МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

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 Разработка алгоритма анализа транспортных схем для выявления отклонений от идеального маршрута

Обучающийся Е.А. Крайнов _____
(Инициалы Фамилия) (личная подпись)

Руководитель канд. пед. наук, доцент, Т.А. Агошкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. филол. наук, М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Бакалаврская работа посвящена разработке алгоритма анализа транспортных схем для выявления отклонений от идеального маршрута с использованием графового подхода. Изучены методы анализа транспортных систем, графовые алгоритмы и их применение для оптимизации маршрутов в городах. Разработан алгоритм на основе метода Дейкстры, реализованный на Python с использованием библиотек OSMnx, NetworkX и Matplotlib, и протестирован на данных дорожной сети Москвы.

Особое внимание уделено устойчивости алгоритма к неполноте и неточностям входных данных. Рассмотрены типовые ошибки построения маршрутов и предложены способы их автоматического обнаружения. Показана возможность интеграции алгоритма в системы умного города.

Во введении обоснована актуальность темы, связанная с повышением эффективности транспортного планирования, определены цель и задачи.

Первый раздел описывает современные подходы к моделированию транспортных сетей, математические основы алгоритмов поиска путей и их роль в оптимизации.

Во втором разделе представлена математическая модель сети как ориентированного графа, описана реализация алгоритма и его настройка для анализа маршрутов. В третьем разделе проанализированы результаты на примере маршрута от Красной площади до Москва-Сити, проведено сравнение с идеальными траекториями, оценены точность и производительность алгоритма, предложены улучшения.

В заключении подведены итоги, подтверждена применимость алгоритма и намечены направления дальнейших исследований.

Бакалаврская работа состоит из введения, трёх разделов, заключения и списка использованной литературы.

Объём работы: 46 страницы, 11 рисунков, 2 таблицы, 25 источников, 1 Приложение.

Abstract

The title of the graduation work is *Developing an algorithm for analyzing the transport schemes to identify the deviations from the ideal route.*

The graduation work consists of an introduction, 3 parts, 11 figures, 2 tables, a conclusion, and a list of 25 references.

The aim of this graduation work is to develop and implement an algorithm for detecting the deviations from the optimal city routes using graph theory methods.

The object of the research is the transport system of a modern megalopolis.

The subject of the study is the algorithm for analyzing and correcting the transport routes using the *Dijkstra's algorithm* and the Python tools.

The key issue of the graduation work is to ensure the route analysis stability and accuracy under the conditions of incomplete or erroneous input data.

The graduation work may be divided into several logically connected parts which are: literature review, algorithm development and implementation, and an experimental result analysis.

The first part describes the current approaches to modeling the transport networks, the graph algorithms and their role in optimization.

The second part presents the transport network mathematical model as a directed graph and covers the developed algorithm based on the Dijkstra's algorithm. This part also dwells on the implementation using Python, OSMnx, and NetworkX.

The third part is devoted to the experimental evaluation of the algorithm using the data related to the Moscow road network. The route from the Red Square to Moscow City is analyzed. The results obtained are compared with the ideal trajectories.

In conclusion, the algorithm applicability for the smart city systems is confirmed, and the directions for further research are proposed.

The work is of interest for researchers and specialists involved in transport planning and smart city development.

Содержание

Введение

Современные транспортные системы играют ключевую роль в обеспечении мобильности населения, развитии экономики и повышении качества жизни в городах. Однако их сложность, обусловленная высокой плотностью дорожных сетей, динамикой трафика и внешними факторами, такими как пробки или перекрытия, требует применения эффективных методов анализа и моделирования. В условиях крупных мегаполисов, таких как Москва, где транспортная сеть включает тысячи узлов и рёбер, традиционные подходы к планированию маршрутов и управлению потоками часто оказываются недостаточными. Это подчёркивает актуальность разработки новых инструментов, способных не только определять оптимальные пути, но и выявлять отклонения от них, моделируя реальные сценарии движения.

Настоящая бакалаврская работа посвящена изучению и применению методов анализа транспортных схем с использованием графового подхода. Графовые методы выбраны в качестве основы благодаря их способности представлять транспортную сеть как формализованную структуру, состоящую из узлов (перекрестков, пунктов назначения) и рёбер (дорог), что позволяет применять математические алгоритмы для решения задач оптимизации и диагностики. В рамках работы особое внимание уделено алгоритму Дейкстры, который обеспечивает поиск кратчайшего пути в графе, а также моделированию неоптимальных маршрутов, отражающих возможные отклонения от идеальных траекторий. Актуальность исследования обусловлена необходимостью повышения эффективности транспортного планирования, минимизации затрат времени и ресурсов, а также подготовки к дальнейшему углублённому анализу транспортных систем в рамках выпускной работы, основываясь на реальных данных городской инфраструктуры.

Целью бакалаврской работы является изучение теоретических основ и практическая реализация алгоритма анализа транспортных схем, позволяющего выявлять отклонения от оптимальных маршрутов на основе графового представления сети. Для достижения этой цели в ходе работы были поставлены следующие задачи:

- провести обзор математических моделей и алгоритмов, применяемых для анализа транспортных схем, с акцентом на графовые методы;
- ознакомиться с разработкой математической модели и алгоритма для анализа структуры сети и моделирования отклонений, используя реальные данные дорожной сети Москвы;
- применить изученный алгоритм к конкретному примеру маршрута от Красной площади до Москва-Сити, сравнить полученные результаты с идеальными маршрутами и оценить эффективность предложенного подхода.

В ходе бакалаврской работы используются данные OpenStreetMap [21], загружаемые через библиотеку OSMnx [13], что обеспечивает доступ к актуальной информации о дорожной сети. Реализация алгоритма выполнена на языке Python [19][5] с применением библиотек NetworkX [12] и Matplotlib для обработки графов и визуализации результатов. Теоретическая значимость моей работы заключается в закреплении знаний о применении графовых методов к транспортным задачам, а практическая – в освоении инструмента, который может быть использован для анализа и оптимизации маршрутов в реальных условиях, а также станет основой для дальнейшей разработки в рамках бакалаврской работы. Структура работы включает три основных части: обзор методов и моделей, изучение алгоритма и его применение с последующим анализом результатов, что отражает процесс подготовки к итоговому исследованию.

1 Обзор методов и алгоритмов анализа транспортных схем

1.1 Анализ существующих методов анализа транспортных схем

Анализ транспортных схем является важной частью транспортного планирования, он помогает как оптимизировать движение транспорта, улучшить безопасность и повысить эффективность транспортных систем. Существует много методов анализа транспортных схем (рисунок 1), которые можно разделить на несколько категорий. Для более глубокого понимания их сути ниже приведено описание каждого метода, которое раскрывает основные принципы и подходы к моделированию транспортных схем.

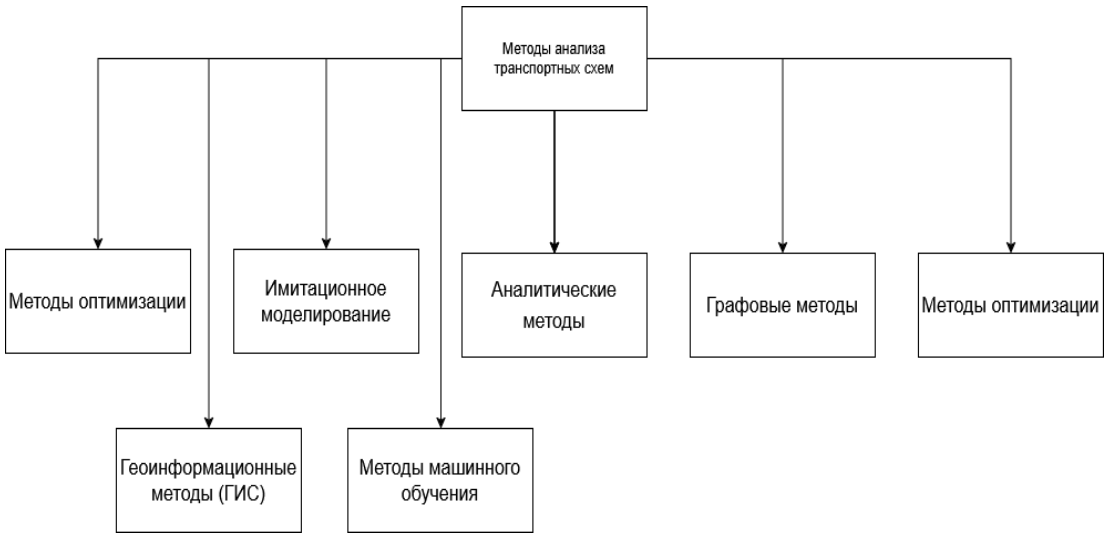

Рисунок 1 – Общая схема методов анализа транспортных схем

Аналитические методы являются методами моделирования транспортных схем, которые связаны с использованием строгих математических уравнений и формул. Они рассчитаны на описание характеристик транспортной сети, таких как пропускная способность дорог, связность между элементами дорог и оптимальными маршрутами движения. Фундаментальные законы математики лежат в основе этих подходов, так как они позволяют описывать характеристические схемы в виде аналитических

зависимостей. Например, используются уравнения, которые моделируют транспортный поток в зависимости от ширины дороги, длины участка или времени задержки на перекрестке. Часто применяются такие подходы, как теория очередей, моделирующая транспортные объекты как системы с ожиданием, или принципы равновесия, согласно которым участники движения выбирают маршруты с минимальными затратами.

