МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Применение методов обработки графов для задач анализа больших данных»

Студент П.В. Гусаров _____
(И.О. Фамилия) (личная подпись)

Руководитель канд.пед.наук., доцент, Т.А. Агошкова
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема выпускной квалификационной работы – «Применение методов обработки графов для задач анализа больших данных».

Для решения задач анализа больших данных требуются новые парадигмы, методы и алгоритмы, которые позволяют эффективно обрабатывают большие объемы данных, используя их структуру. Применение методов обработки графов для задач анализа больших данных представляет научно-практический интерес.

Объектом исследования бакалаврской работы является анализ больших данных.

Предметом исследования бакалаврской работы являются методы обработки графов.

Цель бакалаврской работы – применение методов обработки графов для повышения эффективности анализа больших данных.

Методы исследования – наука о данных (Data Science), методы обработки графов.

Практическая значимость бакалаврской работы заключается в разработке программного обеспечения для анализа больших данных методами визуализации графов.

Результаты бакалаврской работы могут быть рекомендованы для разработчиков программ, использующих методы обработки графов для задач анализа больших данных.

Выпускная квалификационная работа состоит из 44 страниц текста, 24 рисунков, 2 таблиц и 21 источника.

Abstract

The topic of the given graduation work is Utilizing graph processing methods for big data analysis tasks.

To solve the problems of big data analysis, new paradigms, methods and algorithms are required that allow us efficiently processing large amounts of data using their structure. Utilizing graph processing methods for big data analysis problems is of scientific and practical interest.

The object of study of the graduation work is big data analysis.

The subject of study of the graduation work is graph processing methods.

The aim of the graduation work is utilizing graph processing methods to improve the efficiency of big data analysis.

Research methods: Data Science, graph processing methods.

The practical significance of the graduation work lies in the development of software for analyzing big data using graph visualization methods.

The results of the graduation work can be recommended for program developers using graph processing methods for big data analysis tasks.

The graduation work consists of an explanatory note on 44 pages including 24 figures, 2 tables, the list of 21 references.

Оглавление

Введение

В последнее время бизнес-аналитики в различных областях человеческой деятельности сталкиваются с проблемой обработки быстро растущих объемов данных, которые собираются в многочисленных приложениях.

К таким областям относятся: биохимические и генетические исследования, фундаментальные физические эксперименты и астрономические наблюдения, социальные сети, исследования поведения потребителей и многое другое.

В этих приложениях большие объемы необработанных данных могут использоваться для принятия решений и планирования действий, но их размеры и сложная структура ограничивают применимость многих хорошо известных подходов, широко используемых с небольшими наборами данных, таких как анализ главных компонентов, разложение по сингулярным значениям. положение, спектральный анализ и другие.

Для решения данной проблемы требуются новые парадигмы, методы и алгоритмы, которые позволяют эффективно обрабатывают большие объемы данных, используя их структуру.

Естественной формой представления структурированных или частично структурированных данных являются графы.

Применение методов обработки графов для задач анализа больших данных представляет научно-практический интерес.

Объектом исследования бакалаврской работы является анализ больших данных.

Предметом исследования бакалаврской работы являются методы обработки графов.

Цель бакалаврской работы – применение методов обработки графов для повышения эффективности анализа больших данных.

Для достижения данной цели необходимо выполнить следующие

задачи:

– проанализировать методы обработки графов;

– проанализировать алгоритмы анализа больших данных на графах;

– разработать программное обеспечение для анализа больших данных методами визуализации графов и оценить их эффективность.

Методы исследования – наука о данных (Data Science), методы обработки графов.

Практическая значимость бакалаврской работы заключается в разработке программного обеспечения для анализа больших данных методами визуализации графов.

Данная работа состоит из введения, трех глав, заключения и списка используемой литературы.

В первой главе даны обзор и анализ методов обработки больших графов.

Во второй главе рассматриваются характеристики алгоритмов анализа обработки больших графов и их применение для повышения эффективности анализа больших данных.

Третья глава посвящена разработке программного обеспечения для анализа больших данных методами визуализации графов и оценке их эффективность.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Бакалаврская работа состоит из 44 страниц текста, 24 рисунков, 2 таблиц и 21 источника.

Глава 1 Методы обработки больших графов

Для задач анализа больших данных с помощью аппарата графов используется технология интеллектуального анализа графов или Graph Mining.

Graph Mining - это набор инструментов и методов, используемых для анализа свойств реальных больших графов, прогнозирования того, как структура и свойства данного графа могут повлиять на некоторые приложения, и разработки моделей, которые могут генерировать графы, соответствующие образцам (шаблонам) обнаруженным в исследуемых реальных графах [10].

Рассмотрим методы обработки больших графов.

1.1 Сопоставление шаблонов подграфов

Одна из самых сложных задач в интеллектуальном анализе графов - это анализ шаблонов в больших графах.

Эта задача заключается в использовании алгоритмов интеллектуального анализа данных для обнаружения интересных, неожиданных и полезных закономерностей в больших объемах данных с графовой структурой [8].

Нахождение часто встречающихся подграфов (Frequent subgraph mining – FSM) позволяет эффективно структурировать информацию, предлагая метрики, позволяющие охватить различные понятия структуры.

Например, обнаружение общих структур белка или общих шаблоны структуры объектов, обнаружение мошенничества путем выявления похожих структур мошеннических транзакций в миллионах электронных платежей и т.д. [3].

Задача сопоставления подграфов или поиска изоморфного подграфа определяется следующим образом [21]:

Пусть $G = (V, E, l)$ - граф,

где:

V - множество вершин;

E - множество ребер,

l - функция разметки, которая присваивает метку каждой вершине в V.

Пусть $Q = (V_q, E_q, l_q)$ - граф запроса (шаблон),

где:

V_q - набор вершин;

E_q - набор ребер:

l_q - функция разметки на V_q.

Интуитивно понятно, что цель сопоставления с шаблоном подграфа состоит в том, чтобы найти все подграфы из графа данных G, которые соответствуют шаблону графа Q.

Таким образом, G' (V', E', l') является подграфом G тогда и только тогда, когда соблюдаются условия:

$V' \subseteq V$;

$E' \subseteq E$;

$\forall u \in V'$: $l'(u) = l$.

На рисунке 1 изображен пример исходного графа (а) и его подграфа (б).

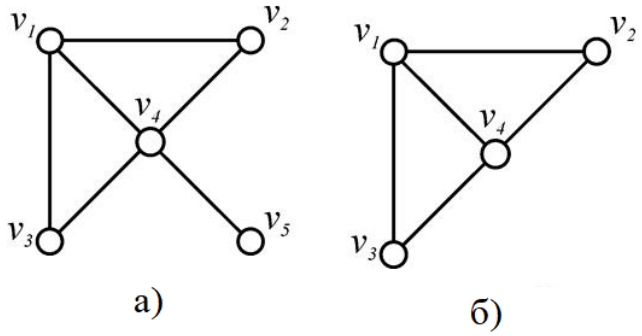

Рисунок 1 – Пример графа и подграфа

Предполагается, что все вершины помечены, все ребра ориентированы и кратных ребер нет.

Также предполагаем, что граф запроса является связным графом, потому что результат сопоставления для несвязанного графа запроса равен объединению результатов для его связанных компонентов.

Мы используем термины шаблон и граф запроса как синонимы.

Рассмотрим типы сопоставлений шаблонов:

1. Изоморфизм подграфов. Это наиболее широко изучаемая проблема сопоставления графов с шаблонами.

По определению изоморфизм подграфов описывает биективное отображение между шаблоном $Q(V_q, E_q, l_q)$ и подграфом графа данных $G = (V, E, l)$.

То есть, допуская, что G' (V', E', l') является подграфом G, граф Q будет подграфом, изоморфным G, если существует биективная функция f из вершин Q в вершины G′ такая, что (u,v) является ребром в Q тогда и только тогда, если $(f(u), f(v))$ - ребро в G' .

Следует отметить, что функция гарантирует, что u и $f(u)$ имеют одинаковые метки. Для решения данной задачи используются алгоритмы Ульмана и VF2.

Однако в целом эта проблема является NP-трудной, поэтому точные алгоритмы не используются для очень больших графов.

2. Моделирование графа.

Моделирование графа позволяет ускорить поиск изоморфизма подграфов, ослабив некоторые ограничения. Шаблон $Q(V_q, E_q, l_q)$ сопоставляется с графом данных $G(V, E, l)$ посредством моделирования графа, если существует двоичное отношение $R \subseteq V_q \times V$, такое, что 1) для каждого $u \in V_q$ существует $u' \in V$ такое, что $(u,u') \in R$; 2) $l_q(u) = l(u')$ и 3) для каждого $v \in$ Child(u) существует $(v, v') \in R$ такое, что $v' \in$Child (u'). Здесь функция Child возвращает всех прямых потомков данной вершины.

