МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»**

Кафедра математического и компьютерного моделирования

Различные методы решения задачи коммивояжёра

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 441 группы

направление 09.03.03 — Прикладная информатика

механико-математического факультета

Столярова Артёма Сергеевича

Научный руководитель
доцент, к.ф.-м.н., доцент _____ С.В. Иванов

Зав. кафедрой
зав. каф., д.ф.-м.н., доцент _____ Ю.А. Блинков

Саратов 2025

Введение. Задача коммивояжёра является одной из самых известных и широко исследуемых задач в области теории графов и оптимизации. Суть задачи заключается в нахождении кратчайшего маршрута, который проходит через все заданные города, при этом каждый город должен быть посещен ровно один раз и путь должен возвращаться в исходный город. Эта задача имеет большое значение в таких областях, как логистика, транспортировка товаров и другие. В связи с вычислительной сложностью задачи, существует множество методов её решения, включая как точные, так и приближённые алгоритмы.

Цель работы — анализ и сравнение различных алгоритмов для решения задачи коммивояжёра, а также разработка веб-приложения, позволяющего тестировать эти алгоритмы на реальных данных.

Задачи, поставленные в рамках данной работы, включают: изучение теоретических аспектов задачи коммивояжёра, рассмотрение основных алгоритмов решения, разработку приложения для визуализации этих алгоритмов и проведение анализа их эффективности.

В первом разделе рассматривается задача коммивояжёра, которая была сформулирована в 19 веке и является классической задачей в теории оптимизации. Формулировка задачи заключается в нахождении кратчайшего маршрута, который проходит через все заданные города, возвращаясь в исходную точку. Каждый город должен быть посещен ровно один раз. С увеличением числа городов количество возможных путей растет факториально, что делает её решение вычислительно сложным.

Задача была предложена в 1850 году и привлекла внимание ученых после того, как в 1970-х годах были разработаны первые алгоритмы для её приближенного решения. Несмотря на развитие новых методов и алгоритмов, задача остаётся актуальной и сложной для больших наборов данных.

Во втором разделе рассматриваются методы решения задачи коммивояжёра.

1. Метод полного перебора

   Метод полного перебора заключается в переборе всех возможных путей между городами, что гарантирует нахождение оптимального решения. Однако его сложность составляет $O(n!)$, где n — количество городов.

Это делает метод непригодным для решения задачи с большим числом городов, так как количество маршрутов растет факториально. Например, для 10 городов существует 3 628 800 маршрутов.

2. «Жадные» алгоритмы

   «Жадные» алгоритмы решают задачу коммивояжёра, выбирая на каждом шаге тот путь, который кажется наилучшим. Они быстры и просты в реализации, но не всегда дают оптимальное решение, так как могут привести к неправильному выбору пути, который на начальных этапах выглядит хорошим, но в итоге оказывается неэффективным.

3. Генетические алгоритмы

   Генетические алгоритмы основаны на принципах естественного отбора и эволюции. Начальная популяция решений генерируется случайным образом, и в процессе «эволюции» создаются новые решения через скрещивание и мутацию. Эти алгоритмы могут найти хорошие приближенные решения, но не гарантируют нахождение оптимального маршрута.

4. Методы имитации отжига

   Методы имитации отжига используют случайные изменения для поиска минимального значения функции. Алгоритм постепенно «охлаждает» систему, уменьшая вероятность принятия менее оптимальных решений. Эти методы могут дать хорошие приближенные решения для большого числа городов, но их точность зависит от начальных условий и параметров алгоритма.

5. Динамическое программирование и метод ветвей и границ

   Метод динамического программирования решает задачу поэтапно, запоминая уже вычисленные решения, что снижает вычислительные затраты. Метод ветвей и границ использует стратегию разделения задачи на подзадачи и отбрасывания неэффективных путей. Эти методы быстрее полного перебора, но требуют значительных вычислительных ресурсов.

6. Муравьиный алгоритм

   Муравьиный алгоритм имитирует поведение муравьинных колоний при поиске пищи. Он использует принцип распределения информации через феромоны, которые муравьи оставляют на пути. Алгоритм эффектив-

но исследует пространство решений, избегая локальных минимумов и находя приближённые оптимальные решения за приемлемое время, но не гарантирует нахождение глобального оптимума.

