**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Решение задачи о коммивояжере с помощью муравьиного алгоритма»**

**Студент гр. 23Б15-пу**

**Абрахин Е.Д.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2025 г.**

Оглавление

1. [Цель работы 3](#_Toc195382143)
2. [Описание задачи (формализация задачи) 3](#_Toc195382144)
3. [Теоретическая часть 4](#_Toc195382145)
4. [Основные шаги программы 7](#_Toc195382146)
5. [Блок схема программы 9](#_Toc195382147)
6. [Описание программы 9](#_Toc195382148)
7. [Рекомендации пользователя 12](#_Toc195382149)
8. [Рекомендации программиста 12](#_Toc195382150)
9. [Исходный код программы 13](#_Toc195382151)
10. [Контрольный пример 14](#_Toc195382152)
11. [Анализ результатов работы алгоритма 17](#_Toc195382153)
12. [Вывод 20](#_Toc195382154)
13. [Источники 21](#_Toc195382155)

# Цель работы

Целью данной лабораторной работы является исследование муравьиного алгоритма для решения задачи коммивояжера, а также анализ эффективности его модифицированной версии с использованием шаблонов. В ходе работы будет разработана программа, реализующая муравьиный алгоритм, с возможностью визуализации найденного маршрута. Программа будет протестирована на заданном взвешенном орграфе для оценки качества полученного решения и сравнения его с оптимальным маршрутом.

# Описание задачи (формализация задачи)

1. Изучение задачи коммивояжера и муравьиного алгоритма.

Ознакомиться с муравьиным алгоритмом, его применением для поиска приближенного решения и с понятием феромонных следов.

1. Разработка программы для нахождения кратчайшего гамильтонова цикла.

Написать программу, реализующую муравьиный алгоритм для задачи коммивояжера. Добавить возможность использования шаблонов в качестве модификации алгоритма. Реализовать визуализацию построенного маршрута на графе.

1. Тестирование программы на взвешенном орграфе.

Провести тестирование программы на контрольном примере, представленном во взвешенном орграфе. Сравнить полученный маршрут с оптимальным решением, оценив точность и эффективность метода.

1. Анализ результатов

Оценить влияние введения шаблонов на качество решения. Проанализировать поведение алгоритма при различных параметрах (количество муравьёв, уровень испарения, коэффициент усиления шаблонов и др.).

# Теоретическая часть

**Муравьиный алгоритм**

Муравьиный алгоритм (Ant Colony Optimization, ACO) — это метаэвристический метод оптимизации, вдохновлённый коллективным поведением реальных муравьёв при поиске кратчайшего пути от муравейника к источнику пищи. В основе алгоритма лежит идея о том, что муравьи оставляют феромонный след на маршруте, усиливая вероятность его выбора другими особями. Со временем наилучшие маршруты получают наибольшее усиление и формируют основу для приближённого поиска оптимального решения.

**Параметры алгоритма и их роль**

1. **— влияние феромона**

Параметр определяет, насколько сильно при выборе следующего узла муравей ориентируется на \*\*уровень феромона\*\* на рёбрах.

– При = 0 муравьи полностью игнорируют феромон и двигаются только по расстоянию.

– При больших значениях решение становится более жадным, алгоритм быстро концентрируется на одном маршруте. Коэффициент испарения

1. **— влияние расстояния**

Параметр регулирует, насколько муравей учитывает эвристическую информацию — обычно это обратное расстояние:

– При = 0 выбор основан только на феромоне.

– При высоких муравьи тяготеют к ближайшим вершинам.

1. **— коэффициент испарения феромона**

Параметр (0 < < 1) определяет долю испарения феромона после каждой итерации:

Меньшие значения позволяют дольше сохранять старые следы, а большие значения — быстрее «забывать» неэффективные маршруты.

1. **— количество феромона (вклад муравья)**

После прохождения маршрута муравей откладывает феромон вдоль своего пути пропорционально качеству маршрута. Если длина маршрута

то вклад определяется как:

**Вероятность перехода**

После Вероятность того, что муравей в вершине 𝑖 выберет переход в вершину 𝑗 определяется по формуле:

где:

– — количество феромона на ребре

– — эвристическая привлекательность (обратное расстояние)

– — коэффициенты влияния феромона и расстояния

**Модификация алгоритма**

В модифицированной версии алгоритма добавляется механизм запоминания часто используемых рёбер (паттернов), которые получают дополнительное усиление феромоном. Это позволяет укреплять маршруты, повторяющиеся среди лучших муравьёв.

