клик в графе 23

4.2.3 Расширение клик 24

4.2.4 Удаление потенциальных дубликатов 24

4.2.5 Конвертация результатов работы алгоритма 26

4.3 Тестирование 27

4.3.1 Тестовый стенд 27

4.3.2 Оценка эффективности 27
4.3.3 Тестирование на больших объёмах данных 27
4.3.4 Выводы 28

Заключение 30

Список литературы 31

Аннотация

В данной работе рассмотрена задача поиска сообществ пользователей в социальных сетях. Был реализован распределённый алгоритм поиска пересекающихся сообществ на базе платформы Apache Hadoop, а также проведён ряд испытаний с целью оценки эффективности и производительности решения.

Введение

Графы крайне удобны для представления широкого спектра систем в различных областях. Биологические, социальные, технологические и информационные сети могу быть представлены в качестве графов, и поэтому анализ графов стал ключевым механизмом для изучения свойств этих систем. Например, анализ социальных сетей берёт начало в 30х годах прошлого столетия, и к настоящему моменту стал одним из важнейших направлений в социологии [22].

Графовые представления реальных систем отличаются своей неоднородностью. В особых группах вершин концентрация дуг высока, в то время как между этим группами она невелика. Это свойство реальных сетей называется структурой сообществ [12] или кластеризацией. Сообщество, или кластер, – это набор вершин, относительно сильно связанных друг с другом, и, возможно, обладающих общими свойствами и/или играющих схожие роли в сети. На рисунке 1 представлен пример графа с выделенными сообществами.

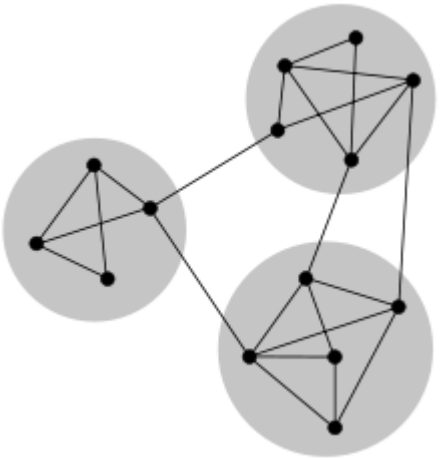


Рис. 1: Граф с тремя сообществами

Общество предлагает большое количество возможных сообществ: семья, коллеги, друзья, жители одного города, граждане одного государства и т.д. Широкое распространение Интернета привело к появлению большого количества виртуальных групп, онлайн сообществ. В сетях взаимодействий белок-белок сообществами можно назвать группы протеинов, имеющих одинаковые функции в клетке [17, 6]. В графе Всемирной Паутины сообщества могут соответствовать наборам страниц, посвященных общей

или связанным темам.

Приведу несколько примеров конкретных применений сообществ. Кластеризация веб-клиентов по их интересам и географическому положению позволяет улучшить производительность сервисов в Интернете, каждый полученный кластер может быть обработан отдельным вспомогательным сервером [13]. Кластеры больших графов могут быть использованы при создании удобных структур данных для эффективного хранения таких графов и обработки навигационных запросов, например, поиска пути из одного узла в другой [2]. Самоконфигурирующиеся сети, структура которых часто меняется (например, сети мобильной связи), обычно не имеют централизованной таблицы маршрутизации. Группировка узлов в кластеры позволяет быстро генерировать компактные локальные таблицы, обеспечивающие эффективную коммуникацию [19].

Определение сообществ и их границ позволяет классифицировать узлы графа по их функциям в кластере. Узлы в центральных позициях, имеющие большое количество связей с остальными членами группы, несут регулирующие и стабилизирующие функции. А находящиеся на границах сообществ узлы являются связующими, и ответственны за взаимодействие с другими кластерами. Приведённая классификация свойственна различного рода социальным сетям [5, 11].

Ещё один важный аспект, связанный с кластерной структурой, – иерархическая организация большинства реальных сетей. Такие сети обычно состоят из сообществ, в свою очередь содержащих более мелкие сообщества. В качестве примера можно привести сеть, отражающую строение человеческого организма: человек состоит из органов, органы из тканей, ткани из клеток и т.д. Другим примером могут служить фирмы, организованные в пирамиду из сотрудников, рабочих групп, отделов и т.д.

Целью поиска сообществ в графах является определение кластеров и, по возможности, их иерархии, используя лишь информацию о структуре этого графа.

Сообщества в социальных сетях

В настоящий момент существует более 200 широкоизвестных, и огромное количество менее популярных сайтов социальных сетей. В этих сетях

зарегистрированы миллионы людей, каждый из которых связан с десятками или даже сотнями других пользователей. Все эти сложные взаимосвязи можно удобно представить в виде социального графа пользователей.

Одной из проблем, возникающих при анализе социальных сетей (как, впрочем, и многих других реальных сетей), является направленность связей между узлами. К примеру, в графе Всемирной Паутины вершинами являются страницы, а связями – ссылки с одних страниц на другие. Эти ссылки направлены – если мы может перейти по ссылке со страницы A на страницу B, то далеко не всегда мы можем проделать обратное действие. Поиск сообществ в веб-графе может помочь определить искусственные кластеры, созданные «фермами ссылок» с целью улучшить значения PageRank [3] для определённых сайтов и повысить их позицию в поисковой выдаче Google. Таким образом, учёт направления связей может существенно улучшить качество разделения на кластеры и предоставить дополнительную ценную информацию о системе. Однако, в целях упрощения алгоритма поиска сообществ, информацию о направлении связей часто опускают.

К сожалению, направленность связей в графе не единственная проблема при работе с реальными графами. Во многих таких сетях узлы могут относиться более чем к одному кластеру. В этом случае говорят о пересекающихся сообществах. Классическим примером опять же являются социальные сети, где человек обычно принадлежит к нескольким группам одновременно: коллеги, семья, схожие спортивные предпочтения и т.д. Традиционные алгоритмы выделения сообществ сопоставляют каждую вершину только одному кластеру, таким образом, теряя потенциально важную информацию. Вершины, принадлежащие более чем одному сообществу, часто играют важную роль посредника между различными секциями графа. В тоже время стоит учитывать, что поиск пересекающихся сообществ вычислительно гораздо более сложная задача, нежели обычная кластеризация.