В третьем разделе описывается реализация приложения. Для решения задачи коммивояжёра было решено создать веб-приложение, так как оно имеет несколько ключевых преимуществ по сравнению с традиционными настольными приложениями:

1. Веб-приложение доступно на любых платформах (Windows, macOS, Linux, мобильные устройства)
2. С помощью веб-технологий можно создать интуитивно понятный и удобный интерфейс для работы с методами решения задачи коммивояжёра. Веб-приложение позволяет пользователю взаимодействовать с системой в реальном времени, что важно для визуализации процесса.
3. Веб-приложение предоставляет отличные возможности для графической реализации задачи, что позволяет пользователям наглядно увидеть результаты работы различных алгоритмов и сравнить их эффективность.
4. Обновления веб-приложения можно вносить на сервере, и они автоматически становятся доступными всем пользователям. Это исключает необходимость вручную обновлять программу на каждом устройстве, как в случае с настольными приложениями.

С учетом этих факторов было принято решение реализовать приложение как веб-сайт с использованием Flask для серверной части и стандартных технологий веб-разработки (HTML, CSS, JavaScript) для создания интерфейса.

Архитектура приложения разделена на две основные части:

1. Серверная часть (Backend):
   - Flask — легковесный веб-фреймворк для Python, который используется для создания серверной части приложения. Он позволяет легко настроить обработку HTTP-запросов, управлять взаимодействием с клиентом и запускать сервер для тестирования.
2. Клиентская часть (Frontend):
   - HTML — язык разметки для создания структуры веб-страницы.

- CSS — используется для стилизации интерфейса приложения, улучшения внешнего вида.
- JavaScript — для реализации клиентской логики, включая обработку событий, асинхронное взаимодействие с сервером и динамическое обновление интерфейса.
- Canvas — элемент HTML5, используемый для визуализации города и маршрута, а также для интерактивного взаимодействия с пользователем.

Для реализации решения задачи коммивояжёра были выбраны методы: полный перебор, метод ветвей и границ, динамическое программирование, метод ближайшего соседа, алгоритм имитации отжига, генетический алгоритм, муравьиный алгоритм.

Для создания интерфейса было важно сделать его как можно более простым и интуитивно понятным, чтобы пользователи могли:

1. Добавлять города вручную, вводя их имена и координаты.
2. Генерировать случайные города для эксперимента с разными входными данными.
3. Выбирать один из методов решения задачи коммивояжёра.
4. Наблюдать за вычислениями через прогресс-бар.
5. Видеть результат в виде оптимального маршрута и его длины.
6. Использовать графическую визуализацию на холсте для отображения маршрута и городов.

Особое внимание уделено обработке ошибок, чтобы система информировала пользователя о некорректных данных.

Проект представляет собой удобное веб-приложение для решения задачи коммивояжёра с использованием различных алгоритмов оптимизации, доступное на всех устройствах.

Далее идет реализация приложения и выбранных методов решения задачи на Python:

Файл utils.py реализует вспомогательные функции для работы с расстояниями между городами. В частности, функция calculate_distance вычисляет Евклидово расстояние между двумя городами, а функция path_distance

вычисляет общую длину пути для маршрута, суммируя расстояния между всеми городами на маршруте.

Файл bruteforce.py реализует метод полного перебора, который генерирует все возможные маршруты между городами и выбирает маршрут с минимальной длиной. В случае наличия начального города, он будет фиксирован, а все другие города будут перебираться в различных комбинациях. Для генерации всех возможных перестановок используется метод itertools.permutations. Для каждого маршрута вычисляется его длина с помощью функции path_distance.

Файл branch_and_bound.py реализует метод ветвей и границ, который ищет решение с использованием рекурсии и отсечения неэффективных путей на основе текущего расстояния и ограничений задачи. Для каждого города проверяются все возможные пути, и если путь не приводит к оптимальному результату, он исключается из дальнейших вычислений. Этот метод позволяет значительно сократить количество проверяемых решений, что делает его более эффективным, чем полный перебор.