Интуитивно, графическое моделирование фиксирует только дочерние отношения вершин. Для моделирования графов используется HHK - квадратичный алгоритм, который эффективно вычисляет набор совпадений на

графах среднего размера, но он все еще недостаточно эффективен для массивных графов.

3. Двойное моделирование.

Двойное моделирование расширяет моделирование графа, также принимая во внимание родительские отношения вершин, что приводит к более строгому набору соответствий.

Шаблон $Q(V_q, E_q, l_q)$ сопоставляется с граф данных $G(V, E, l)$ посредством двойного моделирования Q. Метод представляет собой модельное сопоставление с G с отношением сопоставления $R \subseteq V_q \times V$, причем для каждого $w \in$Parent (u) существует $(w, w') \in R$ такой, что $w' \in$Parent (w'). Метод Parent возвращает всех прямых родителей вершины.

4. Строгое моделирование.

Строгое моделирование основано на двойном моделировании путем введения условия локальности.

Как показывает практика, использование того или иного типа сопоставления шаблона подграфа зависит от конкретной задачи, объема данных и структуры исследуемого графа.

1.2 Модели вычислений на больших графах

С появлением вычислений больших данных были пересмотрены вычислительные модели для алгоритмов графов.

За прошедшие годы был предложен ряд идей для эффективной и масштабируемой обработки больших графов:

– MPI-подобные модели.

Несколько библиотек были разработаны с использованием MPI (Message Passing Interface, интерфейса передачи сообщений) в течение последнего десятилетия, чтобы предоставить платформы для обработки распределенных графов, например, Parallel BGL и CGMgraph [5].

Однако эти библиотеки не поддерживают отказоустойчивость или

некоторые другие проблемы, важные для очень крупномасштабных распределенных систем.

– Модель MapReduce.

Google представила систему, основанную на модели MapReduce для обработки больших наборов данных. Она обеспечивает отказоустойчивость и простоту программирования. Однако эта модель по своей сути не подходит для итерационного алгоритма, который характерен для большинства алгоритмов графа.

– Vertex-Centric BSP (вершино-центрическая с параллельной обработкой)- это модель, предложенная компаний Valiant в качестве модели вычислений для параллельной обработки (рисунок 2) [20].

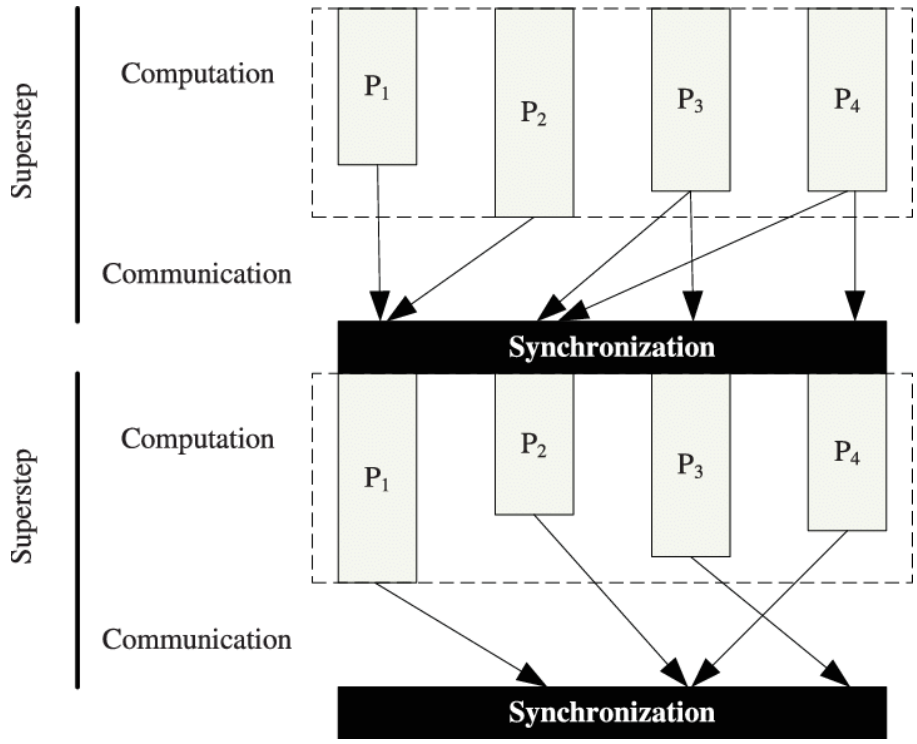

Рисунок 2 – Модель вычислений Vertex-Centric BSP

Как показано на рисунке, вычисления в этой модели представляют собой серию супершагов (supersteps).

Каждый супершаг содержит три упорядоченных этапа:

– параллельные вычисления, когда разные процессы выполняются одновременно;

– коммуникация, при которой все процессы обмениваются сообщениями;

– барьерная синхронизация, при которой каждый процесс ожидает, пока другие достигнут того же состояния, прежде чем перейти к следующему супершагу.

В модели программирования Vertex-Centric BSP каждая вершина графа данных представляет собой вычислительную единицу, которую можно концептуально сопоставить с процессом в модели параллельных вычислений BSP [9].

– модель Vertex-Centric Asynchronous (вершино-центрическая асинхронная).

Хотя реализация системы, основанной на модели BSP, делает систему по сути свободной от каких-либо тупиков, ее этап синхронизации может создать узкое место, так как на каждом супершаге возможно ожидание выполнения самого медленного из них.

Однако есть несколько других систем, которые следуют идее Vertex-Centric Asynchronous.

Они пользуются преимуществами вершино-центрической модели, но избегают узких мест модели BSP.

Примерами являются GraphLab и Signal / Collect в его асинхронном режиме [3].

– партиционирование графов.

Партиционирование графов находит широкое применение во многих областях, включая проектирование телефонных сетей, проектирование сверхбольших интегральных схем и планирование задач.

Задача состоит в том, чтобы разбить граф на примерно равное количество частей, так чтобы количество ребер, соединяющих вершины в разных частях, было минимальным.

Это важно в контексте распределенных вычислений, потому что мы хотим разбить граф на части, чтобы каждая часть была в основном автономной.

То есть мы хотим максимально сократить его взаимодействие с другими частями, что теоретически приводит к ускорению.

Однако это нетривиальная проблема. Когда мы разбиваем граф, нам нужно позаботиться о двух моментах:

– необходимо приложить усилия, чтобы разделить граф на равные части, чтобы каждое вычислительная система получила соответствующую нагрузку;

– необходимо минимизировать количество ребер, идущих от одной раздела к другому.

Партиционирование, обеспечивающее выполнение двух перечисленных выше условий, называется разбиением min-cut [17]. Это NP-полная проблема, которая сама по себе широко изучена.

Она успешно использовалась со значительными улучшениями в различных приложениях. Следует отметить, что партиционирование графа не гарантирует последовательного улучшения всех алгоритмов графа. Это ограничивает применение данного метода для анализа больших графов.

1.3 Методы визуализации графов

Визуализация графов - это область математики и информатики, находящаяся на пересечении геометрической теории графов и визуализации информации. Она связана с визуальным представлением графов, которое выявляет структуры и аномалии, которые могут присутствовать в данных, и помогает пользователю понять и анализировать большие графы [10].

Визуализация графа, иногда называемая «анализом связей» или «визуализацией сети», представляет собой процесс визуального представления сетей связанных объектов в виде узлов и связей.

Визуализация графа - это визуальное представление узлов и ребер графа.

Основные преимущества визуализации:

– уменьшение затрат времени на усвоение информации, потому что человеческий мозг обрабатывает визуальную информацию намного быстрее, чем записанную. Визуальное отображение данных обеспечивает более быстрое понимание, что, в конечном итоге, сокращает время, необходимое для действий;

– больше возможностей для обнаружения закономерностей, взаимодействуя с данными. Инструменты визуализации графиков предлагают возможность манипулировать данными. Это поощряет присвоение данных, их опрос и, в конечном итоге, увеличивает возможность обнаружить действенные идеи. Исследование показало, что менеджеры, использующие инструменты визуального обнаружения данных, на 28% чаще находят своевременную информацию, чем те, кто полагается исключительно на управляемую отчетность и информационные панели;

– лучшее понимание проблемы. Инструменты визуализации графов идеально подходят для визуализации отношений, а также для понимания контекста данных. Можно получить полный обзор того, как все взаимосвязано, что позволяет выявлять тенденции и корреляции в данных;

– эффективная форма общения. Визуальные представления предлагают более интуитивно понятный способ понимания данных и являются эффективным средством, позволяющим поделиться своими выводами с лицами, принимающими решения;

– для взаимодействия с визуализацией графов не требуются определенные навыки программирования. Это увеличивает потенциал создания стоимости.