Последняя важная проблема в обработке социальных графов, о которой хотелось бы упомянуть, – это необходимость эффективной работы с огромными объёмами данных. К примеру, в сети Twitter более 250 миллионов пользователей и 15 миллиардов связей между ними, а в Facebook более 500 миллионов пользователей и 50 миллиардов связей. Для обработки таких объёмов данных необходимо привлекать распределённые вычислитель-

ные системы и использовать алгоритмы, допускающие распараллеливание. В качестве одного из таких вариантов можно привести модель распределённых вычислений MapReduce [7], позволяющую легко масштабировать обработку данных.

1 Постановка задачи

Целью работы является исследование и разработка распределённых алгоритмов для поиска сообществ пользователей в социальных сетях. Условно задачу можно разбить на несколько этапов:

- Провести анализ и сравнение существующих алгоритмов поиска сообществ на графах
- Выбрать наиболее эффективные и многообещающие из них
- Руководствуясь идеями, реализованными в выбранных алгоритмах, создать программное средство на базе платформы для распределённых вычислений Apache Hadoop[1], осуществляющее поиск сообществ пользователей в социальных графах
- Провести тестирование и оценку производительности разработанного решения

[1] http://hadoop.apache.org/

2 Обзор существующих решений

Целью обзора является оценка и сравнение наиболее удачных и эффективных алгоритмов кластеризации графов. На основании результатов исследования будет проведён отбор алгоритмов, на базе которых будет разработано распределённое решение.

В обзор включены наиболее успешные представители нескольких семейств алгоритмов. Далее погорим о каждом из них.

2.1 Методы иерархической кластеризации

Иерархические подходы используют меру подобия вершин и группируют вершины, чтобы получить естественное разбиение графа.

Методы иерархической кластеризации делятся на две основные категории:

- агломеративные – итеративное слияние групп вершин с высокой степенью подобия
- дивизимные – итеративное удаление рёбер между вершинами с малой степенью подобия

Широко известный алгоритм, использующий иерархическую кластеризацию, был предложен Гирван и Ньюменом [12]. Это дивизимный алгоритм, итеративно удаляющий рёбра с высоким показателем промежуточности (betweenness). Меру промежуточности можно вычислить как, например, количество кратчайших путей, пролегающих через ребро. Чем выше это значение, тем вероятнее ребро находится вне кластера и является связующим.

На рисунке 2 представлен пример распределения меры промежуточности по рёбрам.

Иерархическая кластеризация позволяет получить хорошие результаты на реальных данных, но является вычислительно сложной и немасштабируемой.

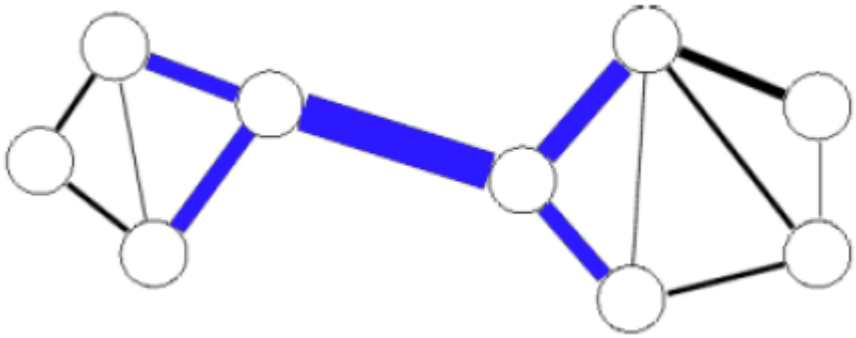


Рис. 2: Промежуточность

2.2 Максимизация целевой функции

Алгоритмы этой группы оптимизируют значение некой функции качества разбиения графа на части. Классическим выбором целевой функции является модулярность (modularity) и её модификации:

$$Q = \frac{1}{2m} \sum_{c \in P} \sum_{i,j \in c} A[i,j] - Pr(A[i,j] = 1),$$

где P – множество сообществ, A – матрица инцидентности.

Ньюмен в [16] использует агломеративный подход вкупе с жадной оптимизацией. Изначально каждая вершина является кластером, на каждом следующем шаге происходит слияние кластеров, максимально увеличивающее показатель модулярности.

Модулярность вводится как характеристика разбиения графа на непересекающиеся компоненты.

2.3 Линейные графы

Алгоритм линейных графов [10], по сути, заключается в кластеризации рёбер. Можно выделить три основных шага алгоритма:

- преобразование исходного графа в линейный
  - каждое ребро становится вершиной в линейном графе
  - каждая вершина становится кликой в линейном графе
- разбиение линейного графа на части алгоритмом для поиска непересекающихся сообществ

- обратное преобразование полученных сообществ линейного графа в сообщества исходного графа

Пример преобразования участка графа в линейный представлен на рисунке 3.

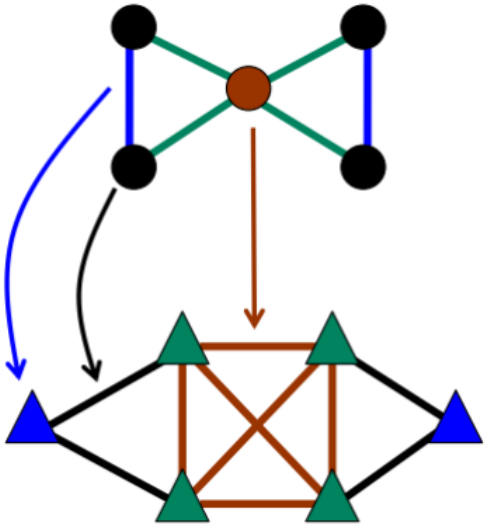


Рис. 3: Преобразование в линейный граф

2.4 Локальные методы

При локальной работе с кластерами (не принимая во внимание структуру целого графа) часто рассматривается отношение количества внутренних и внешних рёбер или треугольников кластера [8]. Кроме того, сообщества могут составляться из наборов связанных клик[2] различного размера [21, 18].

2.4.1 Перколяция клик

Метод перколяции клик [21] строит сообщества из смежных k-клик[3]. Две k-клики называются смежными, если они содержат k-1 общих вершин. Сообщество определяется, как максимальное объединение k-клик, любые две из которых соединены набором смежных k-клик. Такое определение сообщества допускает пересечения естественным образом.

[2]Клика – полный подграф графа.

[3]K-клика – полный подграф графа, состоящий из k вершин.