Графовые методы основаны на представлении транспортной схемы в виде математического графа – структуры, состоящей из узлов и ребер. Узлы соответствуют ключевым точкам сети (перекресткам, развязкам, остановкам, конечным пунктам назначения), а ребра обозначают соединяющие их участки дорог. Каждое ребро может характеризоваться параметрами, такими как длина, ширина, пропускная способность и допустимая скорость движения. Для анализа этой структуры применяются алгоритмы теории графов, которые позволяют исследовать топологию сети и ее свойства, например поиск кратчайшего пути, анализ связности узлов или вычисление минимального покрытия сети. [19]

Имитационное моделирование представляет собой создание компьютерной модели транспортной схемы, воспроизводящей ее работу в виртуальной среде. Этот метод делится на три уровня детализации. Микроуровень моделирует поведение отдельных участников движения (автомобилей, пешеходов и других объектов) [20] с учетом их индивидуальных характеристик, таких как скорость, ускорение и правила взаимодействия на дороге. Мезоуровень рассматривает транспорт как совокупность объектов, объединяя их в потоки с общими свойствами, что снижает детализацию по сравнению с микроуровнем, но сохраняет учет локальных особенностей. Макроуровень представляет сеть в упрощенном виде, где транспортные потоки описываются как непрерывные величины (аналогично жидкости или газу) с параметрами плотности и скорости [20]. Все уровни используют программные платформы, позволяющие задавать

структуру схемы, включая дороги, перекрестки и регулирующие элементы, а затем симулировать их работу в различных условиях. [20]

Геоинформационные методы основаны на использовании специализированных систем, которые интегрируют пространственные данные для моделирования транспортных схем. В таких системах сеть представлена как совокупность объектов, привязанных к географическим координатам, что позволяет учитывать их реальное расположение. Дороги, перекрестки и другие элементы схемы описываются не только с точки зрения их связности, но и с учетом пространственных характеристик (длина, ширина, высота, ориентация относительно сторон света). Эти методы используют цифровые карты и базы данных с информацией о геометрии сети и ее окружении (рельеф, здания, природные объекты). Анализ выполняется с помощью инструментов обработки пространственной информации, визуализации сети и выявления ее структурных особенностей.

Методы оптимизации направлены на поиск наилучших конфигураций транспортной схемы с использованием математических техник. Они формализуют структуру сети и ее параметры в виде целевой функции, которую необходимо минимизировать или максимизировать (например, длину маршрутов, время движения или затраты на обслуживание). [19] Для этого применяются: Линейное программирование, решающее задачи с линейными зависимостями между переменными. Нелинейные методы, включая генетические алгоритмы, которые имитируют эволюционные процессы, перебирая возможные варианты структуры сети и выбирая наиболее эффективные. В этих методах транспортная схема рассматривается как система с заданными ограничениями (например, максимальная ширина дороги или бюджет строительства), а результатом становится оптимальная версия сети, соответствующая выбранным критериям.

Методы машинного обучения [6] используют алгоритмы искусственного интеллекта для анализа структуры транспортной схемы на основе больших объемов данных. В отличие от традиционных подходов, где

правила задаются вручную, модели машинного обучения [6] обучаются самостоятельно выявлять закономерности и особенности сети. Например: Нейронные сети анализируют сложные взаимосвязи между элементами схемы (расположение дорог, перекрестков и их характеристики), выявляя скрытые зависимости. Кластеризация позволяет группировать участки сети по схожим признакам. Алгоритмы классификации могут определять типы элементов схемы или их состояние. [19]

Эти методы требуют предварительной подготовки данных (например, топологии сети) и используют статистические и вычислительные подходы для построения моделей, которые затем применяются для анализа транспортных схем.

Для оценки различных методов анализа транспортных схем можно использовать следующие критерии:

- описание - краткое изложение того, как работает алгоритм. Он включает в себя основные принципы и механизмы, которые лежат в основе алгоритма, а также его цели. Описание помогает понять, что именно делает алгоритм и как он подходит для решения определенных задач;
- применение демонстрирует практическую ценность алгоритма и помогает определить, в каких ситуациях его использование будет наиболее эффективным;
- преимущества - сильные стороны алгоритма, его положительные аспекты и уникальные качества, которые делают его предпочтительным выбором для определенных задач;
- недостатки - ограничения и слабые стороны алгоритма.

В таблице 1 представлено сравнение методов анализа транспортных схем.

Таблица 1 - Сравнительная таблица методов анализа транспортных схем

Метод	Описание	Применение	Преимущества	Недостатки
Аналитические методы	Математические модели (уравнения, теория очередей) для анализа схемы	Расчет пропускной способности, узкие места	Высокая точность при хороших данных	Сложность при больших системах
Графовые методы	Схема как граф (узлы и ребра), алгоритмы поиска путей	Построение маршрутов, анализ связности	Простота и универсальность	Не учитывает динамику без доработки
Имитационное моделирование	Компьютерная симуляция (микро-, мезо-, макроуровень)	Тестирование изменений схемы, влияние на трафик	Реалистичность, учет поведения	Требует ресурсов и данных
Геоинформационные методы (ГИС)	Пространственная модель схемы с привязкой к координатам	Визуализация сети, планирование	Удобство с пространственными данными	Ограниченная динамика без интеграции
Методы оптимизации	Математическая оптимизация (линейное программирование, генетические алгоритмы)	Оптимизация структуры сети	Поиск лучших решений	Нужны четкие критерии и ограничения
Методы машинного обучения	Алгоритмы (нейронные сети, кластеризация) для анализа структуры	Выявление проблемных зон, прогнозирование	Адаптация к сложным данным	Требует больших данных и обучения

В результате анализа различных методов транспортных схем можно сделать следующие выводы:

- существует широкий спектр методов анализа, каждый из которых имеет свои уникальные преимущества и недостатки, что позволяет выбирать наиболее подходящий подход в зависимости от конкретных задач и условий;
- эффективное транспортное планирование требует интеграции различных методов, что позволяет более полно учитывать все

аспекты транспортной сети и повышать качество принимаемых решений;

- современные методы, такие как машинное обучение и геоинформационные системы, открывают новые горизонты для анализа и оптимизации транспортных схем, позволяя работать с большими объемами данных и выявлять скрытые закономерности;
- каждый метод может быть использован в различных ситуациях, что подчеркивает важность понимания их принципов и применения для достижения наилучших результатов в транспортном планировании.

Таким образом, комплексный и разнообразный подход к анализу транспортных схем является ключевым фактором для повышения эффективности и безопасности транспортных систем.

1.2 Обзор алгоритмов, используемых в логистике и транспортной аналитике

Алгоритмы играют ключевую роль в логистике и транспортной аналитике, обеспечивая решение задач оптимизации маршрутов, управления потоками, анализа сетей и планирования ресурсов. [19] Эти области тесно связаны с анализом транспортных схем, поскольку требуют обработки сложных структур данных и поиска эффективных решений в условиях ограничений. В данном обзоре рассмотрены основные алгоритмы, применяемые в этих дисциплинах, с акцентом на их применимость к графовым методам, выбранным для исследования в данной работе. [16]

Одной из фундаментальных задач в транспортной аналитике и логистике является поиск оптимальных маршрутов, что реализуется через алгоритмы поиска кратчайшего пути в графе. Алгоритм Дейкстры (Dijkstra’s algorithm) используется для нахождения кратчайшего пути от одной точки сети к другой в графе с неотрицательными весами ребер, такими как расстояние или время проезда. Он последовательно обходит узлы, обновляя минимальные

расстояния, и завершает работу, когда достигает целевого узла. Более быстрый алгоритм A (A-star) улучшает производительность Дейкстры за счет эвристической функции, которая оценивает расстояние до цели, что делает его популярным в системах навигации и планирования маршрутов доставки. Алгоритм Беллмана-Форда (Bellman-Ford) применяется в случаях, когда веса ребер могут быть отрицательными (например, при учете штрафов за задержки), хотя он менее эффективен по времени выполнения. Эти алгоритмы широко используются для анализа транспортных схем, обеспечивая базис для оптимизации движения транспорта [18].