Визуализация позволяет получить огромные преимущества от выхода за рамки плоской модели данных с помощью мощного программного обеспечения для визуализации. Специальные алгоритмы, называемые макетами, вычисляют положения узлов и отображают данные в двух (иногда

трехмерных) пространствах.

Для выбора метода обработки больших графов используем сравнительную таблицу 1. Критерии оценивания:

0 – полное несоответствие требованиям;

1 – значительное несоответствие требованиям;

2 – незначительное несоответствие требованиям;

3 – полное соответствие требованиям.

Таблица 1 – Сравнительный анализ методов обработки больших графов

Характеристика/балл	Сопоставление шаблонов подграфов	Модели вычислений на больших графах	Визуализация графов
аналитические возможности	1	2	3
производительность	2	1	3
простота реализации	3	2	2
Итого	6	5	8

Таким образом, наилучшими возможностями для аналитической обработки больших графов обладает метод визуализации графов.

Выводы по главе 1

Результаты проделанной работы позволили сделать следующие выводы:

1. Для задач анализа больших данных с помощью аппарата графов используется технология интеллектуального анализа графов или Graph Mining.

2. Можно выделить следующие методы обработки больших графов для последующего анализа: сопоставление шаблонов подграфов, модели вычислений на больших графах и визуализация графов.

3. Результаты сравнительного анализа показали, что наилучшими возможностями для аналитической обработки больших графов обладает метод визуализации графов.

Глава 2 Алгоритмы анализа больших данных на графах

Рассмотрим некоторые из наиболее важных алгоритмов графов, которые используются для анализа больших данных [11].

2.1 Алгоритм нахождения связанных компонентов

Две вершины (*u* и *v*) ориентированного графа называют сильно связными, если существует путь из *u* в *v* и существует путь из *v* в *u*.

«Ориентированный граф называется сильно связным, если любые две его вершины сильно связны.

Отношение сильной связности — это отношение эквивалентности.

Компонентой сильной связности называется класс эквивалентности множества вершин ориентированного графа относительно отношения сильной связности.

Другими словами, компонента сильной связности – это сильно связный подграф. Так как сильная связность — это отношение эквивалентности, то граф разбивается на сильно связные компоненты.

Задача состоит в нахождении таких классов эквивалентности.

Алгоритмы нахождения связных графов основаны на методе DFS — это алгоритм поиска или обхода графа в глубину» [2].

Рассмотрим поиск связанных компонентов с помощью алгоритма Косараджу [16].

Алгоритм Косараджу - это алгоритм на основе DFS, используемый для поиска сильно связанных компонентов в графе.

Он основан на идее, что если можно достичь вершины *v*, начиная с вершины *u*, то можно достичь вершины *u*, начиная с вершины *v*, и в таком случае можно сказать, что вершины *u* и *v* являются сильно связные - они находятся в сильно связном подграфе (рисунок 3).

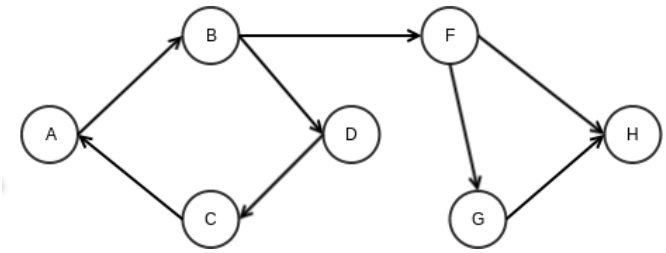

Рисунок 3 – Сильно связанные графы

Алгоритм Косараджу состоит из следующих шагов:

Шаг 1. Выполните DFS на исходном графе, отслеживая время окончания каждого узла. Это можно сделать с помощью стека, когда DFS завершает размещение исходной вершины в стеке. Таким образом, узел с наибольшим временем завершения будет на вершине стека.

Шаг 2. Переверните исходный граф, используя смежный список.

Шаг 3. Сделайте DFS на перевернутом графе с исходной вершиной в качестве вершины на вершине стека. По завершении работы DFS все посещенные узлы сформируют один прочно связанный компонент. Если какие-либо другие узлы остаются не посещенными, это означает, что есть еще более прочно связанные компоненты, поэтому выталкивайте вершины из вершины стека, пока не будет найден действительный не посещенный узел.

Это будет самое большое время завершения из всех узлов, которые в настоящее время не посещаются. Этот шаг повторяется до тех пор, пока не будут посещены все узлы.

Псевдокод алгоритма Косараджу представлен на рисунке 4.

```
stack STACK
void DFS(int source) {
  visited[s]=true
  for all neighbours X of source that are not visited:
    DFS(X)
  STACK.push(source)}
```

Рисунок 4 –Псевдокод алгоритма Косараджу

```
CLEAR ADJACENCY_LIST
for all edges e:
   first = one end point of e
   second = other end point of e
   ADJACENCY_LIST[second].push(first)
while STACK is not empty:
   source = STACK.top()
   STACK.pop()
   if source is visited :
      continue
   else :
DFS(source)
```

Продолжение рисунка 4

Главное преимущество алгоритма Косараджу – простота понимания и реализации.

Однако данных алгоритм недостаточно эффективен для анализа методами визуализации графов.

2.2 Алгоритм PageRank

Алгоритм PageRank выводит распределение вероятностей, используемое для представления вероятности того, что человек, случайно нажимающий на ссылки, попадет на любую конкретную страницу.

PageRank можно рассчитать для коллекций документов любого размера.

В нескольких исследовательских работах предполагается, что распределение равномерно распределяется между всеми документами в коллекции в начале вычислительного процесса.

Для вычисления PageRank требуется несколько проходов, называемых «итерациями», через коллекцию для корректировки приблизительных значений PageRank для более точного отражения теоретического истинного

значения [18].

PageRank - это алгоритм ранжирования страниц, который долгое время использовался в Google (рисунок 5).

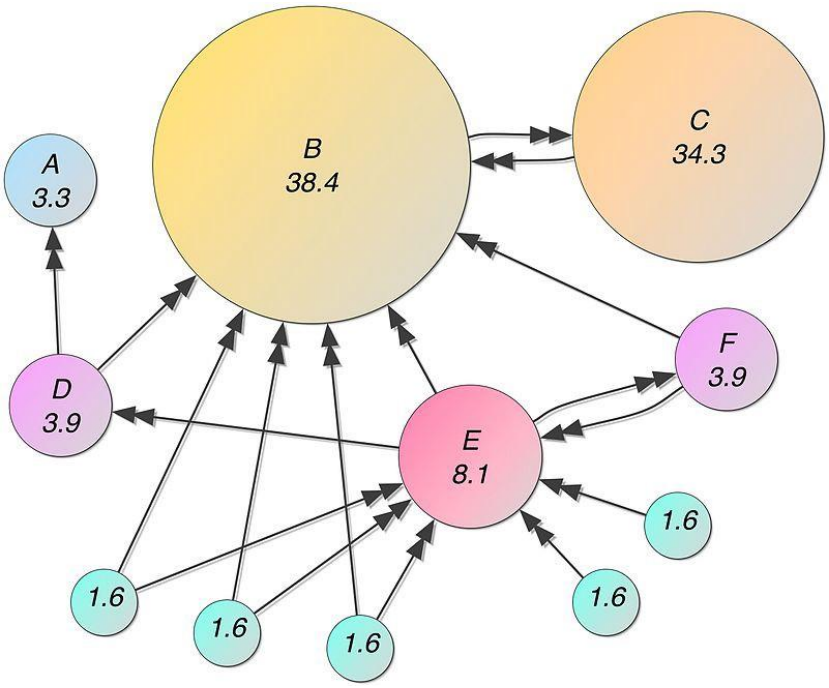

Рисунок 5 – Пример ранжирования сайта

Следует отметить, Google PageRank является вариантом меры центральности собственного вектора (Eigenvector Centrality) [13].

Центральность собственного вектора используется для измерения уровня влияния узла в сети. Каждому узлу в сети будет присвоена оценка или значение: чем выше оценка, тем выше уровень влияния в сети. Эта оценка относится к количеству соединений, которые узел будет иметь с другими узлами. Соединения с узлами центральности собственного вектора с высокой оценкой вносят больший вклад в оценку узла, чем равные соединения с узлами с низкой оценкой.