1. После прохождения всех маршрутов формируется словарь, сохраняющий вхождения количества ребер в маршруты.
2. На этапе обновления феромона учитывается это количество:

где:

– 0.1 — коэффициент усиления шаблонов

– — количество повторений ребра в маршрутах

– — базовое количество откладываемого феромона

**Принцип работы**

1. Инициализируется граф с равными начальными феромонами на всех рёбрах.
2. На каждой итерации множество муравьёв независимо строит маршруты, выбирая рёбра на основе вероятности перехода.
3. Завершив маршрут, каждый муравей оставляет феромоны на пройденных рёбрах — чем короче путь, тем больше усиление.
4. Применяется испарение феромонов — значения уменьшаются на всех рёбрах, не усиливающихся текущими решениями.
5. Повторяется процесс до достижения заданного количества итераций или сходимости маршрута.

**Оценка эффективности**

Основные параметры, влияющие на качество решения:

1. Скорость сходимости

Благодаря феромонной памяти и выбору маршрутов с учётом шаблонов, алгоритм быстро концентрируется на хорошем приближении к оптимуму. Однако слишком сильное усиление может привести к преждевременной сходимости.

1. Качество решения

Механизм шаблонов помогает избегать локальных минимумов, удерживая полезные под маршруты даже при высоком испарении феромонов.

1. Число итераций

Алгоритм останавливается после достижения заданного количества итераций, что позволяет управлять временем работы и качеством результатов.

# Основные шаги программы

Основные шаги программы The-ant\_algorithm.pyw

1. Запуск программы:

Инициализация графического интерфейса (GUI) с элементами управления:

Область визуализации графа.

Таблица для редактирования весов рёбер.

Поля ввода параметров.

Поле для вывода результатов.

Флажок для включения модификации.

Кнопки управления: «Рассчитать», «Отмена», «Очистить», «Загрузить», «Сохранить».

1. Настройка графа:

Добавление вершин кликом по области визуализации.

Создание рёбер путем выбора двух вершин (автоматически рассчитывается расстояние).

Отображение данных о вершинах и рёбрах в таблице.

Возможность редактирования весов рёбер.

1. Запуск алгоритма:

При нажатии кнопки «Рассчитать» выполняется алгоритм поиска пути.

Если включена модификация, алгоритм использует шаблоны.

1. Поиск решения:

Генерация начального маршрута.

Итеративное улучшение маршрута методом муравьиного алгоритма.

Обновление феромонов.

Испарение феромонов.

Повторение процесса.

1. Обновление интерфейса:

Визуализация найденного пути на графе.

Заполнение таблицы актуальными данными.

Вывод результатов: маршрут, длина пути, время выполнения.

1. Дополнительные возможности:

Отмена последнего действия (удаление вершины или ребра).

Очистка графа для нового расчёта.

Загрузка и сохранение графа в Excel.