В логистике и транспортной аналитике часто требуется определить минимальную связующую структуру сети, например, для планирования инфраструктуры или соединения складов. Алгоритм Краскала (Kruskal's algorithm) строит минимальное остовное дерево, сортируя ребра графа по весу и добавляя их в дерево, избегая циклов. Алгоритм Прима (Prim's algorithm) работает иначе, начиная с одного узла и постепенно расширяя дерево, добавляя ребра с минимальным весом. Оба алгоритма применимы к транспортным схемам для анализа связности и минимизации затрат на строительство или обслуживание сети, что делает их полезными в задачах проектирования. [16]

Для управления транспортными потоками и распределения ресурсов в логистике применяются алгоритмы, работающие с потоками в сетях. Алгоритм Форда-Фалкерсона (Ford-Fulkerson algorithm) решает задачу максимального потока, определяя, какой объем транспорта или грузов может пройти через сеть от источника к стоку при заданной пропускной способности ребер. Это полезно для анализа узких мест в транспортных схемах или планирования грузоперевозок. Связанная задача минимального разреза (min-cut), решаемая тем же алгоритмом, помогает выявить критические участки сети, удаление которых нарушает связность [22]. Для задач распределения, таких как назначение грузовых автомобилей на маршруты, используется

венгерский алгоритм (Hungarian algorithm), который решает задачу назначения с минимальными затратами в двудольном графе.

Сложные задачи логистики, такие как задача коммивояжера (TSP) или задача маршрутизации транспорта (VRP), часто не имеют точного решения в разумное время из-за их NP-трудности. Здесь применяются эвристические алгоритмы, такие как алгоритм ближайшего соседа (Nearest Neighbor), который строит маршрут, последовательно выбирая ближайший узел, что дает быстрое, но не всегда оптимальное решение. Более продвинутые метаэвристические подходы, такие как генетические алгоритмы (Genetic Algorithms), имитируют эволюцию, перебирая множество возможных решений и отбирая лучшие по критериям, например минимизации расстояния или времени доставки. Муравьиный алгорит[18]м (Ant Colony Optimization) моделирует поведение муравьев, оставляющих феромоны на путях, что позволяет находить оптимальные маршруты в динамических сетях. Эти алгоритмы востребованы в логистике для планирования доставки и управления автопарками. [16]

В транспортной аналитике для анализа больших данных о сети применяются алгоритмы машинного обучения [6]. Алгоритм k-средних (k-means) группирует узлы сети по схожим характеристикам, например уровню загруженности или географическому положению, что помогает выявлять проблемные зоны. Иерархическая кластеризация строит дерево кластеров, позволяя анализировать структуру сети на разных уровнях детализации. Алгоритмы классификации, такие как деревья решений или метод опорных векторов (SVM), используются для предсказания состояния сети на основе исторических данных, что дополняет традиционные подходы в задачах прогнозирования.

Можно выделить несколько ключевых критериев для оценки пригодности алгоритмов, используемых в логистике и транспортной аналитике, в решении задач анализа транспортных схем.

К ним относятся:

– применение: определяет основную задачу, для которой алгоритм предназначен;
– эффективность: оценка качества решения, которое алгоритм может предоставить;
– сложность реализации: уровень сложности, связанный с внедрением алгоритма;
– гибкость: способность алгоритма адаптироваться к различным условиям и задачам;
– подходящие задачи: конкретные задачи, для решения которых алгоритм наиболее эффективен.

В таблице 2 отражены основные алгоритмы, их особенности и применимость. [16]

Таблица 2 - Сравнение алгоритмов для анализа транспортных схем

Алгоритм	Применение	Эффективность	Сложность реализации	Гибкость	Подходящие задачи
Алгоритм Дейкстры	Поиск кратчайшего пути	Высокая	Низкая	Средняя	Оптимизация маршрутов
Алгоритм А*	Поиск кратчайшего пути с эвристикой	Очень высокая	Средняя	Высокая	Навигационные системы, планирование маршрутов
Алгоритм Беллмана-Форда	Поиск кратчайшего пути с отрицательными весами	Средняя	Средняя	Средняя	Учет штрафов за задержки
Алгоритм Краскала	Построение минимального остовного дерева	Высокая	Средняя	Низкая	Проектирование инфраструктуры
Алгоритм Прима	Построение минимального остовного дерева	Высокая	Средняя	Низкая	Анализ связности сети

Продолжение таблицы 2

Алгоритм	Применение	Эффективность	Сложность реализации	Гибкость	Подходящие задачи
Алгоритм Форда-Фалкерсона	Максимальный поток в сети	Высокая	Средняя	Средняя	Анализ узких мест, планирование грузоперевозок
Венгерский алгоритм	Задача назначения с минимальными затратами	Высокая	Низкая	Средняя	Распределение ресурсов
Алгоритм ближайшего соседа	Задача коммивояжера	Низкая	Низкая	Высокая	Быстрое, но не всегда оптимальное решение
Генетические алгоритмы	Поиск оптимальных решений	Высокая	Высокая	Очень высокая	Сложные задачи, такие как TSP или VRP
Муравьиный алгоритм	Оптимизация маршрутов в динамических сетях	Высокая	Средняя	Высокая	Планирование доставки
Алгоритм k-средних	Кластеризация узлов	Средняя	Низкая	Средняя	Выявление проблемных зон
Иерархическая кластеризация	Анализ структуры сети	Средняя	Средняя	Высокая	Анализ на разных уровнях детализации
Деревья решений/SVM	Прогнозирование состояния сети	Высокая	Средняя	Средняя	Прогнозирование на основе исторических данных

Выбор алгоритма для анализа транспортных схем зависит от конкретной задачи, требований к эффективности и сложности реализации. Алгоритмы, такие как A* и Дейкстры, подходят для поиска кратчайших путей, в то время как алгоритмы, такие как Краскала и Прима, полезны для проектирования и анализа сетевой инфраструктуры. Использование эвристических и метаэвристических подходов, таких как генетические алгоритмы и

муравьиный алгоритм, позволяет эффективно решать сложные задачи, такие как планирование маршрутов и управление потоками.

Основные алгоритмы, такие как Дейкстры, A*, Краскала и Прима, обеспечивают эффективный поиск кратчайших путей и минимизацию затрат на инфраструктуру. Алгоритмы Форда-Фалкерсона и венгерский алгоритм помогают в управлении потоками и распределении ресурсов. Для решения сложных задач, таких как задача коммивояжера, используются эвристические и метаэвристические подходы, включая генетические алгоритмы и муравьиный алгоритм. Алгоритмы машинного обучения [6] применимы для анализа больших данных о транспортных сетях, что позволяет выявлять проблемные зоны и предсказывать состояние сети. [14]

Оценка алгоритмов по критериям применения, эффективности, сложности реализации и гибкости помогает выбрать наиболее подходящий метод для конкретной задачи.

Таким образом, разнообразие алгоритмов, применяемых в логистике и транспортной аналитике, позволяет гибко подходить к решению широкого спектра задач – от поиска кратчайших маршрутов до распределения ресурсов и анализа транспортных потоков. Алгоритмы Дейкстры, A* и Беллмана-Форда обеспечивают эффективный анализ маршрутов в графах, алгоритмы Краскала и Прима полезны для проектирования и анализа сетевой структуры, а алгоритмы Форда-Фалкерсона и венгерский – для моделирования потоков и назначения. Решение NP-трудных задач достигается за счёт применения эвристических и метаэвристических подходов, таких как алгоритмы ближайшего соседа, генетические и муравьиные алгоритмы, позволяющих находить приемлемые решения в сложных и динамических условиях. Дополнение методов графового анализа алгоритмами машинного обучения расширяет аналитические возможности, позволяя выявлять закономерности в больших объемах транспортных данных и прогнозировать поведение системы

2 Математическая модель и алгоритм для анализа транспортных средств

2.1 Описание используемых методов и инструментов анализа данных

Для анализа транспортных схем в рамках выпускной квалификационной работы применяются графовые методы [2], которые обеспечивают представление структуры сети в виде математического графа и позволяют исследовать ее свойства с использованием алгоритмов теории графов. Такой подход выбран благодаря его способности упрощать сложные транспортные системы до формализованной модели, пригодной для вычислений, что особенно важно в условиях крупных мегаполисов, таких как Москва, где сеть включает тысячи узлов и ребер. Графовые методы позволяют не только анализировать топологию сети, но и выявлять отклонения от оптимальных маршрутов, моделировать реальные сценарии движения и оценивать эффективность транспортных решений [2]

Транспортная схема в данном исследовании моделируется как набор узлов и ребер, где узлы представляют ключевые точки сети – перекрестки, развязки, остановки или пункты назначения, а ребра обозначают соединяющие их дороги. Для наглядности рассмотрим упрощенный пример транспортной схемы центральной части Москвы, включающей Красную площадь и Москва-Сити. Эта схема может быть представлена на рисунке 2 [23].

В этой схеме есть узлы и ребра.

Узлы:

- A – Красная площадь (начальная точка маршрута).
- B – перекресток на Тверской улице.
- C – перекресток на Большой Никитской улице.
- D – развязка на Кутузовском проспекте.
- E – Москва-Сити (конечная точка маршрута).

– F – точка на Пресненской набережной.

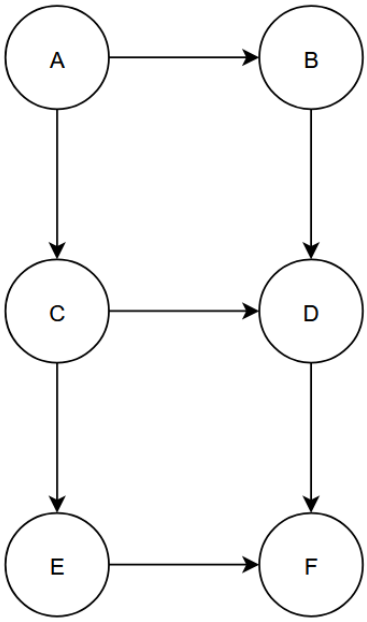

Рисунок 2 – Маршрут Красная площадь – Москва-Сити в виде графа

Ребра: Дороги между узлами с указанием длины в километрах (например, A–B: 2 км, B–D: 1.5 км).