Чтобы поместить это в контекст, узел с высокой оценкой степени (то есть множеством соединений) может иметь только относительно низкую оценку центральности собственного вектора, потому что многие из этих

соединений имеют узлы с аналогичной низкой оценкой. Кроме того, узел может иметь высокий показатель промежуточности (что указывает на то, что он соединяет разрозненные части сети), но низкий показатель центральности собственного вектора, поскольку он все еще находится на некотором расстоянии от центров силы в сети.

Математически собственного вектора описывается следующим образом.

Пусть x_i – оценка центральности i-го узла. Обозначим матрицу смежности сети через $A_{i,j}$.

Тогда $A_{i,j}$ =1, если i-й узел находится рядом с j-м узлом и, $A_{i,j}$ =0 – в противном случае.

В более общем случае, записи в A могут быть действительными числами, представляющими силу соединения, как в стохастической матрице.

Пусть для i-го узла оценка центральности будет пропорциональна сумме оценок всех узлов, которые к нему подключены.

Следовательно, получим:

$$x_i(t) = \frac{1}{\lambda} \sum_{j \notin M(i)} x_i = \frac{1}{\lambda} \sum_{j=1}^{N} A_{i,j} x_i, \qquad (1)$$

где M(i) – множество узлов, подключенных к i–му узлу;

N – общее количество узлов;

λ – константа.

В векторной нотации получим:

$$\mathbf{x} = \frac{1}{\lambda} \mathbf{A}\mathbf{x} \qquad (2)$$

Как правило, будет много разных собственных значений центральности λ, для которых существует решение для собственного вектора (рисунок 6).

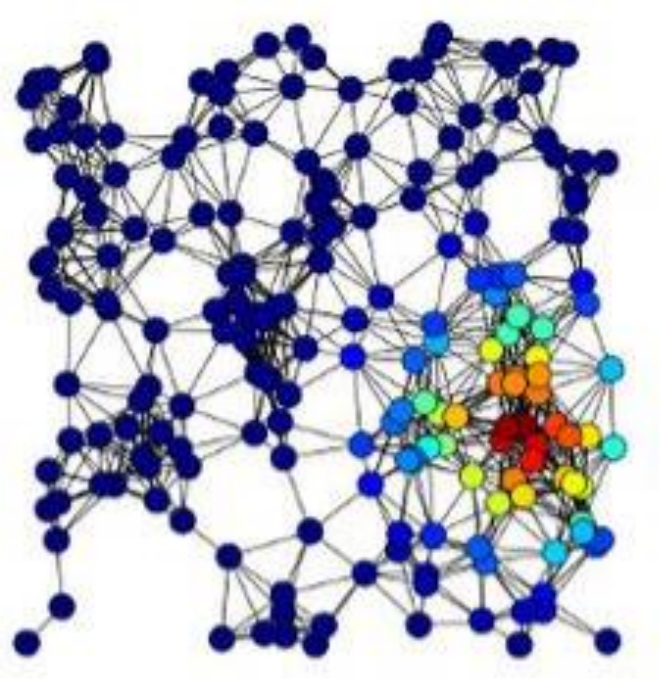

Рисунок 6 – Центральность по собственному вектору

Можно использовать центральность собственного вектора, чтобы определить, кто или что имеет широкое влияние в данной сети.

Как показывает практика, метод PageRank обеспечивает высокую эффективность при анализе графов методом их визуализации.

2.3 Меры центральности

В рамках теории графов и сетевого анализа существуют различные меры центральности вершины в графе, которые определяют относительную важность вершины в графе (например, насколько важен человек в социальной сети или в теории пространственного синтаксиса, насколько важна комната в здании или насколько хорошо используется дорога в городской сети) [1].

2.3.1 Центральность по степени

Анализ степени центральности **(**Degree centrality**)** является самым простым.

«Степень центральности определяется как число связей, приходящихся на узел (т.е. количество связей, которые имеет узел).

Степень часто интерпретируется с точки зрения непосредственного риска того, что узел может перехватить то, что проходит через сеть (например, вирус или некоторая информация).

Если сеть направлена (имеется в виду, что связи имеют направление), то обычно определяют две отдельные меры центральности степени:

- внутренние связи - это число связей, направленных на узел;
- внешние связи - количество связей, которые узел направляет другим.

Для позитивных отношений, таких как дружба или совет, обычно интерпретируют внутренние связи как форму популярности, а внешние связи – как общительность.

На рисунке 7 представлены примеры графов, узлы которых нагружены внутренними (а) и внешними связями (б)» [12].

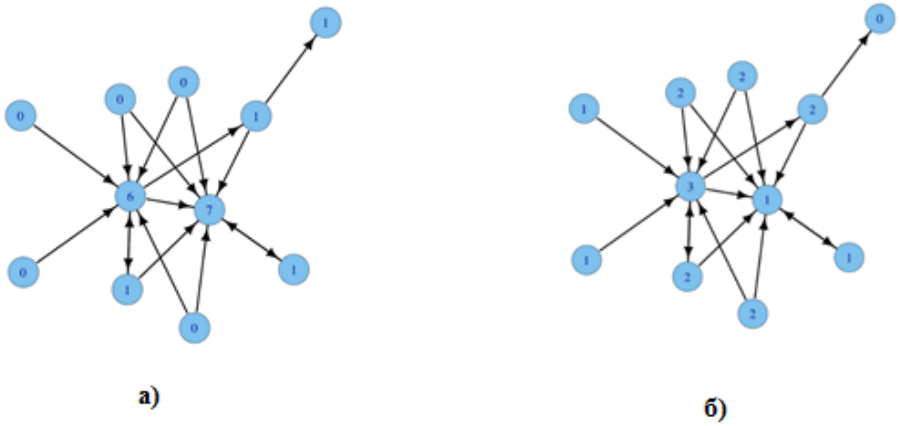

Рисунок 7 – Примеры графов, с узлами, нагруженные показателями степени центральности

Для графа, имеющего n вершин центральность по степени для вершины v определяется по формуле:

$$C_D(v) = \frac{deg(v)}{n-1}, \quad (3)$$

Временная сложность вычисления центральности по степени для всех узлов V в графе плотной матрицей смежности составляет $\Theta(V^2)$, а для всех ребер E в графе с разреженной матрицей смежности – $\Theta(E)$.

2.3.2 Центральность по близости

«Центральность по близости (Closeness centrality) является показателем, насколько быстро распространяется информация в сети от одного участника к остальным.

В топологии и смежных областях математики близость является одним из основных понятий в топологическом пространстве.

Интуитивно считаем, что два множества близко, если они произвольно близко друг к другу. Понятие может быть определено естественным образом в метрическом пространстве, где определено понятие расстояния между элементами пространства, но оно может быть обобщено на топологические пространства, где у нас нет конкретного способа измерения расстояний.

В теории графов близость является мерой центральности вершины в графе. Вершины, которые «неглубокие» по отношению к другим вершинам (то есть те, которые имеют тенденцию иметь короткие геодезические расстояния до других вершин в графе), имеют более высокую близость.

Близость предпочтительнее в сетевом анализе для обозначения длины кратчайшего пути, поскольку она дает более высокие значения большему количеству центральных вершин и поэтому обычно положительно связана с другими показателями, такими как степень (рисунок 8)» [1].

В теории сетей близость является сложной мерой центральности. Она определяется как среднее геодезическое расстояние (то есть кратчайший путь) между вершиной v и всеми другими достижимыми вершинами:

$$C_C(v) = \frac{n-1}{\sum_{t \in V \setminus v} d_G(v,t)}, \quad (4)$$

где $d_G(v,t)$ – кратчайший путь от вершины v до вершины t.

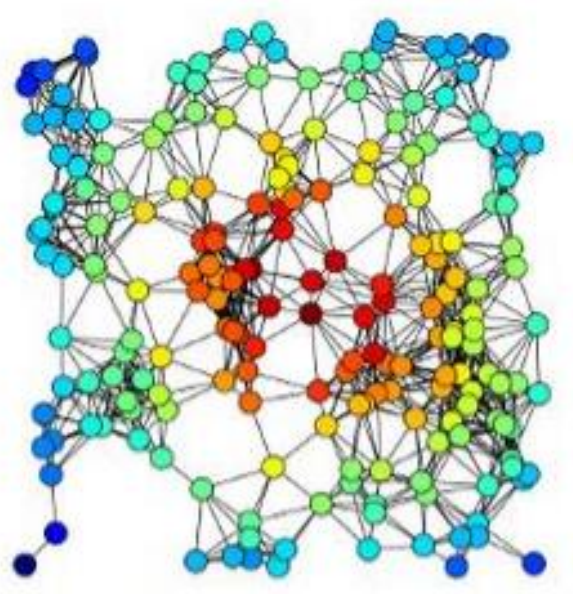

Рисунок 8 – Центральность по близости

Рассмотрение расстояния из или во все другие узлы неприменимо для неориентированных графов, тогда как в ориентированных графах они дают совершенно различные результаты.