В улучшенном варианте классической перколяции, предложенном в [18], для поиска сообществ используется двудольный граф. Узлами это графа могут быть либо k-клики, либо k-1-клики. K-клика связана с k-1-кликой, только если последняя является подкликой этой k-клики. Таким образом, связанные непересекающиеся компоненты этого графа представляют собой конечные сообщества.

На рисунке 4 приведён пример найденного покрытия графа при k=4.

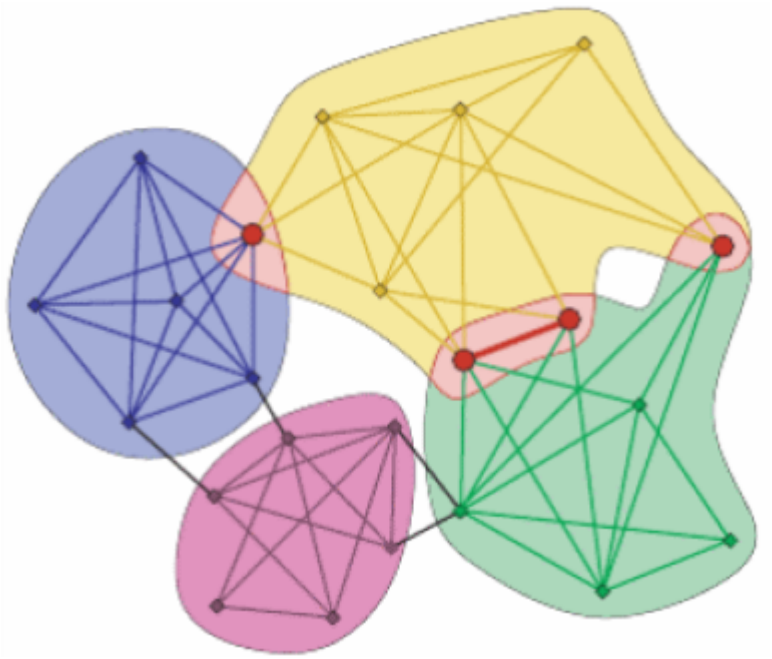


Рис. 4: Результат работы метода перколяции клик

2.4.2 Жадный алгоритм расширения клик

Этот локальный алгоритм пытается оптимизировать оценочную функцию для каждого конкретного сообщества. В начале работы алгоритм выбирает начальные ядра сообществ, как правило, это вершины или треугольники графа. Затем запускается цикл, состоящий из следующих шагов:

- выбирается самое большое незавершённое сообщество
- к этому сообществу последовательно присоединяются вершины-соседи, максимизирующие прирост оценочной функции сообщества, до тех пор, пока этот прирост не станет отрицательным
- если оценочная функция достигла максимум, сообщество считается завершённым, и алгоритм переходит на новый виток

Одним из главных улучшений, предложенных в [8], является выбор максимальных клик в качестве ядер сообществ.

3 Исследование и построение решения задачи

3.1 Исследование методов

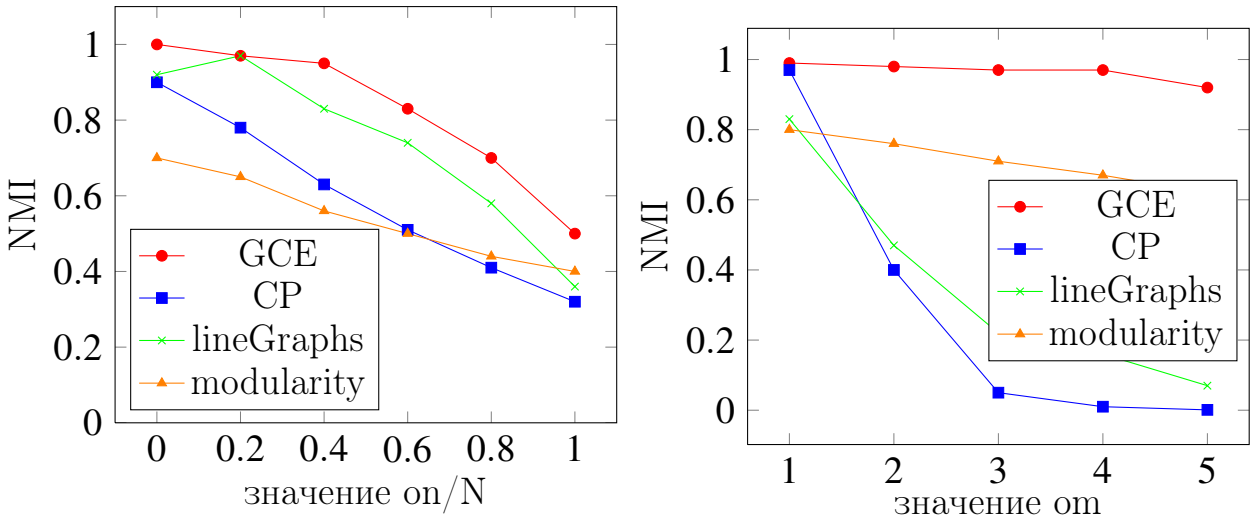
Для сравнения разбиений графа предложено немалое количество различный мер, но большинство из них подходит для работы лишь с непересекающимися разбиениями. В данной работе для оценки описанных выше методов используется мера Normalized Mutual Information (NMI) [15], которая подходит для оценки пересекающихся разбиений (покрытий):

$$NMI(X,Y) = \frac{2(H(X)+H(Y)-H(X,Y))}{H(X)+H(Y)},$$

где X и Y – разбиения графа на пересекающиеся компоненты, H – энтропия Шеннона.

В качестве тестовых графов с пересекающимися компонентами были использованы графы, сгенерированные модифицированным алгоритмом LFR [14]. На вход генератору передаются следующие параметры:

- N – количество вершин
- k – среднее значение степени вершины
- k_{max} – максимальное значение степени вершины
- C_{min} – минимальное количество вершин в сообществе
- C_{max} – максимальное количество вершин в сообществе
- τ_1 – экспонента степенного распределения степени вершины
- τ_2 – экспонента степенного распределения размера кластера
- μ – параметр выраженности кластерной структуры
- on – количество вершин, принадлежащих более чем одному сообществу
- om – скольким сообществам принадлежит каждая вершина из области пересечения



(a) (b)

Рис. 5: Результаты сравнения алгоритмов

Было сгенерировано два набора тестовых графов. Члены первого набора отличаются значением om, второго – on.

На рисунке 5 представлены графики с полученными результатами.