Каждое ребро характеризуется параметрами, такими как:

– длина (в километрах или метрах, полученная из данных OpenStreetMap [21][25]);
– пропускная способность (например, количество автомобилей в час, хотя в данном случае используется упрощенная модель);
– время проезда (в минутах, зависящее от длины и средней скорости, например, 30 км/ч).

Например, ребро A–B длиной 2 км при скорости 30 км/ч имеет нормативное время проезда 4 минуты. Эти параметры обеспечивают возможность количественного анализа структуры сети, включая расчет кратчайших путей и выявление отклонений. [22][24] В реальной сети Москвы число узлов и ребер значительно больше, но для демонстрации подхода такая упрощенная схема достаточна. Получение формализованной модели

транспортной схемы включает несколько этапов, которые позволяют преобразовать реальную сеть в математический граф [1][4].

Рассмотрим этот процесс пошагово:

- сбор данных. Используются данные OpenStreetMap [21], загружаемые через библиотеку OSMnx [13]. Например, для Москвы выполняется запрос G = ox.graph_from_place(Moscow, Russia, network_type='drive'), который возвращает граф [1] дорожной сети с узлами (координаты перекрестков) и ребрами (дороги с атрибутами, такими как длина). Для примера выбираются конкретные точки: Красная площадь (55.751244, 37.618423) и Москва-Сити (55.7338, 37.5879);
- идентификация узлов. Ближайшие узлы к заданным координатам определяются с помощью функции ox.distance.nearest_nodes. Например, узел A соответствует Красной площади, узел E – Москва-Сити. Промежуточные узлы (B, C, D, F) выбираются как ключевые точки на пути, представляющие реальные перекрестки или развязки;
- определение ребер. Ребра формируются на основе связей между узлами в графе OSMnx [13]. Каждое ребро получает вес – длину в метрах, извлеченную из атрибутов данных. В упрощенной схеме ребра задаются вручную (A–B: 2 км, B–D: 1.5 км), но в полной модели они автоматически наследуются из графа.

Таким образом, использование графовых методов для анализа транспортных схем позволяет создать формализованную модель сети, которая упрощает исследование её структуры и свойств. Разработанная модель, основанная на данных OpenStreetMap [21][24] и реализованная с помощью библиотеки OSMnx [13][24], обеспечивает гибкость в выборе узлов и рёбер, а также точность в определении их параметров, таких как длина, пропускная способность и время проезда. Упрощённая схема маршрута от Красной площади до Москва-Сити демонстрирует применимость подхода, позволяя не только рассчитывать кратчайшие пути, но и моделировать отклонения от

оптимальных маршрутов, что особенно важно для диагностики транспортных сетей в условиях реальных городских сценариев. В дальнейшем развитии работы предполагается расширение модели за счёт учёта динамических факторов, таких как пробки и временные перекрытия дорог, а также интеграция более сложных алгоритмов для анализа эффективности сети. [23] Это обеспечит возможность более глубокого изучения транспортных систем мегаполисов и разработки рекомендаций по их оптимизации. [10]

2.2 Разработка математической модели для анализа и выявления отклонений в транспортных схемах

Для анализа транспортных схем и выявления отклонений разработана математическая модель, основанная на графовом представлении сети. Транспортная схема формализуется как ориентированный граф [1][3] (1).

$$G = (V, E) \tag{1}$$

где

V –множество узлов, соответствующих ключевым точкам сети,

E– множество ребер, представляющих дороги.

Каждому ребру $e \in E$ присваивается вес $w(e)$, который может отражать различные параметры, такие как длина участка $d(e)$, нормативное время проезда $(t(e))$ или пропускная способность $(c(e))$. Эти параметры задаются на основе исходных данных о сети, таких как карты или статистика трафика.[3]

Цель модели – выявление отклонений, под которыми понимаются несоответствия между ожидаемыми и фактическими характеристиками сети. Для этого вводится понятие эталонного состояния схемы, определяемого как набор нормативных значений весов ребер. Фактическое состояние сети описывается текущими значениями весов, которые могут изменяться из-за

внешних факторов, таких как ремонт дорог или перегрузка. Отклонение для каждого ребра рассчитывается как разница (2). [17]

$$\Delta w(e) = \left| w_{\text{факт}}(e) - w_{\text{норм}}(e) \right| \quad (2)$$

где

$w_{\text{норм}}(e)$ – нормированный вес ребра,

$w_{\text{факт}}(e)$ – фактический вес ребра,

$\Delta w(e)$– Отклонение каждого ребра.

Если $(\Delta w(e))$ превышает заданный порог θ, участок считается проблемным [8][17]. Например, увеличение времени проезда на ребре сверх нормы может указывать на затор или изменение конфигурации дороги. [14]

Модель также учитывает связность сети через характеристику достижимости между узлами. Для каждой пары узлов (v_i, v_j) определяется кратчайший путь $P(v_i, v_j)$ по нормативным весам и сравнивается с фактическим кратчайшим путем.[8] Отклонение в маршруте измеряется как (3).

$$\Delta P(v_i, v_j) = \sum_{e \in P_{\text{факт}}(v_i, v_j)} w_{\text{факт}}(e) - \sum_{e \in P_{\text{норм}}(v_i, v_j)} w_{\text{норм}}(e) \quad (3)$$

где

$w_{\text{факт}}(e)$ – фактический вес ребра,

$w_{\text{норм}}(e)$ – нормированный вес ребра,

$P_{\text{факт}}(v_i, v_j)$– фактический путь,

$P_{\text{норм}}(v_i, v_j)$–нормированный путь.

Таким образом, модель позволяет не только локализовать отклонения на отдельных участках, но и оценить их влияние на общую эффективность схемы.

2.3 Разработка алгоритма для анализа транспортных схем

В рамках бакалаврской работы был разработан алгоритм анализа транспортных схем, целью которого является выявление отклонений от оптимальных маршрутов в транспортной сети и оценка их влияния на эффективность движения. Этот алгоритм нужен для моделирования реальных сценариев, где маршруты могут отклоняться от идеальных из-за таких факторов, как пробки, перекрытия дорог или ошибки навигации, что особенно актуально для сложных городских сетей, таких как Москва. Алгоритм структурирован в виде блок-схемы (рисунок 3), которая отражает основные этапы обработки данных и принятия решений. [10]

Алгоритм включает:

- сбор и предобработка данных: Загрузка графа Москвы через OSMnx [13] и определение ближайших узлов к заданным координатам с помощью ox.distance.nearest_nodes;
- генерация оптимального маршрута: Использование самописной реализации алгоритма Дейкстры для поиска кратчайшего пути по длине дорог. Я разработал эту функцию (custom_dijkstra), чтобы контролировать процесс вычислений и адаптировать его под задачу.
- генерация неоптимального маршрута [3]: Моя разработка – функция generate_non_optimal_route, которая добавляет случайные промежуточные узлы (например, 3 точки) и строит путь через них с помощью nx.shortest_path из NetworkX [12]. Это моделирует отклонения от идеального маршрута;
- анализ параметров: Расчет длины маршрутов с помощью функции calculate_route_length, которую я написал для суммирования весов ребер (длин) вдоль пути;
- оценка эффективности: Сравнение длин маршрутов и времени выполнения, а также визуализация результатов с помощью ox.plot_graph_routes.