Например, интернет-сайт может иметь высокую степень близости от исходящего соединения, но низкую степень близости от входящих соединений).

2.3.3 Посредническая центральность

«Посредническая центральность (Betweenness centrality) является мерой центральности вершины в графе.

Вершины, которые встречаются на многих кратчайших путях между другими вершинами, имеют более высокую промежуточность чем те, которые этого не делают.

Для графа с n вершинами промежуточность для вершины вычисляется следующим образом:

Шаг 1. Для каждой пары вершин (s, t) вычислим все кратчайшие пути между ними.

Шаг 2. Для каждой пары вершин (s, t) определим долю кратчайшие пути, которые проходят через рассматриваемую вершину (здесь вершина v).

Шаг 3. Суммируем эту долю по всем парам вершин (s, t).

Таким образом:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \quad (5)$$

где σ_{st} – общее количество кратчайших путей из вершины *s* к вершине *t*;

$\sigma_{st}(v)$ – количество кратчайших путей из вершины *s* к вершине *t*, проходящих через вершину *v*.

Для нормализации нужно разделить на количество пар вершин, за исключением самой вершины *v*, т. е. для ориентированного графа нужно разделить на (n-1)(n-2), для неориентированного – на величину, равную (n-1)(n-2)/2.

На рисунке 9 представлен пример посреднической центральности» [12].

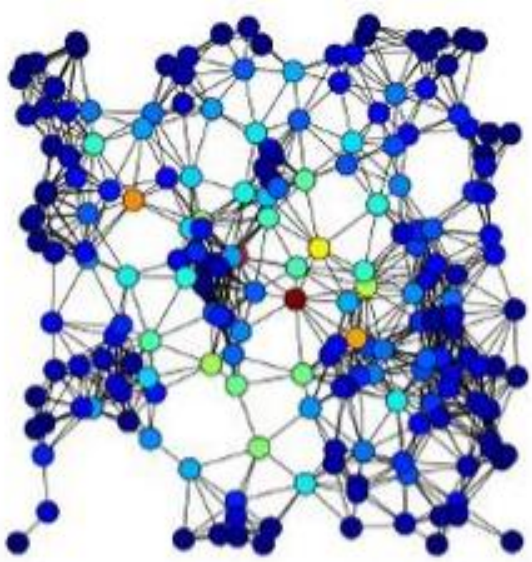

Рисунок 9 – Посредническая центральность

Недостатком посреднической центральности является ее вычислительная сложность.

2.3.4 Центральность Каца

В теории графов центральность Каца является обобщенной мерой центральности в сети (рисунок 10).

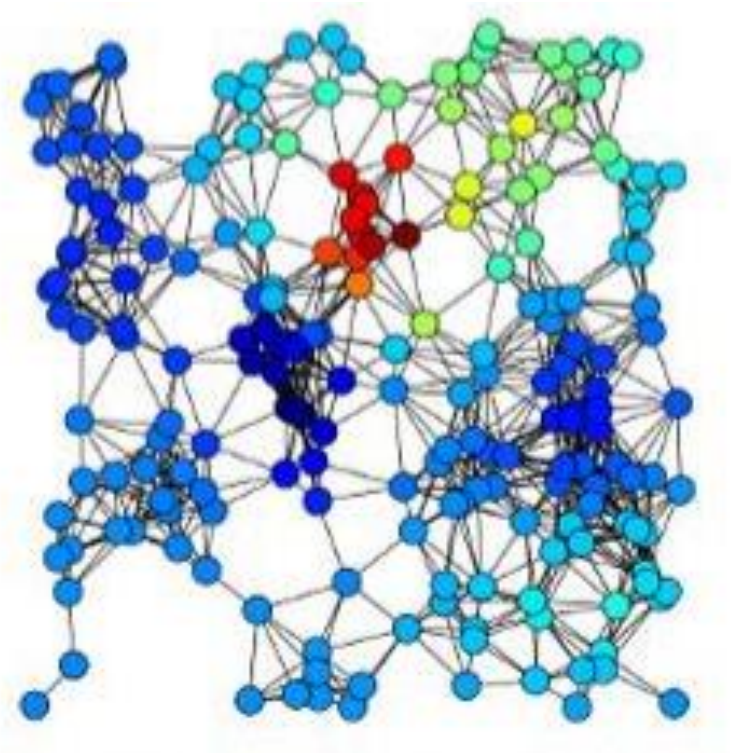

Рисунок 10 – Центральность Каца

«Данная мера была введена Лео Кацем в 1953 году и используется для измерения относительной степени влияния актера (или узла) в социальной сети [15].

В отличие от типичных мер центральности, которые рассматривают только кратчайший путь (геодезический) между парой действующих лиц, центральность Каца измеряет влияние, принимая во внимание общее количество прогулок между парой действующих лиц.

Пусть A – матрица смежности рассматриваемой сети.

Элементы a_{ij} матрицы A являются переменными, которые принимают значение 1, если узел i связан с узлом j, и 0 в противном случае.

Степени A указывают на наличие (или отсутствие) связей между двумя узлами через посредников.

Например, в матрице A^3, если элемент $a_{2,12} = 1$, это указывает, что узел 2 и узел 12 соединены через несколько соседей первой и второй степени узла 2» [12].

В формализованном виде центральность Каца определяется следующим образом:

$$C_{\text{Katz}}(i) = \sum_{k=1}^{\infty} \sum_{j=1}^{n} \alpha^k (A^k)_{ji}, \quad (6)$$

Следует отметить, что в приведенном выше определении используется тот факт, что элемент в расположении (i,j) матрицы A смежности, возведенный в степень k (A^k), отражает общее количество степеней k соединений между узлами i и j.

Для выбора алгоритма для анализа больших данных на графах используем сравнительную таблицу 2.

Таблица 2 – Сравнительный анализ алгоритмов анализа больших данных на графах

Характеристика/балл	Алгоритмы нахождения связанных компонентов	Алгоритм PageRank	Мера центральности
эффективность при анализе методами визуализации графов	1	3	3
производительность	2	2	2
простота реализации	3	2	2
Итого	6	7	7

Критерии оценивания:

0 – полное несоответствие требованиям;

1 – значительное несоответствие требованиям;

2 – незначительное несоответствие требованиям;

3 – полное соответствие требованиям.

По результатам сравнения выбираем в качестве алгоритмов анализа больших данных алгоритмы PageRank и меры центральности, как обеспечивающие высокую эффективность при анализе методами визуализации графов.

Выводы по главе 2

Вторая глава бакалаврской работы посвящена обзору и анализу алгоритмов анализа больших данных на графах.

Результаты проделанной работы позволили сделать следующие выводы:

1. К наиболее важным алгоритмам графов, которые используются для анализа больших данных, относятся алгоритмы нахождения связанных компонентов в графе, алгоритмы PageRank и меры центральности.

2. Как показал сравнительный анализ, высокую эффективность при анализе данных методами визуализации графов обеспечивают алгоритмы PageRank и меры центральности.

Глава 3 Программное обеспечение для анализа больших данных методами визуализации графов

Рассмотрим пример применения алгоритма PageRank, используя демонстрационные данные сети Facebook [4].

Для программирования используем язык Python [7].

Имеется файл ребер/ссылок между пользователями Facebook. Сначала мы создаем граф FB, используя код, представленный на рисунке 11.

```
import os
print(os.listdir('../input/'))
['facebook-combined.txt']
# reading the dataset
fb = nx.read_edgelist('../input/facebook-combined.txt', create_using =
nx.Graph(), nodetype = int)
print(nx.info(fb))
pos = nx.spring_layout(fb)
import warnings
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (20, 15)
plt.axis('off')
nx.draw_networkx(fb, pos, with_labels = False, node_size = 35)
plt.show()
```

Рисунок 11 – Код создания графа FB

Полученный пользовательский граф FB показан на рисунке 12.

Допустим, что необходимо найти пользователей с высоким потенциалом влияния.

Интуитивно алгоритм Pagerank даст более высокий балл пользователю, у которого много друзей, у которых, в свою очередь, тоже много друзей в

Фейсбуке (рисунок 13).