Как видно из графиков, наиболее эффективным алгоритмом оказался жадный алгоритм расширения клик (Greedy clique expansion, GCE), представленный в [8]. Кроме этого, алгоритм является локальным, что увеличивает шансы получить удачную реализацию на платформе Apache Hadoop.

3.2 Модель MapReduce

Для реализации распределённого алгоритма поиска сообществ пользователей была использована модель распределённых вычислений MapReduce, а именно, её свободная реализация – платформа Hadoop от Apache. Элементарная вычислительная задача в данной модели состоит из двух шагов: Map и Reduce. Вычислительные элементы, реализующие эти шаги, называются, соответственно, маппер (mapper) и редьюсер (reducer). Одновременно над одной задачей может работать большое количество мапперов и редьюсеров распределённых по машинам кластера. Данные для обработки с помощью MapReduce должны быть представлены в формате ключ-значение $< k, v >$.

Весь объём входных данных разбивается на фрагменты определённого размера, каждый такой фрагмент поступает на вход одному из мапперов.

Входящая пара $< k_{in}, v_{in} >$ некоторым образом преобразуется в промежуточную пару $< k_{int}, v_{int} >$. Затем промежуточные данные, полученные со всех мапперов, некоторым образом группируются по ключу k_{int} и поступают на вход редьюсерам в виде $< k_{int}, list\, v^i_{int} >$. Таким образом значения, соответствующие одному ключу, попадают в один редьюсер. После окончательном обработки на выходе редьюсера получаем пары $< k_{out}, v_{out} >$, которые уже записываются в выходной файл. Схематично работу MapReduce-задачи (далее MR-задачи) можно представить рисунком 6.

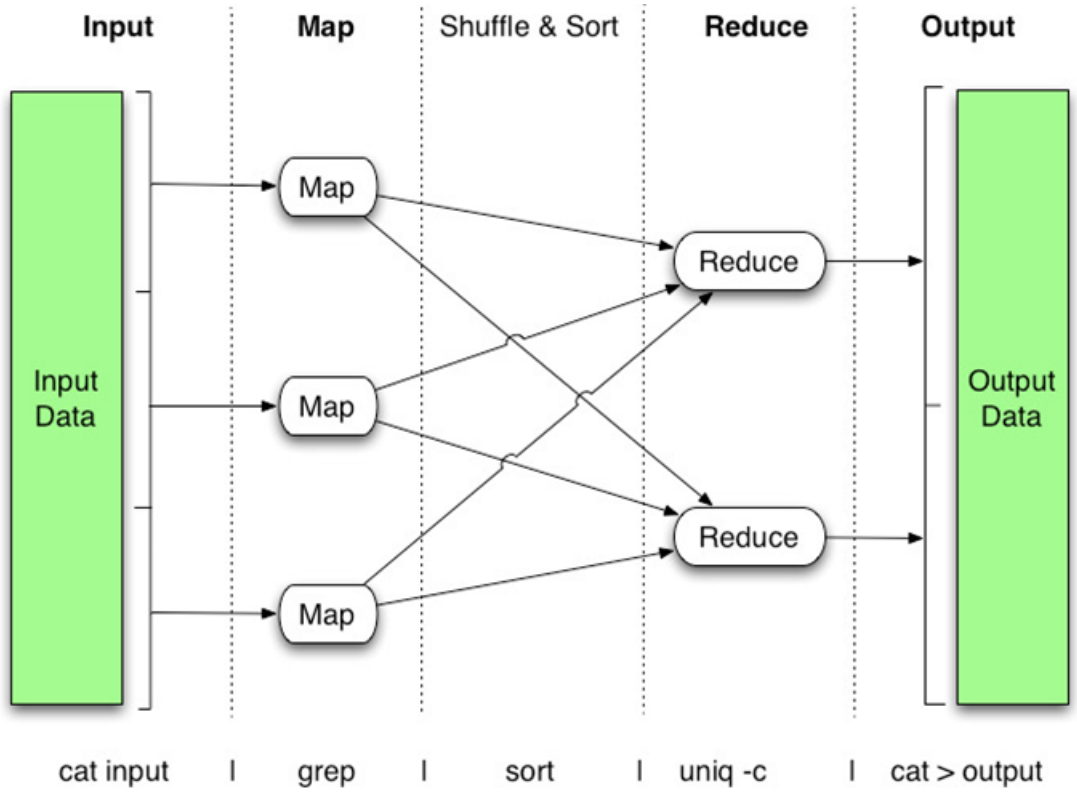


Рис. 6: Схема работы MapReduce

3.3 Построение решения задачи

На основе результатов исследования, приведённых в предыдущей секции, алгоритм GCE был выбран в качестве основы для построения распределённого решения. Рассмотрим основные шаги это алгоритма подробно.

3.3.1 Алгоритм поиска максимальных клик в графе

Одной из ключевых частей алгоритма является поиск максимальных клик в графе. Несмотря на то, что эта задача является вычислительно

сложной, клики могут быть найдены достаточно быстро в разреженных графах, коими и являются социальные сети. Авторы метода GCE в [8] предлагают использовать для этой задачи алгоритм Брона-Кербоша [4], как один из самых эффективных. Томита et al. в [20] показали, что в худшем случае сложность алгоритма $O(3^{n/3})$, где n – количество вершин в графе.

Ниже представлен листинг модифицированного алгоритма Брона-Кербоша, который будет использован в дальнейшем. ***clique*** – множество, содержащее на каждом шаге клику (не обязательно максимальную), соответствующую этому шагу, ***CAND*** – множество вершин-кандидатов на попадание в ***clique***, ***FINI*** – множество вершин, которые уже использовались для расширения ***clique*** на предыдущих итерациях цикла, $\Gamma(v)$ – вершины-соседи для v.

Алгоритм 1 Поиск максимальных клик

```
method dfs(CAND, clique)
  FINI = {}
  while CAND \ FINI != ∅ do
    v = argmax(|Γ(v) ∩ CAND \ FINI|)
    clique ∪= {v}
    CAND' = Γ(v) ∩ CAND \ FINI
    if CAND' = ∅ then
      emit clique
    else
      dfs(CAND', clique)

    CAND \= {v}
    FINI ∪= CAND'
    clique \= {v}
```

На рисунках 7 и 8 представлены пример графа для поиска клик и схема работы алгоритма на этом графе.