# Блок схема программы

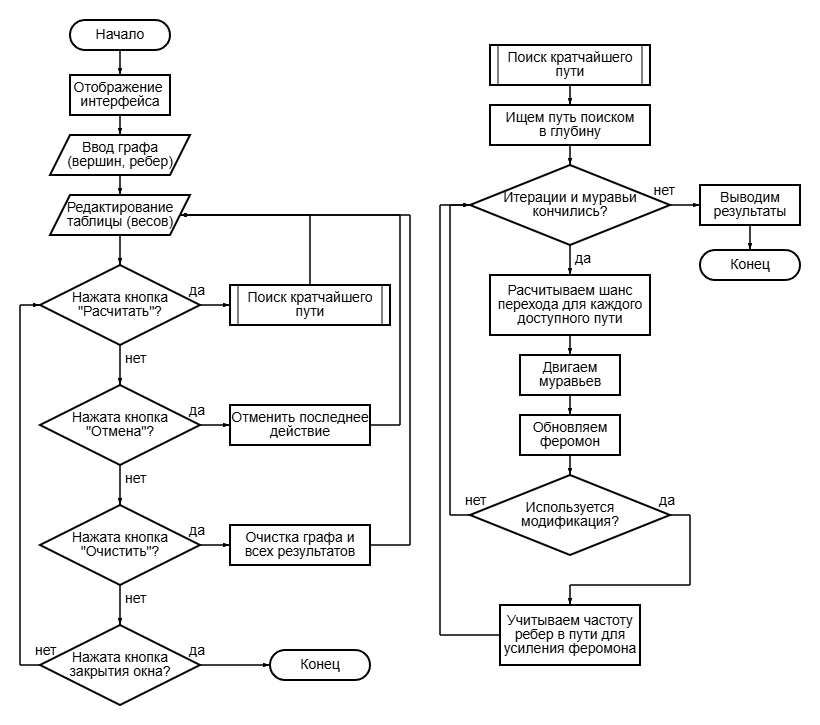


Рис 1. Блок-схема The-ant\_algorithm.py

# Описание программы

Программа написана на языке Python 3.12.6 и использует библиотеки numpy [[2]](#_Источники_1) и PyQt5 [[3]](#_Источники_1) для реализации графического интерфейса решения задачи о коммивояжере с использованием алгоритма ближайшего соседа. В программе реализован только один класс – **TSPApp**, который отвечает за создание интерфейса, обработку графических событий, построение графа, вычисление кратчайшего пути и визуализацию результатов. Класс включает в себя 18 методов, каждый из которых выполняет свою задачу в процессе работы приложения:

Таблица 1. TSPApp

|  |  |  |
| --- | --- | --- |
| Метод | Описание | Возвращаемое значение |
| \_\_init\_\_ | Инициализация интерфейса и параметров приложения. | None |
| initUI | Настройка графического интерфейса приложения, создание и размещение виджетов. | None |
| addEdge | Добавляет ребро между двумя выбранными вершинами, вычисляя расстояние между ними. | None |
| addNode | Добавляет новую вершину в граф по заданной позиции. | None |
| addNodeOrEdge | Определяет, следует ли добавить новую вершину или создать ребро между существующими вершинами, в зависимости от положения курсора. | None |
| clearGraph | Очищает граф, таблицу ребер, историю действий и текстовое поле с результатом. | None |
| drawArrow | Отрисовывает стрелку между вершинами, корректируя начало и конец линии, чтобы не накладываться на узлы. | None |
| drawSolution | Отображает найденный маршрут и вершины графа на отдельной сцене для визуализации решения. | None |
| findClickedNode | Определяет, на какую вершину произведён клик; возвращает идентификатор вершины или None, если клик не попал на узел. | int или None |
| getDistance | Возвращает расстояние между двумя вершинами, если соответствующее ребро существует, иначе — float('inf'). | float |
| generateTestGraph | Генерирует случайный тестовый граф с вершинами и рёбрами, а также сохраняет его в Excel. | None |
| loadGraphFromExcel | Загружает граф из файла Excel, обновляет список вершин и рёбер. | None |
| lockColumns | Делает первые два столбца таблицы недоступными для редактирования. | None |
| redrawGraph | Перерисовывает граф и таблицу ребер после внесённых изменений. | None |
| saveGraphToExcel | Сохраняет текущий граф (матрицу смежности) в файл Excel | None |
| solveTsp | Вычисляет кратчайший путь (гамильтонов цикл) с помощью муравьиного, измеряет время выполнения и выводит результаты. | None |
| undoAction | Отменяет последнее действие (добавление вершины или ребра) и обновляет граф. | None |
| updateEdgeWeight | Обновляет вес ребра в таблице при изменении значения, если ввод корректен. | None |

# Рекомендации пользователя

Программа позволяет решить задачу о коммивояжере с помощью муравьиного алгоритма. С ее помощью можно построить граф, добавить узлы и ребра, а затем найти кратчайший гамильтонов цикл.

1. Запустите программу, например, выполнив команду:

python The-ant\_algorithm.pyw

1. В графическом интерфейсе:
   * Добавление узлов:

Щелкайте на области графа для создания новых узлов.

* + Добавление ребер:

Для соединения узлов выберите сначала один узел, затем второй – программа автоматически создаст ребро между ними.

* + Редактирование:

При необходимости измените вес ребра в таблице.

* + Ввод параметров:

При необходимости измените параметры в поле над выводом результатов.

* + Выбор модификации:

Для использования модификации шаблонов установите галочку в «Использовать модификацию».