Файл dynamic_programming.py реализует метод динамического программирования, который использует кэширование для вычисления минимального пути. Он работает с масками, чтобы отслеживать, какие города уже были посещены, и минимизировать количество вычислений. Метод использует декоратор @lru_cache для хранения промежуточных результатов, что ускоряет выполнение программы. Алгоритм строит матрицу расстояний и применяет рекурсивную функцию для вычисления минимального пути.

Файл nearest_neighbor.py реализует алгоритм ближайшего соседа, который выбирает ближайший не посещённый город на каждом шаге, начиная с произвольного или выбранного города. Стартовый город и на каждом шаге выбирается ближайший город из оставшихся. Для расчёта расстояний между городами используется функция path_distance. Этот алгоритм реализует жадный подход к решению задачи, где на каждом шаге выбирается наиболее оптимальное решение в локальном контексте.

Файл simulated_annealing.py реализует метод имитации отжига, который начинается с случайного пути и выполняет случайные перестановки (изменения маршрута). Если новое решение лучше, оно принимается, а если хуже,

оно может быть принято с некоторой вероятностью, которая уменьшается с течением времени в процессе «охлаждения». Время охлаждения регулируется параметром cooling_rate. Этот алгоритм помогает искать глобальный минимум, избегая попадания в локальные минимумы за счёт вероятности принятия худших решений на начальных этапах.

Файл genetic.py реализует генетический алгоритм, который решает задачу коммивояжёра, используя принципы естественного отбора и эволюции. Вначале создаётся популяция случайных решений (маршрутов), которые затем проходят через этапы скрещивания и мутации для улучшения. Для каждого поколения вычисляются значения приспособленности (fitness), которые показывают, насколько хорош каждый маршрут. Алгоритм использует турнирный отбор для выбора «родителей» для создания следующего поколения, а затем выполняются кроссовер и мутация для генерации новых маршрутов.

Файл ant_colony.py реализует муравьиный алгоритм, который имитирует поведение муравьёв, оставляющих феромоны на пути. Путь с более высокими феромонами имеет большую вероятность быть выбранным. Используется несколько муравьёв, каждый из которых выбирает путь на основе вероятности, которая зависит от феромонов и расстояний. После каждой итерации феромоны обновляются, чтобы улучшить выбранный путь. Этот алгоритм позволяет эффективно исследовать пространство возможных решений, избегая локальных минимумов.

Файл app.py реализует серверную часть приложения с использованием Flask. Этот файл содержит основной сервер, который обрабатывает HTTP-запросы с клиентской стороны. Он позволяет взаимодействовать с приложением через RESTful API, обрабатывая запросы на решение задачи коммивояжёра с использованием выбранного метода.

Далее рассматривается реализиция интерфейса и работа с клиентской частью.

Файл index.html реализует основную структуру веб-страницы, а файл style.css отвечает за оформление и стилизацию этих элементов. В интерфейсе предусмотрены формы для добавления городов, визуализация графа, выбор метода решения задачи коммивояжера и отображение результатов.

Файл script.js реализует клиентскую часть приложения, отвечающую за взаимодействие с пользователем. Он управляет всей логикой на frontend, включая добавление городов, выбор метода решения задачи, отправку запросов на сервер, отображение результатов и визуализацию данных.

Основные действия, которые доступны пользователю:

- Пользователь вводит имя и координаты города, после чего город добавляется в список в соответствии с рисунком 1 и отображается на графе в соответствии с рисунком 2.

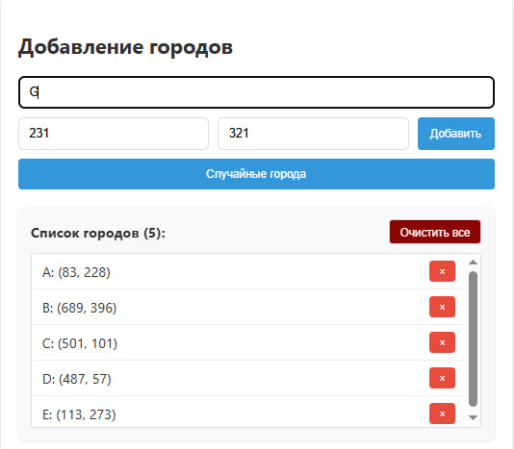

Рисунок 1 — Список городов.

Рисунок 2 — Визуализация городов.