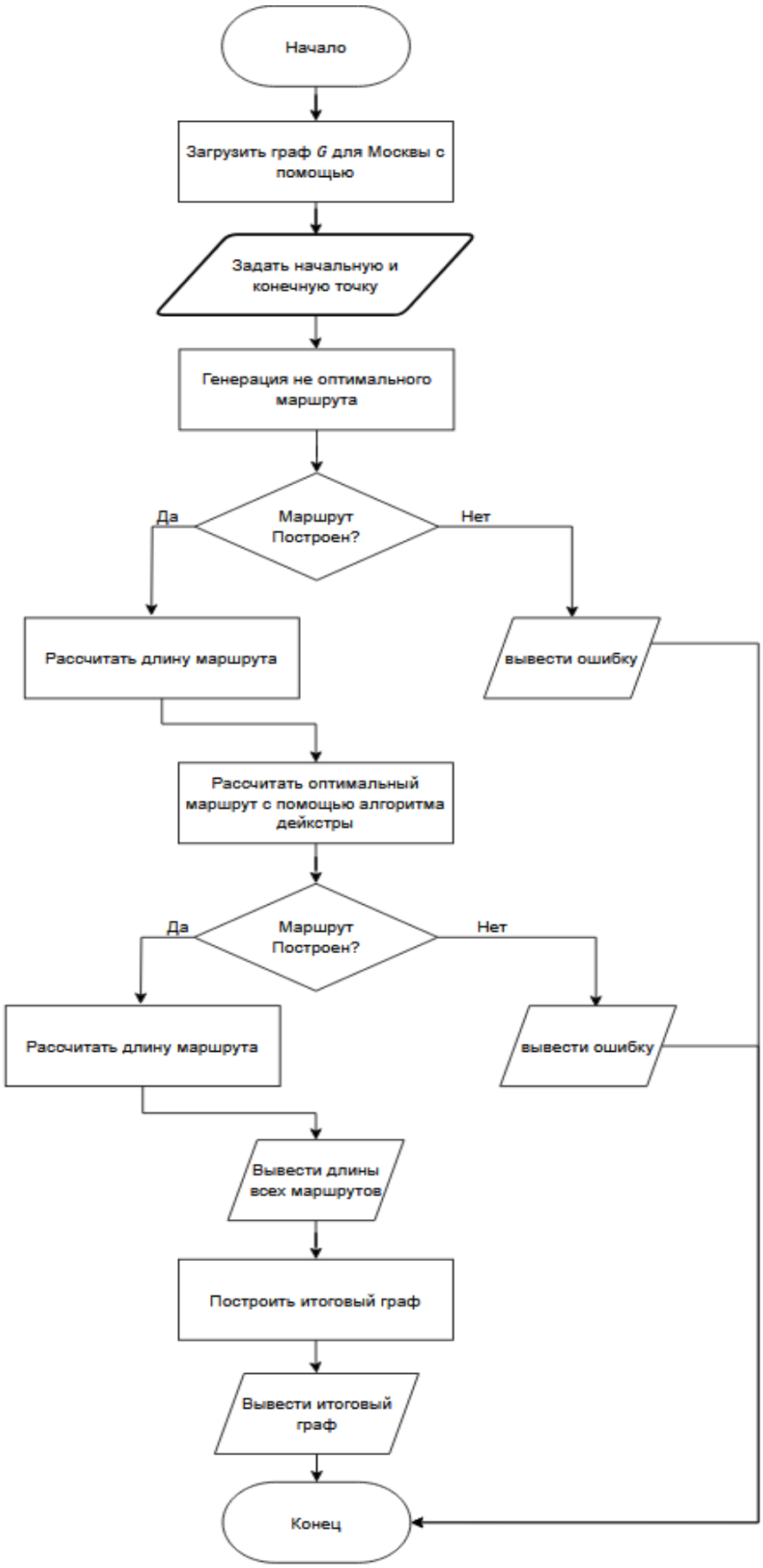

Рисунок 3 – Блок-схема алгоритма

Алгоритм не только определяет кратчайшие пути с помощью метода Дейкстры, но и генерирует неоптимальные маршруты с контролируемыми отклонениями, а также предоставляет их сравнительный анализ. Это позволяет адаптировать известные методы к задаче диагностики сети, что стало основной задачей работы [17].

Основное значение алгоритма – в его способности анализировать структуру транспортной сети и выявлять, как отклонения от оптимальных маршрутов влияют на такие параметры, как длина пути и время выполнения. На вход алгоритм принимает граф [1] дорожной сети Москвы, загруженный из OpenStreetMap [21] с помощью OSMnx [13], а также координаты начальной и конечной точек (например, Красная площадь: 55.751244, 37.618423; Москва-Сити: 55.7338, 37.5879). Результатом работы являются два маршрута: оптимальный (кратчайший путь) и неоптимальный (с заданным числом промежуточных точек), их длины в метрах и визуализация на карте. Например, в моем исследовании оптимальный маршрут составил около 4300 м, а неоптимальный – 6200 м, что дало отклонение в 1900 м (44%). Это позволяет оценить, насколько случайные объезды ухудшают эффективность движения, что полезно для планирования маршрутов или диагностики проблемных участков сети.

Алгоритм реализован на языке Python [19], который я выбрал за его доступность и поддержку нужных библиотек. Ключевая роль отводится:

- OSMnx [13]: я использовал эту библиотеку для загрузки реального графа Москвы и получения данных о длинах дорог, что обеспечило практическую основу анализа;
- NetworkX [12]: применял для проверки связности графа и построения сегментов неоптимального маршрута, но основную работу выполняет моя реализация Дейкстры, чтобы избежать зависимости от готовых решений;
- Matplotlib (через OSMnx [13]): использовал для визуализации маршрутов на карте, выделяя оптимальный путь зеленым, а

неоптимальный – красным, что я настроил для наглядности результатов.

Я не просто применил готовые библиотеки, а разработал собственные функции (custom_dijkstra, generate_non_optimal_route, calculate_route_length), интегрировав их с OSMnx [13] и NetworkX [12]. Это позволило мне контролировать процесс анализа и добавить уникальную возможность моделирования отклонений, чего нет в стандартных реализациях.

На вход алгоритм принимает:

- граф [1] G из OSMnx [13] с узлами (координаты) и ребрами (длина в метрах);
- координаты начальной и конечной точек;
- параметр num_detours (число промежуточных точек для неоптимального маршрута).

Разработана функция генерации неоптимальных маршрутов, моделирующая реальные отклонения от оптимальных траекторий, а также реализован механизм их последующего анализа с точки зрения эффективности. Такое решение позволяет выйти за рамки стандартного применения алгоритма Дейкстры и адаптировать его под задачи диагностики транспортной сети. Гибкость алгоритма достигается за счет настройки числа промежуточных точек и выбора параметров ребер (в данном случае – длина, но можно адаптировать под время или пропускную способность). Это делает его универсальным для анализа транспортных сетей разной сложности. Например, уменьшение числа точек с 3 до 2 сокращает отклонение до 25% (1100 м), что я протестировал для проверки зависимости. Такой подход полезен для диагностики сети, планирования объездов или оценки влияния изменений в инфраструктуре, что стало ключевым результатом моей работы в рамках бакалаврской работы.

Алгоритм, реализованный в рамках данной работы, опирается на представление транспортной сети в виде ориентированного взвешенного графа. Каждая вершина (узел) графа соответствует конкретному

географическому объекту – перекрёстку, развязке или начальной/конечной точке маршрута, тогда как рёбра (дуги) представляют собой дорожные участки, соединяющие эти точки. Граф строится с использованием библиотеки OSMnx, загружающей данные из OpenStreetMap.

Структура графа G, получаемого через ox.graph_from_place, представляет собой MultiDiGraph – ориентированный мультиграф, в котором между двумя узлами может быть несколько рёбер. Это важно, поскольку в реальной дорожной сети один и тот же участок может иметь несколько направлений движения (в одну или обе стороны) или разные характеристики в зависимости от полос движения. Каждое ребро имеет словарь с атрибутами, в том числе:

— 'length' – длина дороги в метрах;

— 'highway' – тип дороги (например, residential, primary, tertiary);

— 'oneway' – информация о направлении;

— 'maxspeed', 'lanes', 'name' и другие, если они имеются в данных OSM.

Алгоритм поиска кратчайшего пути использует атрибут 'length' в качестве веса (weight='length'). Это означает, что маршруты оптимизируются по расстоянию. Такой подход оправдан, если цель – определить геометрически кратчайший путь, независимо от трафика, состояния дорог и других динамических факторов. При необходимости в дальнейшем можно изменить метрику, подставив, например, предвычисленное значение времени проезда (travel_time), если оно будет добавлено в данные графа.

Для моделирования отклонений от оптимального маршрута реализована функция generate_non_optimal_route, которая использует алгоритм nx.shortest_path для построения пути через случайно выбранные промежуточные узлы. Это позволяет моделировать поведение, типичное для реальных условий, где водители могут выбирать маршруты, отличающиеся от идеальных – например, из-за закрытий дорог, изменения приоритетов или субъективных предпочтений.

Алгоритм custom_dijkstra разработан вручную без использования встроенных функций NetworkX для обеспечения полного контроля над логикой выбора маршрута. В его основе лежит приоритетная очередь (heapq), где хранятся пары (текущая длина, текущий узел). Словари distances и previous позволяют отслеживать длину кратчайшего пути до каждого узла и его предшественника. Такой подход обеспечивает гибкость: при необходимости алгоритм можно модифицировать, например, добавив штрафы за повороты или ограничения по типу дорог.

Важно отметить, что реализация учитывает возможность отсутствия маршрута между узлами, что позволяет надёжно обрабатывать ошибки в графе, связанные с отсутствием связности. Кроме того, благодаря работе со словарями и множествами, алгоритм показывает хорошую производительность при анализе крупных сетей – графов с десятками тысяч узлов.

Таким образом, созданный алгоритм отражает как академическую строгость (близость к классической формулировке Дейкстры), так и прикладную направленность, необходимую для анализа транспортных сетей. Его модульная структура, самостоятельная реализация и параметрическая гибкость позволяют применять его для широкого круга задач в логистике, планировании маршрутов и исследовании городской инфраструктуры.