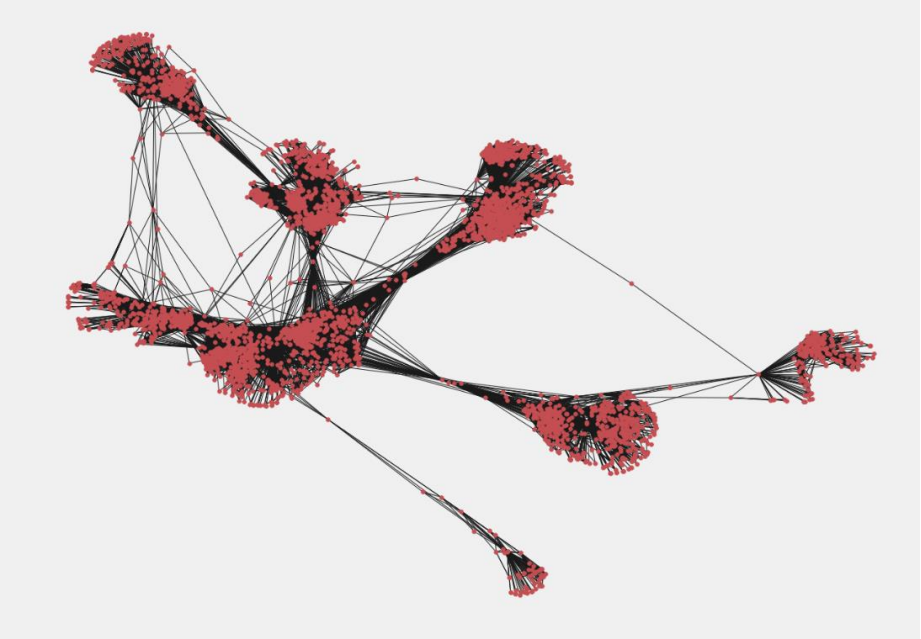

Рисунок 12 – Пользовательский граф FB

```
pageranks = nx.pagerank(fb)
print(pageranks)
import operator
sorted_pagerank = sorted(pagerank.items(),
key=operator.itemgetter(1),reverse=True)
print(sorted_pagerank[:5])
```

Рисунок 13 - Код реализации алгоритма Pagerank

Можно получить отсортированный PageRank или наиболее влиятельных пользователей, используя код, представленный на рисунке 14.

```
import operator
sorted_pagerank = sorted(pageranks.items(),
key=operator.itemgetter(1),reverse = True)
print(sorted_pagerank)
----------
[(3437, 0.007614586844749603), (107, 0.006936420955866114), (1684,
0.006367162138306829 5), (0, 0.006289602618466542), (1912,
0.0038769716008844974), (348, 0.0023480969727805783), (686,
0.0022193592598000193), (3980, 0.002170323579009993), (414,
0.0018002990470702262), (698, 0.0013171153138368807), (483,
0.0012974283300616082), (3830, 0.0011844348977671688), (376,
0.0009014073664792464), (2047, 0.000841029154597401), (56,
0.0008039024292749443), (25, 0.000800412660519768), (828,
0.0007886905420662135), (322, 0.0007867992190291396),.....]
```

Рисунок 14 – Код сортировки PageRank

Вышеуказанные идентификаторы предназначены для самых влиятельных пользователей.

Выделяем подграф для самого влиятельного пользователя с помощью кода, представленного на рисунке 15.

```
first_degree_connected_nodes = list(fb.neighbors(3437))
second_degree_connected_nodes = []
for x in first_degree_connected_nodes:
second_degree_connected_nodes+=list(fb.neighbors(x))
second_degree_connected_nodes.remove(3437)
second_degree_connected_nodes =
list(set(second_degree_connected_nodes))
subgraph_3437 =
nx.subgraph(fb,first_degree_connected_nodes+second_degree_connected_nodes,)
```

Рисунок 15 – Код выделения подграфа самого влиятельного пользователя

```
In [15]:
pos = nx.spring_layout(subgraph_3437)
In [16]:
linkcode
import warnings
warnings.filterwarnings('ignore')
node_color = ['yellow' if v == 3437 else 'red' for v in subgraph_3437]
node_size =  [1000 if v == 3437 else 35 for v in subgraph_3437]
plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (20, 15)
plt.axis('off')
nx.draw_networkx(subgraph_3437, pos, with_labels = False,
node_color=node_color,node_size=node_size )
plt.show()
```

Продолжение рисунка 15

На рисунке 16 самый влиятельный пользователь обозначен желтой точкой.

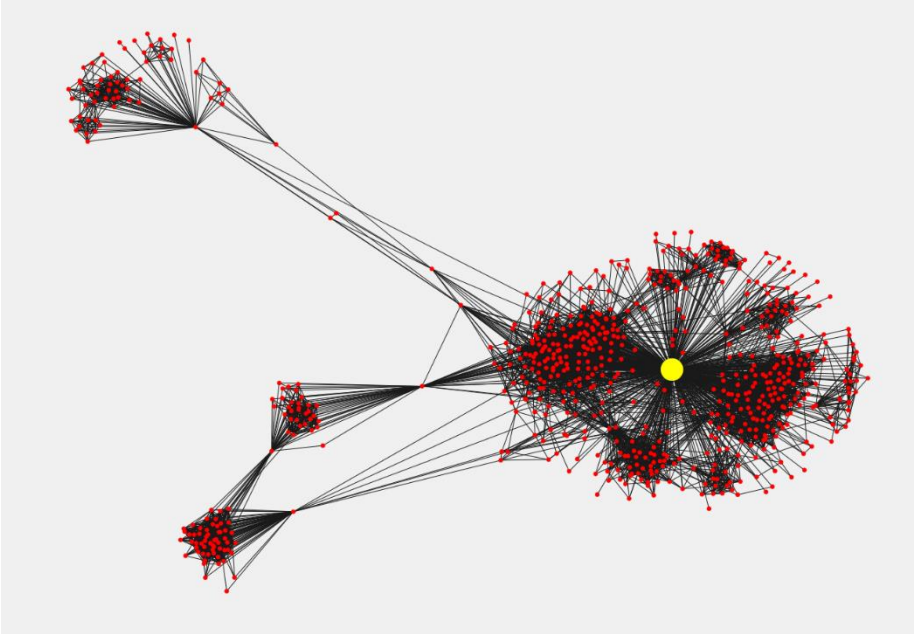

Рисунок 16 – Выделение самого влиятельного пользователя

Рассмотрим примеры использования критериев централизации.

Как было отмечено выше, в посреднической центральности важны не только пользователи, у которых больше всего друзей, но также и пользователи, которые связывают одну географию с другой, поскольку это позволяет пользователям видеть контент из разных географических регионов.

Посредническая центральность количественно определяет, сколько раз конкретный узел проходит по кратчайшему выбранному пути между двумя другими узлами.

Напомним также, что степень центральности - это просто количество соединений для узла.

Следует также учесть, что меры центральности можно использовать как функцию в любой модели машинного обучения.

На рисунке 17 представлен код для определения посреднической центральности для подграфа.

```
pos = nx.spring_layout(subgraph_3437)
betweennessCentrality =
nx.betweenness_centrality(subgraph_3437,normalized=True,
endpoints=True)
node_size =  [v * 10000 for v in betweennessCentrality.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(subgraph_3437, pos=pos, with_labels=False,
                 node_size=node_size )
plt.axis('off')
```

Рисунок 17 – Код определения посреднической центральности для подграфа

На рисунке 18 можно увидеть размеры узлов по их промежуточным

значениям центральности (выделены красным цветом).

Их можно рассматривать как распространителей информации.

Нарушение любого из узлов с высокой промежуточностью центральности разбивает граф на множество частей.