1. Нажмите кнопку **«Рассчитать»** для выполнения алгоритма. Программа вычислит кратчайший путь (гамильтонов цикл) и отобразит:
   * В поле **«Рассчитанный путь»** — последовательность вершин кратчайшего маршрута.
   * На графике — построенный маршрут с выделенными ребрами.
2. Используйте кнопку **«Отмена»** для удаления последнего действия или **«Очистить»** для сброса всего графа.

# Рекомендации программиста

Для корректной работы программы убедитесь, что установлены следующие компоненты и выполнены необходимые шаги:

* Требования:
  + Python версии 3.12.0 или выше.
  + Необходимые библиотеки: PyQt5, numpy.
* Шаги по установке:
  + Выполните команду для запуска программы:

python The-ant\_algorithm.pyw

* + Проверьте, что графический интерфейс запускается корректно и результаты алгоритма (нахождение кратчайшего гамильтонова цикла) отображаются в соответствии с заданием.

# Исходный код программы

<https://github.com/FasterXaos/Algorithms_and_Data_Structures>

# Контрольный пример

The-ant\_algorithm.py

1. Запуск программы и формирование графа

Для запуска программы откройте файл, содержащий код класса TSPApp. Программа откроет графический интерфейс, где в левой части окна расположены таблица для ввода данных о ребрах и панель для отображения вычисленного маршрута, в правой – область для построения графа (Рис. 2).

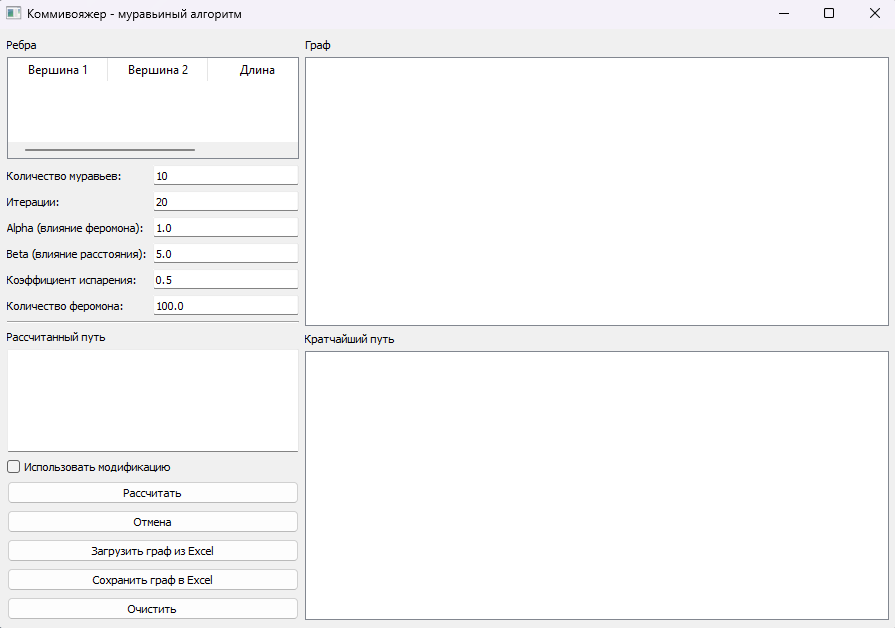


Рис 2. Пример окна программы

1. Ввод исходных данных

После запуска программы можно добавить вершины и соединить их ребрами следующим образом:

* Добавление вершин:

щёлкните в области графа (правой части окна) для создания новой вершины.

* Соединение вершин:

после выбора первой вершины щёлкните по другой, чтобы добавить ребро между ними. Вес ребра вычисляется автоматически как расстояние между центрами вершин.

1. Запуск алгоритма

После формирования графа нажмите кнопку «Рассчитать». Программа выполнит вычисление кратчайшего гамильтонова цикла с помощью муравьиного алгоритма, определяя маршрут, проходящий через все вершины и возвращающийся в исходную точку (Рис. 3). Если поставить галочку «Использовать модификацию», то будет использоваться модификация шаблонов.