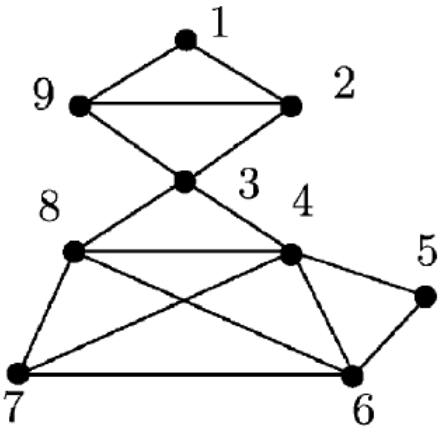


Рис. 7: Пример графа для поиска максимальных клик

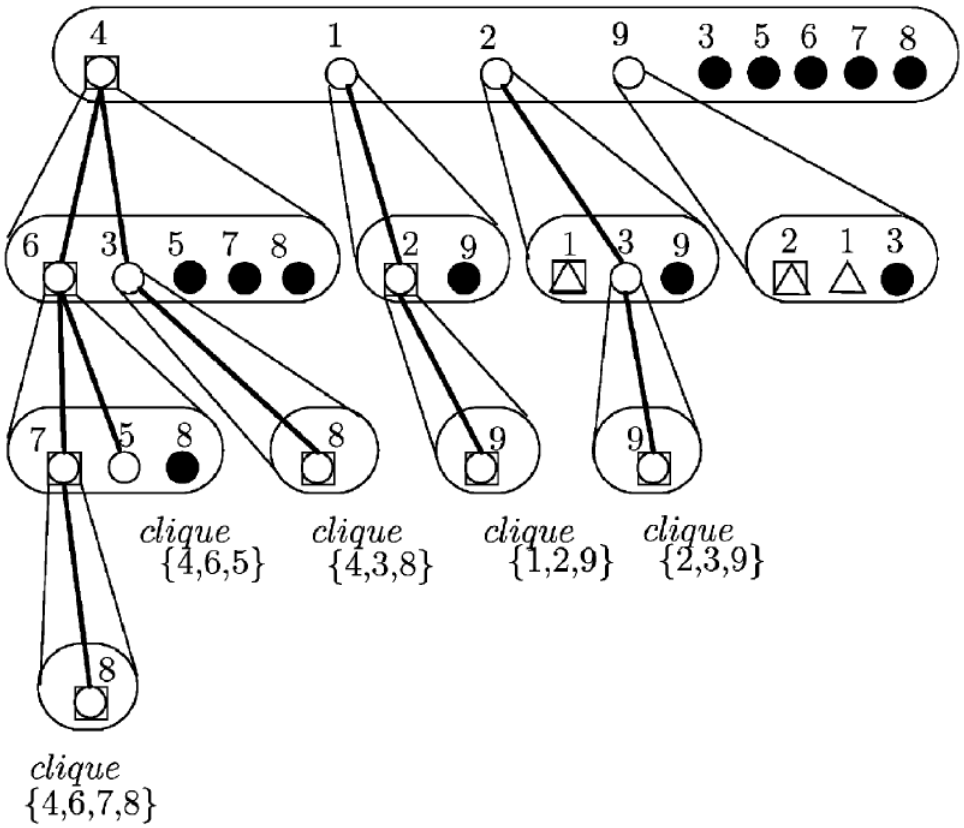


Рис. 8: Схема работы алгоритма поиска максимальных клик

Квадратом отмечены вершины, добавленные в клику на соответствующих шагах. Чёрными кругами отмечены вершины, составляющие ***FINI*** на соответствующем шаге.

3.3.2 Расширение клик

В качестве оценочной функции сообщества, значение которой оптимизирует алгоритм, возьмём локальную функцию адекватности сообщества S:

$$F_S = \frac{k_{in}^S}{(k_{in}^S + k_{out}^S)^\alpha},$$

где k_{in}^S – внутренняя степень сообщества S, k_{out}^S – внешняя степень сообщества S, α – параметр. Иначе говоря, k_{in}^S – это удвоенное число рёбер, обе вершины которых находятся в S (сумма степеней всех вершин в S), а k_{out}^S – число рёбер, только одна вершина которых принадлежит S.

Расширение клик требует знания соседей сообщества (назовём множество соседей сообщества S фронтом $f(S)$). На каждом следующем шаге выбирается вершина v_{max} из фронта, максимально увеличивающая значение адекватности для сообщества, и добавляется в это сообщество. Фронт меняется следующим образом:

$$f(S \cup v_{max}) = (f(S) \cup \Gamma(v_{max}) \backslash S) - \{v_{max}\},$$

где $\Gamma(v_{max})$ – множество соседей вершины v_{max}. При каждом обновлении фронта значения k_{in}^S и k_{out}^S также обновляются:

$$k_{in}^{S \cup v_{max}} = k_{in}^S + 2|S \cap \Gamma(v_{max})|$$

$$k_{out}^{S \cup v_{max}} = k_{out}^S + |\Gamma(v_{max})| - 2|S \cap \Gamma(v_{max})|$$

3.3.3 Отсев потенциальных дубликатов

При расширении клик не исключены случаи, когда изначально различные клики расширяются в одно и то же сообщество. Чтобы повторно не тратить ресурсы на уже проделанную работы, необходимо отсеивать такие дубликаты как можно раньше. Для этого после каждого расширения клики проверяется, не стала ли она слишком похожа на одну из уже завершённых клик. В качестве метрики используется функция расстояния

между сообществами:

$$\delta_E(S, S') = 1 - \frac{|S \cap S'|}{\min(|S|, |S'|)},$$

процент вершин меньшего сообщества, не включённых в большее сообщество. Тогда потенциальными дубликатами S будут являться сообщества, находящиеся на расстоянии не далее ε от него.

3.3.4 Модель данных

Структура `Node` для вершины состоит из двух полей:

- `nid` – идентификатор вершины, целое число, которым вершина представлена в исходных данных
- `adjacencyList` – список смежности вершины, идентификаторы всех соседей

Структура `Community` для сообщества следующая:

- `finished` – метка завершенности сообщества, имеет значение `true`, если сообщество максимально расширено
- `kIn` – внутренняя степень сообщества
- `kOut` – внешняя степень сообщества
- `nodes` – множество идентификаторов вершин, входящих в сообщество
- `frontier` – множество идентификаторов вершин, составляющих фронт сообщества

Свойства `kIn` и `kOut` введены в структуру для более удобного вычисления прироста значения адекватности для конкретной вершины-кандидата.