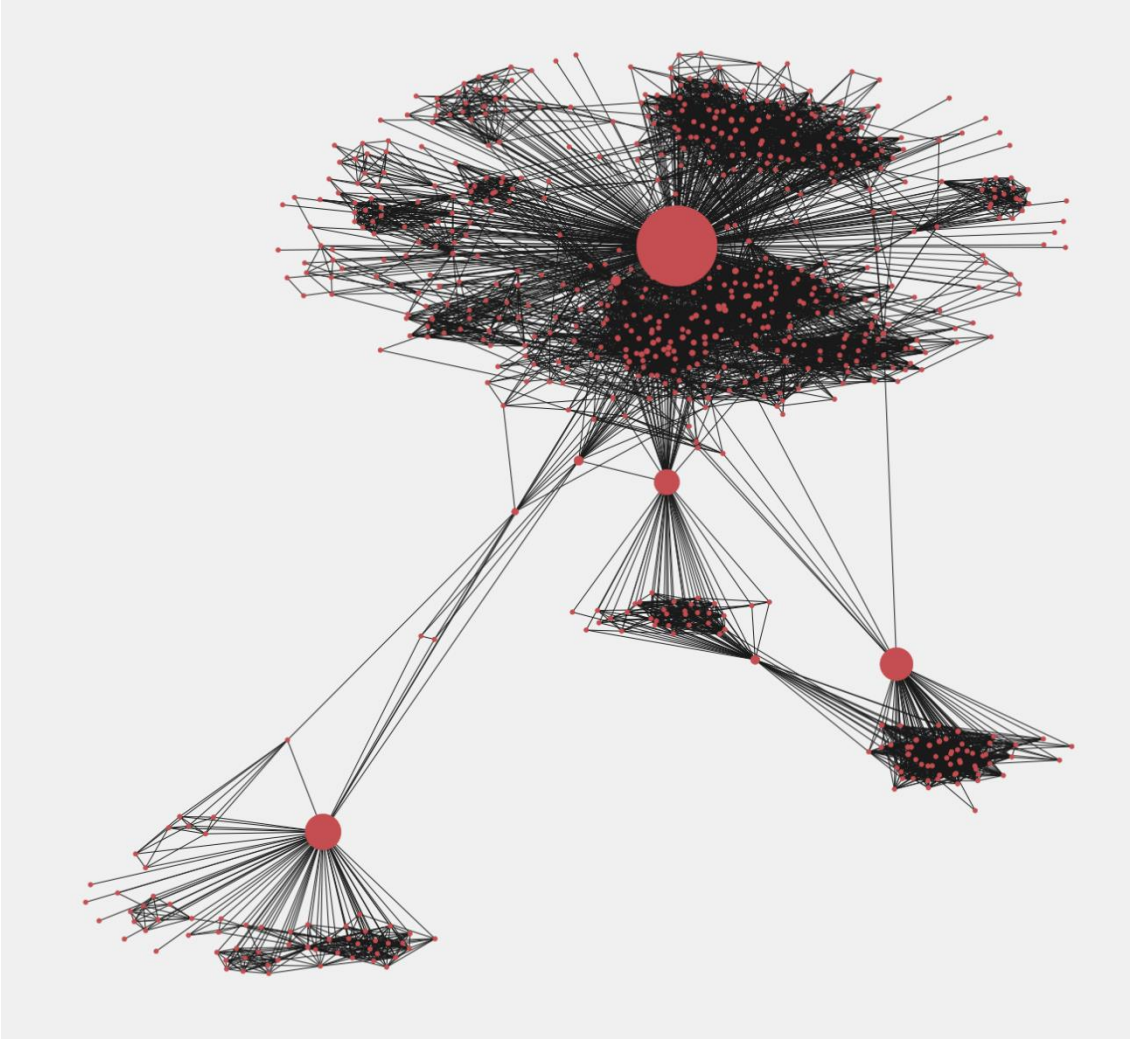

Рисунок 18 – Определение посреднической центральности для подграфа

Рассмотрим пример анализа данных с помощью центральности по степени [19].

Соответствующий программный код имеет вид, представленный на рисунке 19.

```
In [4]:
pos = nx.spring_layout(G1)
degCent = nx.degree_centrality(G1)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in degCent.values()]
plt.figure(figsize=(15,15))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(degCent, key=degCent.get, reverse=True)[:5]
Out[4]:
[107, 1684, 1912, 3437, 0]
In [5]:
sorted(degCent, key=degCent.get, reverse=True)[:5]
Out[5]:
[107, 1684, 1912, 3437, 0]
```

Рисунок 19 – Код анализа данных с помощью центральности по степени

Результирующие подграфы в виде цветных пятен представлены на рисунке 20.

Соответствующий программный код представлен на рисунке 21.

Рисунок 20 – Определение центральности по степени для подграфа

```
In [6]:
pos = nx.spring_layout(G1)
cloCent = nx.closeness_centrality(G1)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in cloCent.values()]
plt.figure(figsize=(13,13))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(cloCent, key=cloCent.get, reverse=True)[:5]
Out[6]:
[107, 58, 428, 563, 1684]
```

Рисунок 21 – Код определения центральности по близости

Результат анализа представлен на рисунке 22.

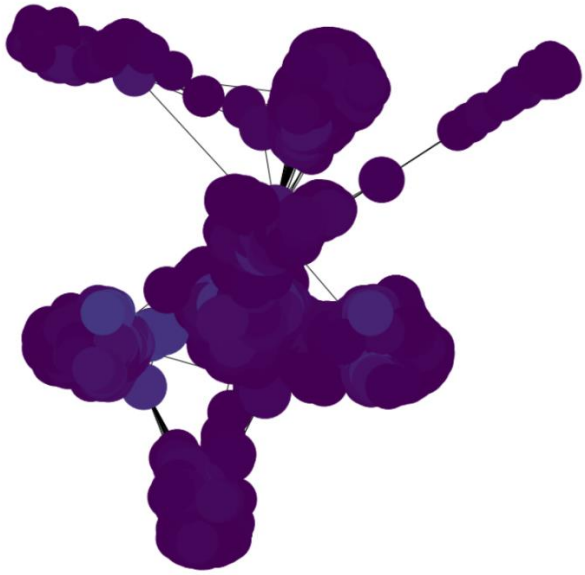

Рисунок 22 – Определение центральности по близости

Рассмотрим пример определения центральности собственного вектора. Пример программного кода приведен на рисунке 23.

```
In [88]:
#pos = nx.spring_layout(G1)
eigCent = nx.eigenvector_centrality(G1)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in eigCent.values()]
plt.figure(figsize=(15,15))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
/usr/local/lib/python3.6/dist-packages/networkx/drawing/nx_pylab.py:579:
MatplotlibDeprecationWarning:
```

Рисунок 23 – Код определения центральности по близости

```
if not cb.iterable(width):
```

```
Out[88]:
(-0.5881275936731919,
 0.8545417764248525,
 -1.0872190971973008,
 0.8315779455522556)
In [89]:
sorted(eigCent, key=eigCent.get, reverse=True)[:5]
Out[89]:
[1912, 2266, 2206, 2233, 2464]
In [ ]:
list(nx.algorithms.community.k_clique_communities(G1,k=2))
In [ ]:
list(nx.algorithms.core.k_core(G1,k=3))
In [5]:
#nx.algorithms.link_analysis.pagerank_alg.pagerank(G1)
309: 0.0001716312351638319, 310: 0.0002748900516265396, 311:
0.0001138944868967976, 312: 0.000512080095393576, 313:
0.0004092399446675024, 314: 0.0001724333430201253, 315:
0.0006036352914088951, 316: 6.581490012111381e-05, 317:
0.0001189454468768049 5, 318: 0.0001468958899178936 8, 319:
0.0002288806369282809, 320: 0.0003079052496118058, 321:
9.264483661467496e-05, 322: 0.0007867992190291396, 323:
0.0004294328223425148, 324: 0.0003296617518206686, 325:
0.0004112008723720156, 326: 0.0003573420959917144, 327:
0.0001275343277989076, 328: 0.0002169317012418637 7, 329:
0.0003988721550833793 4, 330: 0.0002273449678342628 6, 331:
0.0002408747578231588 5, 332: 0.0004759881107568617, 333:
0.0001712234624362550 3, 334: 0.0003178367843185237 5, 335:
5.2591423276218314e-05, 336: 7.450715465063965e-05, 337:
```

Продолжение рисунка 23

```
0.00020454700787763102, 338: 0.00011254118253680223, 339:
0.00035337088392488884,
In [17]:
nx.degree_histogram(G1)
Out[17]:
6
In [16]:
nx.algorithms.graph_clique_number(G1)
----> 1 nx.algorithms.graph_clique_number(G1)
/usr/local/lib/python3.6/dist-packages/networkx/algorithms/clique.py in
graph_clique_number(G, cliques)
   426     if len(G.nodes) < 1:
   427         return 0
--> 428     return max([len(c) for c in cliques] or [1])
   429
   430
/usr/local/lib/python3.6/dist-packages/networkx/algorithms/clique.py in
<listcomp>(.0)
   426     if len(G.nodes) < 1:
   427         return 0
--> 428     return max([len(c) for c in cliques] or [1])
   429
   430
/usr/local/lib/python3.6/dist-packages/networkx/algorithms/clique.py in
find_cliques(G)
   199                     subg = subg_q
   200                     cand = cand_q
--> 201                     u = max(subg, key=lambda u: len(cand & adj[u]))
   202                     ext_u = cand - adj[u]
   203         else:  KeyboardInterrupt:
```

Продолжение рисунка 23

Результирующий граф представлен на рисунке 24.