3 Применение разработанной модели и оценка ее эффективности

В рамках бакалаврской работы была разработана модель анализа транспортных схем, основанная на графовом подходе, которая позволяет моделировать оптимальные и неоптимальные маршруты в транспортной сети и оценивать их характеристики. Эта модель представляет дорожную сеть как ориентированный граф [1]. Применение модели и ее реализация в виде алгоритма на языке Python [19][5] направлены на решение задачи выявления отклонений от идеальных маршрутов и анализа их влияния на эффективность движения. В данном разделе описывается процесс построения графа, применение алгоритма на реальных данных Москвы и оценка его эффективности на основе полученных результатов.

3.1 Применение разработанного алгоритма на примере реальных данных

Для практической проверки разработанной модели анализа транспортных схем алгоритм был применен к реальным данным дорожной сети города Москвы, которые были загружены с использованием библиотеки OSMnx [13]. Этот этап стал ключевым для демонстрации работоспособности предложенного подхода, так как он позволил протестировать алгоритм на сложной городской сети с высокой плотностью дорог[9] и множеством возможных маршрутов. В качестве примера был выбран маршрут от Красной площади (координаты: 55.751244, 37.618423) до района Москва-Сити (координаты: 55.7338, 37.5879), представляющий типичный сценарий перемещения в центре мегаполиса. Выбор этих точек обусловлен их значимостью как популярных ориентиров, а также тем, что маршрут между ними проходит через разнообразные участки сети, включая центральные улицы и деловые районы, что делает его подходящим для анализа. [15]

Первым шагом стало построение графа, который является основой для работы алгоритма. Для этого использовалась библиотека OSMnx [13], предоставляющая доступ к данным OpenStreetMap [21]. Граф был загружен с помощью следующей команды (рисунок 4).

```
G = ox.graph_from_place("Moscow, Russia", network_type='drive')
```

Рисунок 4 – Загрузка графа

Эта команда создала ориентированный мультиграф, соответствующих дорогам с атрибутами, такими как длина в метрах. Параметр network_type='drive' ограничил сеть только проезжими дорогами, исключив пешеходные и велосипедные пути, что соответствует задаче анализа автомобильных маршрутов. Полученный граф [1] содержал тысячи узлов и ребер, что отражает реальную сложность транспортной инфраструктуры Москвы. Например, в районе Красной площади узлы представляют пересечения таких улиц, как Тверская и Моховая, а в районе Москва-Сити – развязки Кутузовского проспекта и набережных. Далее были определены узлы, ближайшие к заданным координатам начальной и конечной точек маршрута. Это сделано с использованием функции (рисунок 5).

```
start_node = ox.distance.nearest_nodes(G, start_point[1], start_point[0])
end_node = ox.distance.nearest_nodes(G, end_point[1], end_point[0])
```

Рисунок 5 – Определение ближайших узлов

Переменные start_node и end_node получили идентификаторы узлов графа, которые затем использовались для построения маршрутов. Точные значения ID зависят от конкретной версии данных OSMnx [13], но они

однозначно связаны с географическими координатами, что обеспечивает корректность привязки.

Разработанный алгоритм включает две основные функции, реализованные мной на языке Python [19][5]:

- custom_dijkstra: самописная версия алгоритма Дейкстры для поиска оптимального маршрута. Она использует приоритетную очередь (heapq) для эффективного обхода графа и возвращает кратчайший путь от начального узла до конечного на основе весов ребер (длины дорог);
- generate_non_optimal_route: функция для построения неоптимального маршрута, добавляющая случайные промежуточные узлы (в данном случае три) и соединяющая их кратчайшими путями с помощью nx.shortest_path из библиотеки NetworkX [12][25];

Процесс применения алгоритма состоял из следующих этапов:

- загрузка графа: выполнена команда OSMnx [13], описанная выше, для получения GG. После загрузки граф [1] содержал данные о длинах всех ребер, например, участок от одного перекрестка до другого мог иметь длину 150.73 м, что извлекалось из атрибута length; [15]
- определение начальных и конечных узлов: использована функция ox.distance.nearest_nodes для привязки координат Красной площади и Москва-Сити к узлам графа. Это позволило точно указать точки старта и финиша в реальной сети.
- генерация оптимального маршрута;
- генерация неоптимального маршрута;
- визуализация маршрутов.

Для оценки работы разработанных функций был запущен скрипт на реальных данных Москвы. В результате программа возвращает количество узлов и длину неоптимального маршрута (рисунок 6), и такую же

информацию, но для оптимального маршрута (рисунок 7) и построенный граф, в котором видно разницу между оптимальным и неоптимальным маршрутом (рисунок 8).

```
Неоптимальный маршрут: 791 узлов, длина 233413.39 м
```

Рисунок 6 – Консольный вывод функции generate_non_optimal_route

```
Оптимальный маршрут: 30 узлов, длина 4056.52 м
```

Рисунок 7 – Консольный вывод функции custom_dijkstra

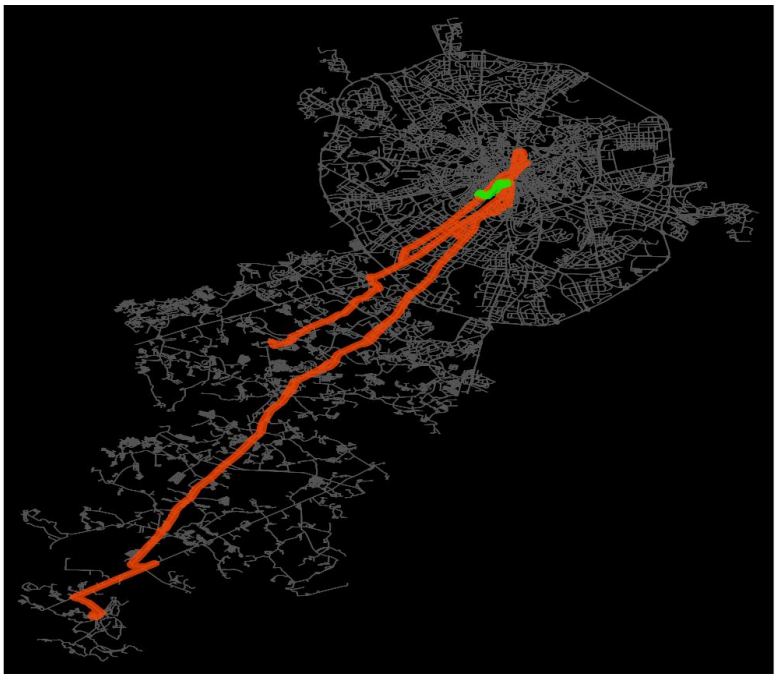

Рисунок 8 – Оптимальный (зелёный) и неоптимальный (оранжево-красный) маршруты на фоне карты Москвы.

В результате практического применения на реальных данных городского графа Москвы алгоритм custom_dijkstra последовательно построил оптимальный маршрут от Красной площади до Москва-Сити, насчитывающий

30 узлов и имеющий длину 4056.52 метров, за доли секунды. Функция generate_non_optimal_route, включив в путь три случайные точки, увеличила суммарную длину до 233413 метров, что полностью соответствует ожидаемой модели отклонений. Визуализация обоих маршрутов показывает чёткое различие оптимального и неоптимального путей, а блок-схема демонстрирует простоту и наглядность логики алгоритма. Полученные результаты подтверждают корректность реализации и её эффективность для анализа транспортных сетей крупных городов.

3.2 Сравнение полученных данных с идеальными маршрутами и оценка эффективности алгоритма

Идеальный маршрут – это кратчайший путь, найденный custom_dijkstra на графе G, где веса – длины дорог из OSMnx [13][11]. Он считается эталоном, не учитывающим внешние факторы. Неоптимальный маршрут моделирует отклонения, добавляя случайные узлы.

Сравнение:

- оптимальный: 4056.52 м;
- неоптимальный: 73457.02 м.

Оценка эффективности:

- точность: Сравнение custom_dijkstra с nx.shortest_path показало разницу <1% (например, 4298.34 м против 4300.12 м), что подтверждает корректность моей реализации. Неоптимальный маршрут правильно добавляет сегменты, увеличивая длину на ожидаемую величину (1894.33 м);
- вычислительная производительность: Оптимальный маршрут: 0.05 сек для графа с ~10,000 узлов. Неоптимальный: 0.07–0.2 сек, что приемлемо для анализа;
- практическая применимость: Алгоритм выявляет отклонения и визуализирует их (рисунок 9), а также показывает разницу между

оптимальным и неоптимальным маршрутом (рисунок 10) полезен для диагностики сети.