4 Описание практической части

4.1 Обоснование выбранного инструментария

В качестве платформы для распределённых вычислений в рамках модели MapReduce была выбрана платформа Apache Hadoop, как самая стабильная открытая реализация на текущий момент.

Основным языком программирования был выбран язык Java. Помимо того, что он является высокоуровневым и обладает достаточно высокой производительностью, язык Java является родным языком программирования для платформы Apache Hadoop и, как следствие, позволяет писать наиболее эффективные MR-программы для этой платформы.

Дополнительным языком для различного рода прототипов и вспомогательных скриптов был выбран Python. Этот язык позволяет писать код очень лаконично и быстро, а также имеет библиотеки на все случаи жизни, что и требовалось при прототипировании и создании скриптов для сопутствующих задач, например, визуализации графов.

4.2 Общая схема работы

Работу алгоритма можно разделить на несколько этапов:

- конвертация входных данных из текстового формата во внутренний бинарный формат
- поиск максимальных клик в графе
- основной этап расширения клик
  - расширение клик
  - удаление потенциальных дубликатов
  - если ещё есть незавершенные клики, то начинаем следующую итерацию, в противном случае переходим к последнему этапу
- конвертация полученных сообществ из внутреннего бинарного представления в воспринимаемое человеком текстовое

На каждом этапе запускается одна или больше MR-задача. Рассмотрим подробно эти этапы алгоритма.

4.2.1 Конвертация входных данных

Алгоритм принимает входящий граф в виде пар связанных вершин $< vertex1, vertex2 >$, группирует списки смежности для каждой вершины и сохраняет преобразованный граф в HDFS[4].

4.2.2 Поиск максимальных клик в графе

Для поиска максимальных клик модифицируем алгоритм предложенный в [9]. Сперва мы должны построить для каждой вершины v соседства второго уровня (соседи соседей). На шаге Map для каждой вершины u из $\Gamma(v)$ мы выдаём тройку $< u, < v, \Gamma(v) >>$. На шаге Reduce мы получаем все значения $< v, \Gamma(v) >$, сгруппированные по u. Таким образом для u мы знаем всех соседей v и соседей соседей $\Gamma(v)$, это и будет соседство второго уровня. Теперь уже в рамках одной машины мы ищем все максимальные клики в полученном соседстве модифицированным алгоритмом Брона-Кербоша, представленном алгоритмом 1. В результате мы найдём все клики в графе, содержащие вершину u нашего соседства. Чтобы избежать дубликатов при поиске клик, во время работы алгоритма Брона-Кербоша мы будем рассматривать только такие вершины v, что $v < u$.

Код MR-задачи второго этапа:

Алгоритм 2 Mapper

```
method map(nid n, Node N)
  emit n, N
  for v ∈ N.adjacencyList do
    emit v.nid, N
```

Алгоритм 3 Reducer

```
method reduce(nid n, [N1, N2, ...])
  cand = [N1, N2, ...]
  dfs(cand, {n})
```

[4]HDFS – Hadoop Distributed File System, распределённая файловая система платформы Hadoop.

4.2.3 Расширение клик

Этап расширения клик состоит из двух MR-задач: CountFitness и FindMaxFitness. Первая расчитывает для каждого сообщества прирост значения адекватности при добавлении одной из вершин фронта. Вторая находит максимум среди всех значений и добавляет соответствующую вершину в сообщество.

На вход CountFitness подаются текущие сообщества и структура графа. Маппер для каждой вершины nid фронта сообщества S отдаёт пару $< nid, S >$. Редьюсер вместе со группированными по nid сообществами получает и саму вершину v с идентификатором nid. Для каждого сообщества вычисляется значение адекватности $F_{S \cup v}$ и отдаётся тройка $< S, < F_{S \cup v}, v >>$.

FindMaxFitness в качестве маппера использует IdentityMapper. В редьюсере же определяется максимальное значение $F_{S \cup v}$ по всем парам $< F_{S \cup v}, v >$, и соответствующая вершина v добавляется к сообществу S. Обновлённое сообщество подаётся на выход.

Этап завершается, если после очередного витка цикла не осталось больше незавершённых сообществ. Иначе запускается пара задач CountFitness и FindMaxFitness для обновлённых сообществ.

Код MR-задач третьего этапа:

Алгоритм 4 CountFitnessMapper

```
method map(nid n, V)
  if isNode(V) then
    emit n, V
  else
    for node ∈ V.frontier do
      emit node, V
```

4.2.4 Удаление потенциальных дубликатов

Этот этап подразделён на две MR-задачи: GetAdjacent и RejectDuplicated. На первом шаге определяются группы пересекающихся сообществ относительно некоторых точек пересечения. Следом выявляются незавершённые

Алгоритм 5 CountFitnessReducer

```
method reduce(nid n, [V1, V2, ...])
  N = null
  communities = []
  for item ∈ [V1, V2, ...] do
    if isNode(item) then
      N = item
    else
      communities += item
  for item ∈ communities do
    emit item, <item.getFitness(N), N>
```

Алгоритм 6 FindMaxFitnessReducer

```
method reduce(Community C, [<double ftns, Node N>, ...])
  node = max_ftns [<double ftns, Node N>, ...]
  C.expand(node)
  emit node.nid, C
```

сообщества, слабо отличающиеся от смежных с ними завершённых. Такие сообщества удаляются.

GetAdjacent на вход принимает текущие сообщества. Маппер для каждой вершины nid сообщества S отдаёт пару $< nid, S >$. Редьюсер сгруппированные по точкам сообщества разделяет на завершённые и незавершённые. Затем для каждого незавершённого сообщества NF и каждого пересекающегося с ним завершённого сообщества F отдаёт пару $< NF, F >$ на вывод.

В RejectDuplicated в качестве маппера используется IdentityMapper. Редьюсер получает на вход пару $< NF, list\, F_i >$, где $list\, F_i$ - список всех завершённых сообществ пересекающихся с незавершённым NF. Для каждого F_i вычисляется расстояние $\delta_E(NF, F_i)$, и если оно меньше порогового значения, то NF отбрасывается.