- После выбора метода и ввода данных, отправляется запрос на сервер для выполнения расчёта. Результаты (путь и расстояние) отображаются на экране в соответствии с рисунком 3.

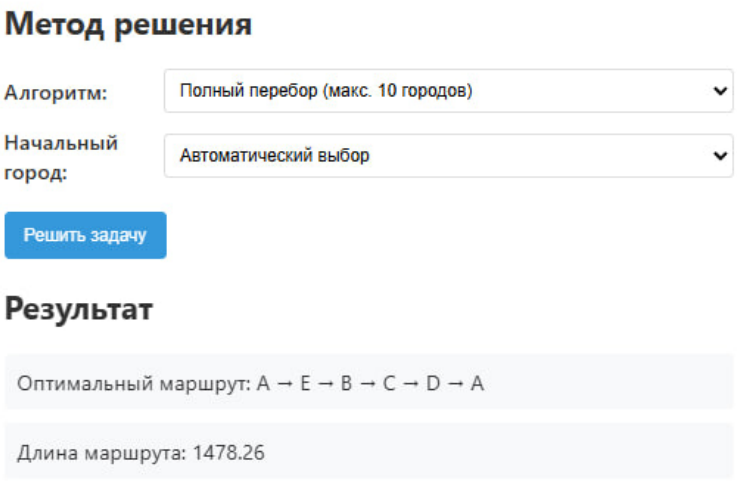

Рисунок 3 — Выбор метода и результаты вычисления.

- Используется элемент <canvas>, где отрисовывается маршрут между городами в соответствии с рисунком 4.

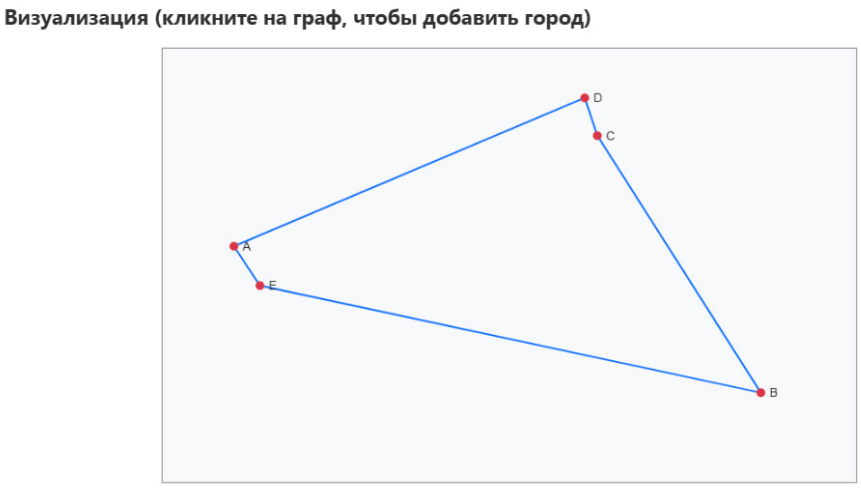

Рисунок 4 — Отрисованный путь на графе.

Реализованное приложение выглядит в соответствии с рисунком 5.