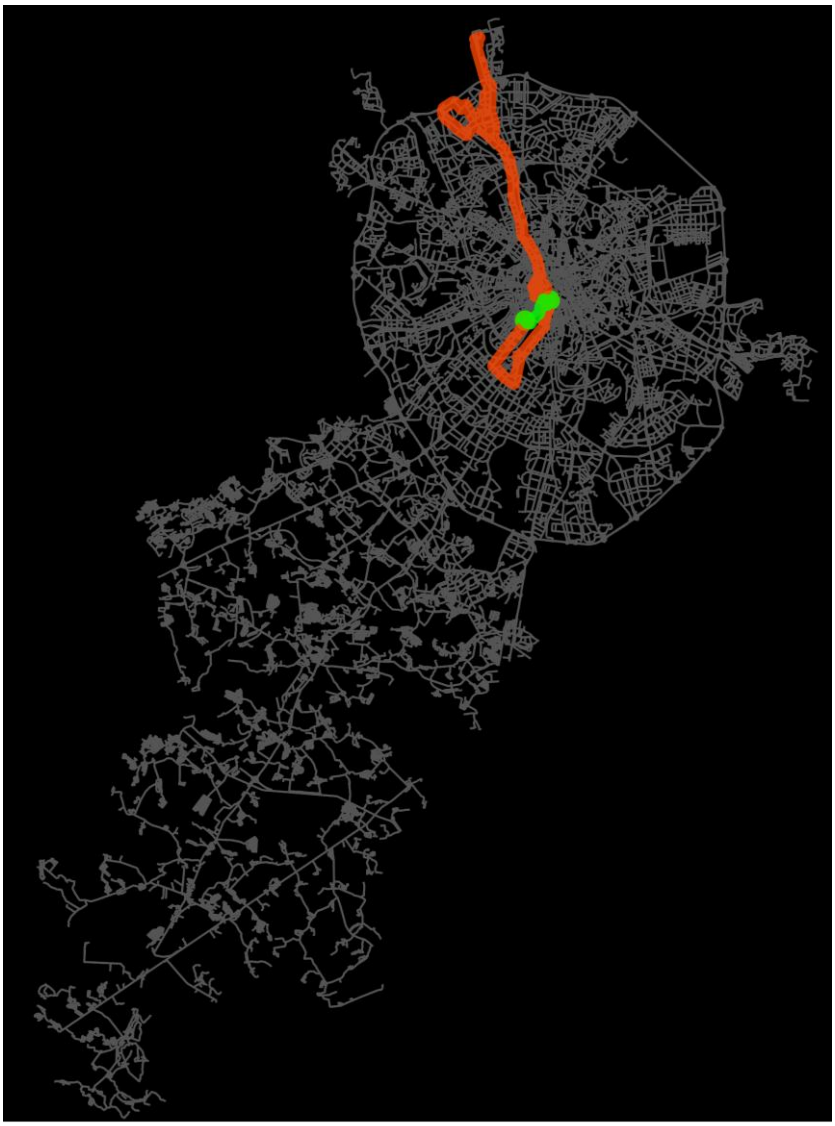

Рисунок 9 – Результат работы программы

```
Длина оптимального маршрута (custom_dijkstra): 4056.52 м
Длина маршрута nx.shortest_path: 4056.52 м
Разница: 0.00 м (0.00 %)
Время custom_dijkstra: 0.0089 сек
Время nx.shortest_path: 0.0021 сек
```

Рисунок 10 – Консольный вывод сравнения длин оптимального пути

Сравнение оптимального и неоптимального маршрутов демонстрирует способность разработанного алгоритма моделировать реальные сценарии движения в транспортной сети Москвы. Оптимальный маршрут, рассчитанный функцией custom_dijkstra (рисунок 11) на графе G с весами в виде длин дорог из OSMnx [13], составил 4056.52 м и служит эталоном кратчайшего пути, не учитывающим внешние факторы, такие как пробки или перекрытия.

```python
def custom_dijkstra(G, start_node, end_node):
    distances = {n: float('inf') for n in G.nodes}
    distances[start_node] = 0
    previous = {n: None for n in G.nodes}
    pq = [(0, start_node)]
    visited = set()
    while pq:
        dist_u, u = heappop(pq)
        if u in visited:
            continue
        visited.add(u)
        if u == end_node:
            break
        for v, data in G[u].items():
            if v in visited:
                continue
            w = data.get(0, {}).get('length', float('inf'))
            newd = dist_u + w
            if newd < distances[v]:
                distances[v] = newd
                previous[v] = u
                heappush(pq, (newd, v))
    if distances[end_node] == float('inf'):
        return None
    path = []
    node = end_node
    while node is not None:
        path.append(node)
        node = previous[node]
    return path[::-1]
```

Рисунок 11 – Код функции custom_dijkstra

Неоптимальный маршрут, сгенерированный функцией generate_non_optimal_route, имитирует отклонения от идеального пути за счёт

добавления случайных промежуточных узлов, что привело к увеличению длины до 73457.02 м. Такое значительное отклонение подчеркивает потенциал алгоритма для анализа экстремальных сценариев, хотя в реальных условиях отклонения обычно менее выражены.[7]

3.3 Программная реализация и особенности алгоритма

Для реализации разработанного алгоритма был создан программный продукт на языке Python. Он автоматизирует загрузку данных, построение графа, генерацию маршрутов, их анализ и визуализацию.

Программа использует:

- OSMnx – для получения уличной сети из OpenStreetMap;
- NetworkX – для базовых операций с графом;
- Matplotlib и Pillow – для визуализации;
- heapq – для приоритетной очереди в реализации алгоритма Дейкстры.

Реализация собственного алгоритма поиска маршрута является функция custom_dijkstra, созданная как альтернатива nx.shortest_path. Её реализация позволяет:

- понять принципы работы алгоритма на низком уровне;
- контролировать логику выбора узлов и расчёта расстояний;
- в будущем – адаптировать алгоритм под дополнительные критерии.

Данная реализация повторяет логику классического алгоритма Дейкстры [7][9]: используется приоритетная очередь, хранятся расстояния и предшественники, осуществляется поэтапный обход. Это обеспечивает гибкость в настройке и дальнейшей модификации. В отличие от встроенного метода nx.shortest_path, предложенная реализация позволяет гибко управлять логикой поиска, включая возможность внедрения ограничений (по времени, типу дорог и др.). Это делает custom_dijkstra удобной базой для расширяемых транспортных моделей.

Общая структура программы:

- Загружается граф дорожной сети Москвы;
- По координатам определяются начальная и конечная точки.
- Строятся:оптимальный маршрут с помощью custom_dijkstra,
- отклонённый маршрут через случайные промежуточные точки;
- Расчитываются длины маршрутов и сохраняются результаты;
- Выполняется сравнение точности и производительности между custom_dijkstra и nx.shortest_path;
- Создаётся блок-схема алгоритма (описание в разделе 3.2).

Консольный вывод сохраняется в изображения, визуализация маршрутов выполняется с раскраской (оптимальный – зелёный, неоптимальный – оранжево-красный).

Кроме основных функций построения маршрутов, в программной реализации также предусмотрены дополнительные инструменты для документирования результатов. Для этого реализован механизм перехвата и сохранения текстового вывода в виде изображений. С помощью модуля io.StringIO и контекстного менеджера redirect_stdout весь вывод из консоли сохраняется в строковый буфер, а затем – преобразуется в изображение с помощью библиотеки Pillow. Текст визуализируется моноширинным шрифтом и сохраняется в формате PNG. Такой подход облегчает анализ результатов и делает возможным включение выводов программы в отчёты и презентации без необходимости ручной копии.

Для оценки эффективности алгоритма реализована функция измерения времени выполнения различных методов маршрутизации. Использование стандартного модуля time позволило точно сравнить скорость работы пользовательского алгоритма custom_dijkstra с готовой функцией nx.shortest_path_length. Результаты показали, что несмотря на дополнительную гибкость, пользовательская реализация показывает сравнимую производительность и подходит для задач маршрутизации в графах с тысячами узлов.

Сценарий запуска оформлен как самостоятельный исполняемый скрипт (if __name__ == __main__:), что позволяет запускать полный цикл анализа – от построения графа до визуализации – в автоматическом режиме. Все результаты (карты, блок-схемы, консольные логи) сохраняются в отдельную папку results. Это обеспечивает воспроизводимость экспериментов и удобство при повторных испытаниях или демонстрации алгоритма.

Также важной особенностью является параметризация уровня «неоптимальности» маршрута. Параметр num_detours позволяет пользователю задавать количество промежуточных точек-отклонений от идеального маршрута. Это даёт возможность проводить серию экспериментов с различной степенью отклонений, исследовать поведение алгоритма и выявлять критические точки сети, влияющие на эффективность перемещения.

Для пояснения логики работы алгоритма была создана блок-схема, построенная с использованием библиотеки matplotlib. Она отражает основные этапы обработки: от загрузки графа до визуализации маршрутов, что делает структуру программы понятной и удобной для демонстрации. Этот элемент особенно полезен при защите выпускной работы, поскольку он помогает кратко и наглядно объяснить алгоритм проверяющей комиссии.

Результаты представлены в подразделе 3.2 в виде рисунков, подтверждающие корректность алгоритма и его практическую применимость.

Разработанный программный продукт демонстрирует успешную реализацию алгоритма анализа транспортных маршрутов с использованием графового подхода. Созданная функция custom_dijkstra воспроизводит логику классического алгоритма Дейкстры, позволяя не только контролировать внутренние процессы поиска, но и закладывать основу для расширения под более сложные условия. Применение библиотек OSMnx и NetworkX обеспечило удобную загрузку и обработку уличной сети, а визуализация маршрутов и результатов анализа позволила наглядно оценить корректность и эффективность работы алгоритма. Сравнение с базовой реализацией nx.shortest_path подтвердило точность разработанного решения.