Код MR-задач четвертого этапа:

Алгоритм 7 GetAdjacentMapper

```
method map(Community C)
  for n ∈ C.nodes do
    emit n, C
```

Алгоритм 8 GetAdjacentReducer

```
method reduce(nid n, Communities [C1, C2, ...])
  finished = []
  proceeding = []
  for comm ∈ [C1, C2, ...] do
    if comm.finished then
      finished += comm
    else
      proceeding += comm
  for NF ∈ proceeding do
    for F ∈ finished do
      emit NF, F
```

Алгоритм 9 RejectDuplicatedReducer

```
method reduce(Community NF, Communities [F1, F2, ...]])
  rejected = false
  for F ∈ [F1, F2, ...] do
    if rejected or distance(NF, F) > ε then
      emit key, F
    else
      if NF.getFitness() < F.getFitness() then
        rejected = true
        emit key, F
  if not rejected then
    emit key, NF
```

4.2.5 Конвертация результатов работы алгоритма

На последнем шаге алгоритма все найденные сообщества конвертируются в текстовый формат. На выходе получаем набор текстовых файлов с сообществами. Каждое сообщество занимает одну строку и представляет собой список идентификаторов вершин, разделённых пробелами.

4.3 Тестирование

На этапе тестирования в экспериментах также был задействован распределённый вариант улучшенного алгоритма перколяции клик из [18]. Этот алгоритм был реализован в рамках курсовой работы [1].

4.3.1 Тестовый стенд

Тестирование проводилось в облачном сервисе Amazon Elastic Compute Cloud (Amazon EC2)[5]. Был использован виртуальный hadoop-кластер из шести рабочих узлов типа m1.small[6]. Конфигурация каждого узла:

- 1.7 ГБ ОЗУ
- 1 стандартное вычислительное ядро (соответствует вычислительной мощности процессора 1.0-1.2 ГГц 2007 Opteron)
- 160 ГБ жёсткий диск
- 32-битная платформа

4.3.2 Оценка эффективности

Для оценка эффективности распределённого алгоритма было сгенерировано два набора LFR-графов. Параметры генератора для обоих наборов приведены в таблице 1. А на графиках 9a и 9b отражены результаты тестирования на первом и втором наборе, соответственно.

4.3.3 Тестирование на больших объёмах данных

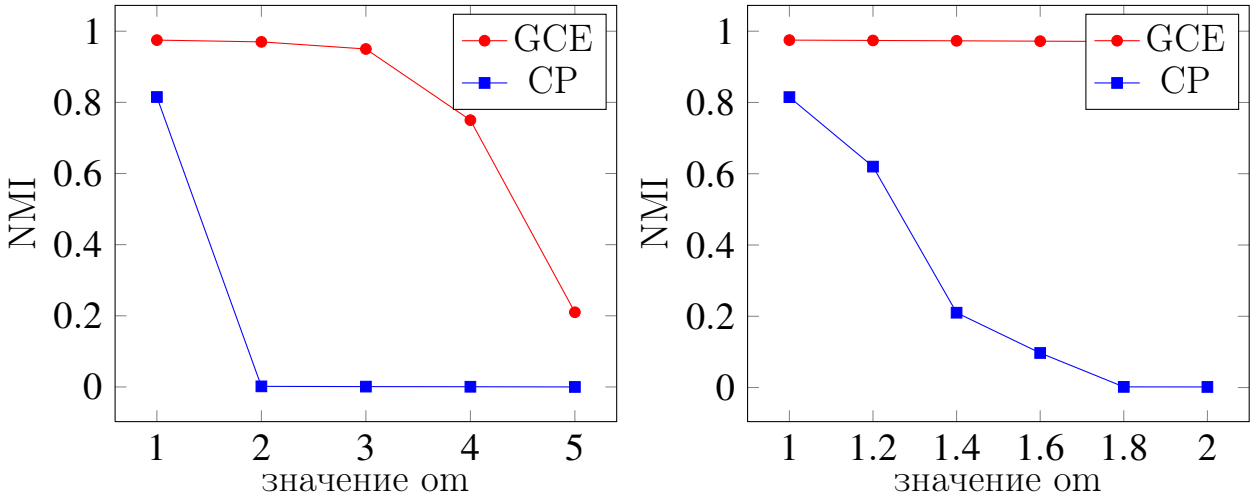
Для оценки производительности алгоритмов на больших данных был подготовлен набор LFR-графов различающихся количеством вершин – от 100 до 1000000. В процессе тестирования при увеличении количества вершин в графе наблюдался экспоненциальный рост времени обработки. К сожалению, на 5000 вершин для GCE и 100000 вершин для CP время работы превысило сутки, по этой причине тестирование решено было прекра-

[5]http://aws.amazon.com/ec2/

[6]http://aws.amazon.com/ec2/instance-types/

параметр	график 9a	график 9b
N	2000	2000
k	18-90	18-36
k_{max}	120	120
C_{min}	60	60
C_{max}	100	100
τ_1	2	2
τ_2	1	1
μ	0.2	0.2
on	2000	0-2000
om	1-5	2

Таблица 1: Параметры LFR-графов



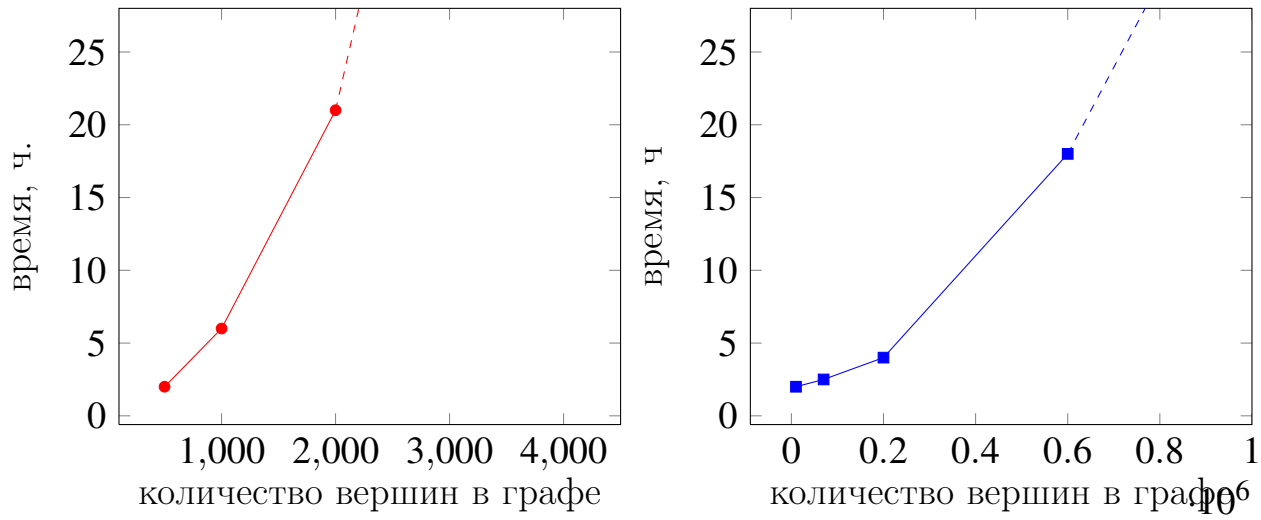
(a) (b)

Рис. 9: Значения показателя NMI для различных значений *om*

тить. На рисунке 10 представлены графики падения производительности в зависимости от количества вершин в графе.