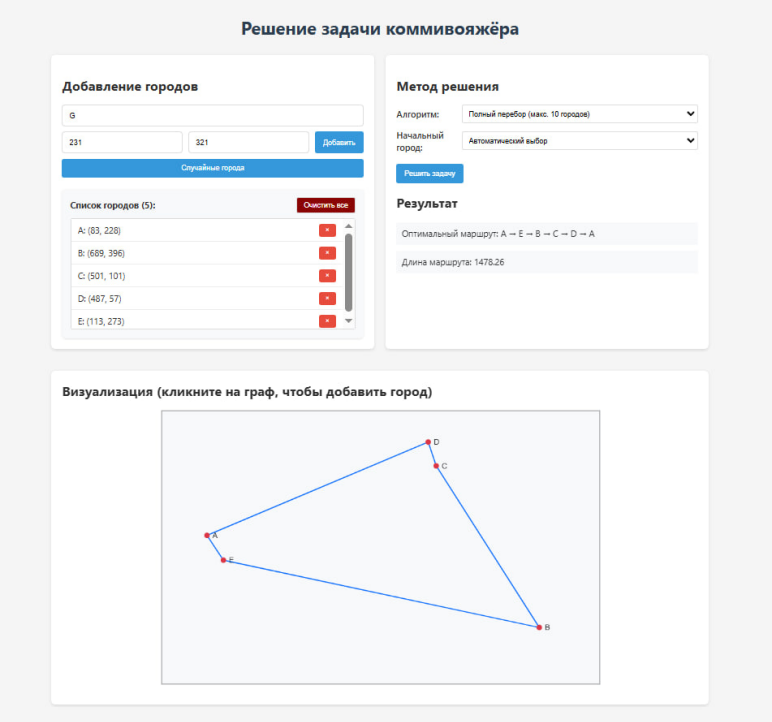

Рисунок 5 — Веб-приложение для решения задачи коммивояжёра.

В четвертом разделе описывается тестирование и сравнение алгоритмов. Тест проводился на наборах из 5, 10 и 100 городов.

Сначала проводится тестирование на наборе из 5 городов.

Данные о городах:

A: (96, 441); B: (531, 335); C: (697, 307); D: (617, 406); E: (155, 287);

Для теста на 5 городах с фиксированным начальным городом A, все методы решения задачи коммивояжёра выдали оптимальный маршрут за приемлемое время работы программы, за исключением метода ближайших соседей. Результаты, которые вывели методы: Путь: $A \to D \to C \to B \to E \to A$. Длина маршрута: 1361.66.

Результат, который вывел метод ближайшего соседа: Путь: $A \to E \to B \to D \to C \to A$. Длина маршрута: 1398.58.

Далее проводится тестирование для 10 городов.

Добавляются города:

F: (368, 169); G: (86, 130); H: (629, 88); I: (356, 456); J: (254, 104);

В этом случае полный перебор показал значительное увеличение времени работы по сравнению с другими методами, а метод ближайшего соседа выда-

вал неоптимальный маршрут, другие же методы показывали все еще быстрое время работы и оптимальный результат.

Оптимальный маршрут:

- Путь: $A \to E \to G \to J \to F \to H \to C \to D \to B \to I \to A$.
- Длина маршрута: 1851.24

Далее проводится тест для метода ближайшего соседа, имитации отжига, генетического алгоритма и муравьиного алгоритма на наборе из 100 городов, где точные методы уже не справляются.

Результат тестирования показал:

1. Метод ближайшего соседа : Длина пути 5035.25, время работы : 0.1 сек.
2. Алгоритм имитации отжига : Длина пути 7457.78, время работы : 1 сек.
3. Генетический алгоритм: Длина пути 9767.65, время работы 3 сек.
4. Муравьиный алгоритм: Длина пути 4487.58, время работы 4 сек.

Если основной задачей является минимизация пути, то муравьиный алгоритм является наилучшим выбором, несмотря на его более высокое время работы. В случае, когда время критично, но точность решения не так важна, метод ближайшего соседа будет лучшим вариантом. Алгоритм имитации отжига и генетический алгоритм не оправдали себя в этом случае, так как они уступили даже методу ближайшего соседа как по качеству решения, так и по времени. Но стоит учитывать, что результаты алгоритма имитации отжига, генетического и муравьиного алгоритмов зависит от подбора гиперпараметров.

Заключение. Задача коммивояжёра — это классическая и важная проблема в теории графов и оптимизации, имеющая широкое практическое значение в таких областях, как логистика, транспортировка товаров и другие. Несмотря на свою вычислительную сложность, разработка эффективных методов решения задачи, как точных, так и приближенных, является важной задачей для повышения эффективности транспортировки и маршрутизации.

Цели и задачи работы были успешно достигнуты: проведен анализ теоретических аспектов задачи, реализованы различные алгоритмы её решения, включая точные и приближенные методы. Разработан веб-интерфейс на базе Flask, позволяющий пользователю тестировать алгоритмы на реальных

данных и сравнивать их эффективность. Тестирование показало работоспособность и эффективность алгоритмов.

Работа позволила получить глубокие знания в области веб-разработки и алгоритмов оптимизации. Разработанное приложение стало эффективным инструментом для визуализации методов и анализа их результатов. Полученные результаты будут полезны как для образовательных целей, так и для специалистов в области оптимизации и логистики.