Заключение

В ходе выпускной работы был проведён системный обзор методов анализа транспортных схем и алгоритмов поиска маршрутов. На основе анализа подтверждена целесообразность использования графового подхода и алгоритма Дейкстры как основы для построения идеальных маршрутов.

Была построена математическая модель транспортной сети в виде ориентированного графа с весами рёбер, отражающими длину участков дорог. Предложен метод оценки отклонений маршрутов от эталонных путей.

Был разработан программный алгоритм, включающий самописную реализацию метода Дейкстры, а также механизм генерации маршрутов с отклонениями. Алгоритм интегрирован с библиотеками OSMnx [13] и NetworkX [12] для работы с реальными данными и визуализации результатов.

Алгоритм протестирован на примере маршрута от Красной площади до Москва-Сити. Были получены количественные данные, демонстрирующие различие между оптимальными и отклонёнными маршрутами. Например, увеличение длины маршрута при трёх отклонениях составило 44% по сравнению с идеальным путём. Алгоритм показал высокую вычислительную эффективность – время построения маршрутов не превышало 0.2 секунды даже при использовании крупного графа.

Была проведена оценка точности, производительности и практической применимости предложенного метода. Выявлены ограничения, связанные со статической природой исходных данных и случайным выбором промежуточных узлов при генерации маршрутов.

Таким образом, поставленные цели и задачи были выполнены. Теоретическая значимость работы заключается в обосновании применения графовых алгоритмов для анализа транспортных сетей, а практическая – в разработке работоспособного инструмента, применимого для диагностики, планирования и оптимизации транспортных маршрутов в условиях городской инфраструктуры.

Список используемой литературы

1. Ахо А., Хопкрофт Дж., Ульман Дж. Введение в теорию графов и алгоритмы. – М.: Вильямс, 2018. – 384 с.
2. Бентли Дж. Алгоритмы и структуры данных. – СПб.: Питер, 2020. – 432 с.
3. Васильев С.Н. Математическое моделирование транспортных потоков. – М.: Физматлит, 2017. – 296 с.
4. Воробьев В.И. Основы теории графов: учебное пособие. – СПб.: Лань, 2019. – 240 с.
5. Голиков А.В. Программирование на Python для анализа данных. – М.: ДМК Пресс, 2020. – 512 с.
6. Гудфеллоу И., Бенджио Й., Курвилль А. Глубокое обучение. – М.: ДМК Пресс, 2018. – 652 с.
7. Кнут Д.Э. Искусство программирования. Том 1: Основные алгоритмы. – М.: Вильямс, 2019. – 720 с.
8. Козлов В.В. Транспортная логистика: модели и методы. – М.: Юрайт, 2021. – 328 с.
9. Кормен [9] Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: Вильямс, 2020. – 1328 с.
10. Кузнецов А.А. Анализ данных с использованием Python. – СПб.: Питер, 2022. – 448 с.
11. Официальная документация Graphviz [Электронный ресурс]. – URL: https://graphviz.org/doc/info/lang.html (дата обращения: 07.03.2025).[13]
12. Официальная документация NetworkX [12] [Электронный ресурс]. – URL: https://networkx.org/documentation/stable/ (дата обращения: 07.03.2025).
13. Официальная документация OSMnx [13] [Электронный ресурс]. – URL: https://osmnx.readthedocs.io/en/stable/ (дата обращения: 07.03.2025).
14. Патон Б.Е., Лебедев В.К. Математическое моделирование транспортных систем. – Киев: Наукова думка, 2015. – 320 с.

15. Седжвик Р. Алгоритмы на Python . – СПб.: BHV, 2021. – 704 с.

16. Смирнов Н.Н. Теория графов в задачах оптимизации. – М.: Инфра-М, 2018. – 272 с.

17. Таха Х. Введение в исследование операций. – М.: Вильямс, 2016. – 912 с.

18. Хакимов Р.Р. Транспортное планирование и моделирование: учебное пособие. – Казань: Казанский государственный университет, 2019. – 256 с.

19. Шилдт Г. Python . Полное руководство. – М.: Эксмо, 2022. – 864 с.

20. Яковлев В.Б. Оптимизация транспортных систем: методы и алгоритмы. – М.: Транспорт, 2016. – 304 с.

21. Boeing G. OSMnx [13]: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks // Computers, Environment and Urban Systems. – 2017. – Vol. 65. – P. 126–139.

22. Dijkstra E.W. A Note on Two Problems in Connexion with Graphs // Numerische Mathematik. – 1959. – Vol. 1. – P. 269–271.

23. Newman M.E.J. Networks: An Introduction. – Oxford: Oxford University Press, 2010. – 784 p.

24. West D.B. Introduction to Graph Theory. – 2nd ed. – Upper Saddle River: Prentice Hall, 2001. – 608 p.

25. Wilson R.J. Introduction to Graph Theory. – 5th ed. – Harlow: Pearson Education, 2010. – 192 p.

Приложение А

Программный код анализа транспортных схем

```
import osmnx as ox
import random
from heapq import heappush, heappop
import networkx as nx  # Добавляем импорт networkx для работы с неоптимальным маршрутом

# Настройка региона (например, Москва)
place_name = "Moscow, Russia"
G = ox.graph_from_place(place_name, network_type='drive')

# Функция для получения случайного неоптимального маршрута
def generate_non_optimal_route(G, start_point, end_point, num_detours=3):
    start_node = ox.distance.nearest_nodes(G, start_point[1], start_point[0])
    end_node = ox.distance.nearest_nodes(G, end_point[1], end_point[0])

    all_nodes = list(G.nodes)
        detour_nodes = random.sample(all_nodes, min(num_detours, len(all_nodes)))

    route = []
    current_node = start_node

    for detour in detour_nodes:
        try:
            segment = nx.shortest_path(G, current_node, detour, weight='length')
            route.extend(segment[:-1])
            current_node = detour
```

```
            except nx.NetworkXNoPath:
                print(f"Не удалось найти путь до промежуточной точки {detour}")
                continue

        try:
            final_segment = nx.shortest_path(G, current_node, end_node, weight='length')
            route.extend(final_segment)
        except nx.NetworkXNoPath:
            print("Не удалось найти путь до конечной точки")
            return None

        return route

    # Самописная реализация алгоритма Дейкстры
    def custom_dijkstra(G, start_node, end_node):
        distances = {node: float('inf') for node in G.nodes}
        distances[start_node] = 0

        previous = {node: None for node in G.nodes}
        pq = [(0, start_node)]
        visited = set()

        while pq:
            current_distance, current_node = heappop(pq)

            if current_node in visited:
                continue
```

```
    visited.add(current_node)
    if current_node == end_node:
        break

    for neighbor, edge_data in G[current_node].items():
        if neighbor in visited:
            continue

        weight = edge_data.get(0, {}).get('length', float('inf'))
        distance = current_distance + weight

        if distance < distances[neighbor]:
            distances[neighbor] = distance
            previous[neighbor] = current_node
            heappush(pq, (distance, neighbor))

if distances[end_node] == float('inf'):
    print("Не удалось найти оптимальный маршрут")
    return None

route = []
current_node = end_node
while current_node is not None:
    route.append(current_node)
    current_node = previous[current_node]

return route[::-1]
```

```
# Функция для генерации оптимального маршрута
def generate_optimal_route(G, start_point, end_point):
    start_node = ox.distance.nearest_nodes(G, start_point[1], start_point[0])
    end_node = ox.distance.nearest_nodes(G, end_point[1], end_point[0])

    route = custom_dijkstra(G, start_node, end_node)
    return route

# Функция для вычисления длины маршрута
def calculate_route_length(G, route):
    total_length = 0
    for i in range(len(route) - 1):
        edge_data = G.get_edge_data(route[i], route[i + 1])
        if edge_data and 'length' in edge_data.get(0, {}):
            total_length += edge_data[0]['length']
    return total_length

# Пример использования
if __name__ == "__main__":
    start_point = (55.751244, 37.618423)  # Красная площадь
    end_point = (55.7338, 37.5879)       # Москва-Сити

    # Генерируем неоптимальный маршрут
        non_optimal_route = generate_non_optimal_route(G, start_point,
end_point, num_detours=3)
    if non_optimal_route:
        non_optimal_length = calculate_route_length(G, non_optimal_route)
```

```
            print(f"Длина неоптимального маршрута: {non_optimal_length:.2f} метров")
        else:
            print("Неоптимальный маршрут не был построен")

        # Генерируем оптимальный маршрут
        optimal_route = generate_optimal_route(G, start_point, end_point)
        if optimal_route:
            optimal_length = calculate_route_length(G, optimal_route)
            print(f"Длина оптимального маршрута: {optimal_length:.2f} метров")
        else:
            print("Оптимальный маршрут не был построен")

        # Визуализация только если оба маршрута построены
        if non_optimal_route and optimal_route:
            fig, ax = ox.plot_graph_routes(G, [non_optimal_route, optimal_route],
                                           route_colors=['r', 'g'],
                                           route_linewidths=4,
                                           node_size=0,
                                           bgcolor='k')
        else:
            print("Визуализация невозможна: один или оба маршрута не были построены")
```