4.3.4 Выводы

В ходе анализа проблемы выяснилось, что основной причиной ухудшения производительности послужило увеличение нагрузки на систему ввода-вывода и сеть. На этапе расширения клик при построении фронта сообщества генерируется большое количество промежуточных данных. Помимо этого, чтобы иметь доступ к структуре графа, не помещающегося в памяти, мы вынуждены передавать эту структуру вместе с остальными нашими данными. Все итоговые записи неизбежно подвергаются сортировке,



(a) GCE

(b) CP

Рис. 10: Падение производительности при увеличении количества вершин

группировке по ключу и пересылке с мапперов на редьюсеры. Как показали наши эксперименты, даже на относительно небольших графах объёмы промежуточных данных велики, и их пересылка занимает значительное время.

На основе вышесказанного можно сделать вывод, что модель вычислений MapReduce является нелучшим выбором при реализации алгоритмов, требующих постоянного доступа к графовой структуре. В качестве возможных путей решения можно выделить поиск алгоритма, подходящего для реализации в модели MapReduce, либо поиск замены для выбранной в этой работе модели распределённых вычислений.

Заключение

В рамках данной дипломной работы были получены следующие результаты:

- Проведён анализ и сравнение существующих алгоритмов поиска сообществ на графах
- Выбраны наиболее эффективные и многообещающие из них
- На базе платформы для распределённых вычислений Apache Hadoop создано программное средство, осуществляющее поиск сообществ пользователей в социальных графах
- Проведено тестирование и оценка производительности разработанного решения

Список литературы

[1] Рябов, С. Распределённый алгоритм для поиска сообществ пользователей в социальных сетях / С. Рябов, А. Коршунов. — 2011.

[2] Agrawal, R. Algorithms for searching massive graphs / R. Agrawal, H. Jagadish // Knowledge and Data Engineering, IEEE Transactions on. — 1994. — V. 6, № 2. — P. 225–238.

[3] Brin, S. The anatomy of a large-scale hypertextual Web search engine / S. Brin, L. Page // Computer networks and ISDN systems. — 1998. — V. 30, № 1-7. — P. 107–117.

[4] Bron, C. Algorithm 457: finding all cliques of an undirected graph / C. Bron, J. Kerbosch // Communications of the ACM. — 1973. — V. 16, № 9. — P. 575–577.

[5] Burt, R. Positions in networks / R. Burt // Social forces. — 1976. — V. 55, № 1. — P. 93–122.

[6] Chen, J. Detecting functional modules in the yeast protein–protein interaction network / J. Chen, B. Yuan // Bioinformatics. — 2006. — V. 22, № 18. — P. 2283–2290.

[7] Dean, J. MapReduce: Simplified data processing on large clusters / J. Dean, S. Ghemawat // Communications of the ACM. — 2008. — V. 51, № 1. — P. 107–113.

[8] Detecting highly overlapping community structure by greedy clique expansion / C. Lee, F. Reid, A. McDaid, N. Hurley // Arxiv preprint arXiv:1002.1827. — 2010.

[9] A distributed algorithm to enumerate all maximal cliques in mapreduce / B. Wu, S. Yang, H. Zhao, B. Wang // Frontier of Computer Science and Technology, 2009. FCST'09. Fourth International Conference on / IEEE. — 2009. — P. 45–51.

[10] Evans, T. Line graphs of weighted networks for overlapping communities /

T. Evans, R. Lambiotte // The European Physical Journal B-Condensed Matter and Complex Systems. — 2010. — V. 77, № 2. — P. 265–272.

[11] Freeman, L. A set of measures of centrality based on betweenness / L. Freeman // Sociometry. — 1977. — P. 35–41.

[12] Girvan, M. Community structure in social and biological networks / M. Girvan, M. Newman // Proceedings of the National Academy of Sciences. — 2002. — V. 99, № 12. — P. 7821–7826.

[13] Krishnamurthy, B. On network-aware clustering of web clients / B. Krishnamurthy, J. Wang // ACM SIGCOMM Computer Communication Review / ACM. — V. 30. — 2000. — P. 97–110.

[14] Lancichinetti, A. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities / A. Lancichinetti, S. Fortunato // Physical Review E. — 2009. — V. 80, № 1. — P. 016118.

[15] Lancichinetti, A. Detecting the overlapping and hierarchical community structure in complex networks / A. Lancichinetti, S. Fortunato, J. Kertész // New Journal of Physics. — 2009. — V. 11. — P. 033015.

[16] Newman, M. Finding and evaluating community structure in networks / M. Newman, M. Girvan // Physical review E. — 2004. — V. 69, № 2. — P. 026113.

[17] Rives, A. Modular organization of cellular networks / A. Rives, T. Galitski // Proceedings of the National Academy of Sciences. — 2003. — V. 100, № 3. — P. 1128–1133.

[18] Sequential algorithm for fast clique percolation / J. Kumpula, M. Kivelä, K. Kaski, J. Saramäki // Physical Review E. — 2008. — V. 78, № 2. — P. 026109.

[19] Steenstrup, M. Cluster-based networks / M. Steenstrup // Ad hoc networking / Addison-Wesley Longman Publishing Co., Inc. — 2001. — P. 75–138.

[20] Tomita, E. The worst-case time complexity for generating all maximal cliques and computational experiments / E. Tomita, A. Tanaka, H. Takahashi // Theoretical Computer Science. — 2006. — V. 363, № 1. — P. 28–42.

[21] Uncovering the overlapping community structure of complex networks in nature and society / G. Palla, I. Derényi, I. Farkas, T. Vicsek // Nature. — 2005. — V. 435, № 7043. — P. 814–818.

[22] Wasserman, S. Social network analysis / S. Wasserman, K. Faust. — 1994.