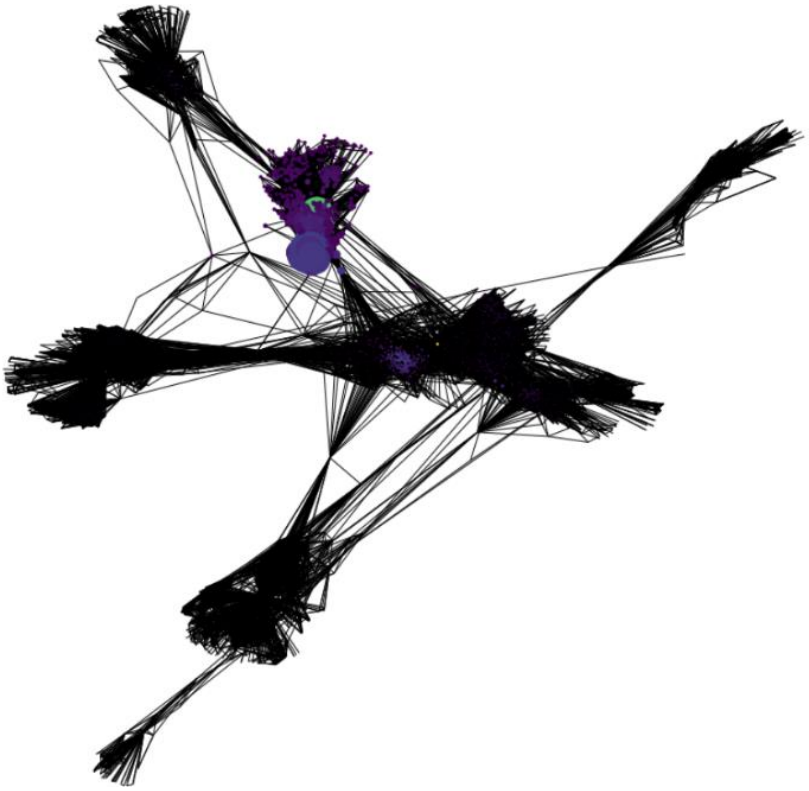

Рисунок 24 – Определение центральности по собственному вектору

Таким образом, представленные алгоритмы позволяют решать задачи анализа больших данных методами визуализации графов.

Выводы по главе 3

Третья глава посвящена разработке программного обеспечения для анализа больших данных методами визуализации графов.

Результаты проделанной работы позволили сделать следующие выводы:

1. Для разработки программного обеспечения анализа данных больших данных методами визуализации графов более предпочтительным является использование языка Python.

2. Реализованные алгоритмы позволяют решать задачи анализа больших данных методами визуализации графов.

Заключение

Выпускная квалификационная работа посвящена актуальной проблеме применения методов обработки графов для повышения эффективности анализа больших данных.

Для решения задач анализа больших данных требуются новые парадигмы, методы и алгоритмы, которые позволяют эффективно обрабатывают большие объемы данных, используя их структуру.

Естественной формой представления структурированных или частично структурированных данных являются графы.

Применение методов обработки графов для задач анализа больших данных представляет научно-практический интерес.

1. Выполнены обзор и анализ методов обработки больших графов. Для решения задач анализа больших данных с помощью аппарата графов используется технология интеллектуального анализа графов или Graph Mining. Как показал анализ, одной из самых сложных задач в интеллектуальном анализе графов является анализ шаблонов в больших графах.

2. Выделены следующие методы обработки больших графов для последующего анализа: сопоставление шаблонов подграфов, модели вычислений на больших графах и визуализация графов. Результаты сравнительного анализа показали, что наилучшими возможностями для аналитической обработки больших графов обладает метод визуализации графов. Визуализация графов позволяет получить огромные преимущества от выхода за рамки плоской модели данных с помощью мощного программного обеспечения для визуализации.

3. Проанализированы алгоритмы анализа больших данных на графах. К наиболее важным алгоритмам графов, которые используются для анализа больших данных, относятся алгоритмы нахождения связанных компонентов в графе, алгоритмы PageRank и меры центральности. Как показал

сравнительный анализ, высокую эффективность при анализе данных методами визуализации графов обеспечивают алгоритмы PageRank и различные меры центральности вершины в графе, которые определяют относительную важность вершины в графе.

4. Разработано программное обеспечение для анализа больших данных методами визуализации графов. Для разработки программного обеспечения использован язык Python. Реализованы алгоритмы, основанные на различных мерах центральности. Тестирование программы показало, что реализованные в ней алгоритмы позволяют решать задачи анализа больших данных методами визуализации графов.

Результаты бакалаврской работы представляют научно-практический интерес и могут быть рекомендованы для разработчиков программ, основанных на методах обработки графов для задач анализа больших данных.

Список используемой литературы

1. Батура Т. В. Модели и методы анализа компьютерных социальных сетей // Программные продукты и системы. 2013. №3 (103). С. 130-137.

2. Поиск компонент сильной связности: алгоритм Косарайю [Электронный ресурс]. URL: https://habr.com/ru/post/331904/ (дата обращения: 14.05.2021).

3. Распределенные вычисления в облаке: GraphLab [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/learn/modules/cmu-analytics-engines-graphlab/ (дата обращения: 14.05.2021).

4. Сайт Kaggle [Электронный ресурс]. URL: https://www.kaggle.com/ (дата обращения: 20.05.2021).

5. Фролов А. С., Семенов А. С., Марков А. С. Обзор инструментальных средств разработки параллельных графовых приложений для суперкомпьютерных комплексов. Comp. nanotechnol., 2015, выпуск 4, 6–17.

6. Черемисинов Д.И., Черемисинова Л.Д. Задача обработки больших графов (Graph mining) // Big Data and advance analytics. №5. 2019. С. 328-333.

7. Язык программирования Python [Электронный ресурс]. URL: https://intuit.ru/studies/courses/49/49/info (дата обращения: 20.05.2021).

8. Aridhi S., Engelbert M.N. Big Graph Mining: Frameworks and Techniques, Big Data Research, vol. 6, 2016. P. 1-10.

9. Bulk synchronous parallel [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Bulk_synchronous_parallel (дата обращения: 14.05.2021).

10. Chakrabarti D. Graph Mining, Encyclopedia of Machine Learning, 2010, Springer US, P. 469-471.

11. Data Scientists, The 5 Graph Algorithms that you should know [Электронный ресурс]. URL: https://towardsdatascience.com/data-scientists-the-five-graph-algorithms-that-you-should-know-30f454fa5513 (дата обращения: 14.05.2021).

12. Degree Centrality [Электронный ресурс]. URL: https://www.sci.unich.it/~francesc/teaching/network/degree.html (дата обращения: 15.05.2021).

13. Eigenvector Centrality [Электронный ресурс]. URL: https://www.sci.unich.it/~francesc/teaching/network/eigenvector.html (дата обращения: 15.05.2021).

14. Hu Y., Nöllenburg M., Sakr S., Zomaya A.Y. Graph Visualization, Encyclopedia of Big Data Technologies, Springer International Publishing, 2019.

15. Katz Centrality (Centrality Measure) [Электронный ресурс]. URL: https://www.geeksforgeeks.org/katz-centrality-centrality-measure/ (дата обращения: 15.05.2021).

16. Kosaraju's Algorithm for Strongly Connected Components [Электронный ресурс]. URL: https://iq.opengenus.org/kosarajus-algorithm-for-strongly-connected-components/ (дата обращения: 14.05.2021).

17. Minimum cut [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Minimum_cut#:~:text=In%20graph%20theory%2C%20a%20minimum,into%20more%20than%20two%20sets (дата обращения: 14.05.2021).

18. Page Rank Algorithm and Implementation [Электронный ресурс]. URL: https://www.geeksforgeeks.org/page-rank-algorithm-implementation/ (дата обращения: 15.05.2021).

19. Social Network Analysis in Python [Электронный ресурс]. URL: https://github.com/miladfa7/Social-Network-Analysis-in-Python (дата обращения: 20.05.2021).

20. Vertex-centric graph processing [Электронный ресурс]. URL: https://www.waitingforcode.com/graphs/vertex-centric-graph-processing/read (дата обращения: 14.05.2021).

21. Zadeh Fard A.J., Nisar U., Miller J.A. Techniques for Graph Analytics on Big Data, IEEE International Congress on Big Data? 2013.