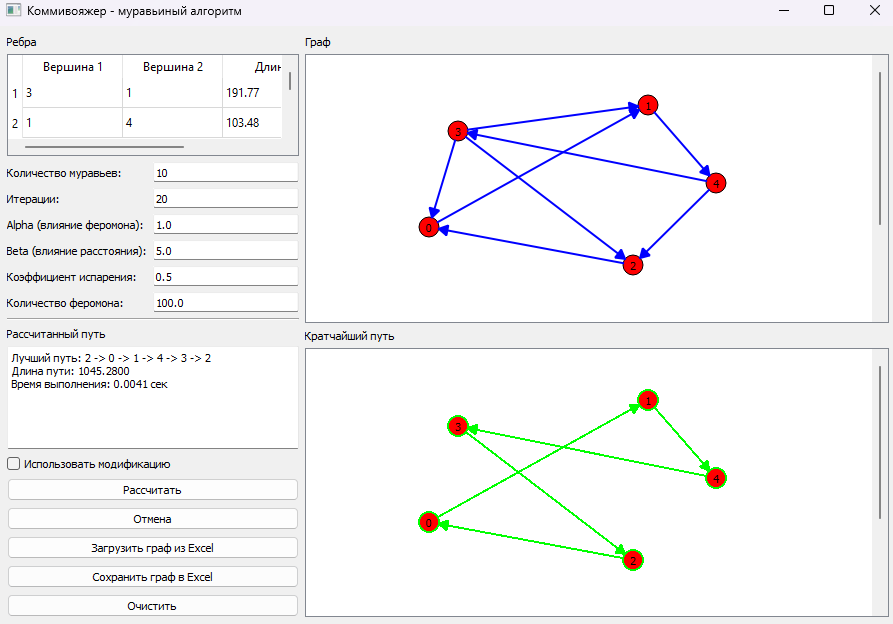


Рис 3. Пример результатов программы

1. Просмотр результатов

После завершения расчёта в нижней части окна будут выведены следующие данные:

* Найденный кратчайший маршрут (последовательность вершин);
* Общая длина маршрута с заданной точностью;
* Время выполнения алгоритма.

Дополнительно в правой части окна отображается графическое представление маршрута с выделенными вершинами и соединяющими их стрелками.

1. Загрузка и сохранение графа.

Для сохранения нарисованного графа или загрузки его из файла Excel нужно нажать кнопки «Загрузить граф в Excel» и «Сохранить граф в Excel». Таблица смежность должна иметь следующий вид для корректной загрузки:

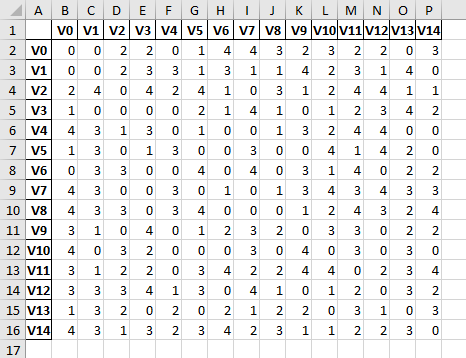


Рис 4. Пример задания таблицы графа

# Анализ результатов работы алгоритма

Для анализа работы муравьиного алгоритма и его модификации приведены примеры результатов для нахождения гамильтонова цикла методом ближайших соседей и методом имитации отжига на разных графах:

Таблица 2. Тесты метода ближайших соседей

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номер графа | Вершины | Расстояние без модификации | | Время без модификации | Расстояние с модификацией | Время с модификацией |
| 1 | 6 | 12 | | 0.0001 сек | 12 | 0.0002 сек |
| 2 | 6 | Не удалось найти путь | | | 13 | 0.0002 сек |
| 3 | 6 | 9 | | 0.0001 сек | 9 | 0.0002 сек |
| 4 | 6 | 11 | | 0.0001 сек | 11 | 0.0002 сек |
| 1 | 15 | 27 | | 0.0004 сек | 18 | 0.0061 сек |
| 2 | 15 | Не удалось найти путь | | | 18 | 0.0059 сек |
| 3 | 15 | 21 | | 0.0004 сек | 19 | 0.0060 сек |
| 4 | 15 | 24 | | 0.0005 сек | 20 | 0.0063 сек |
| 1 | 30 | 43 | | 0.0054 сек | 34 | 0.1655 сек |
| 2 | 30 | Не удалось найти путь | | | 32 | 0.1635 сек |
| 3 | 30 | 37 | | 0.0057 сек | 33 | 0.1643 сек |
| 4 | 30 | Не удалось найти путь | | | 32 | 0.1607 сек |
| 1 | 50 | Не удалось найти путь | | | 52 | 1.9866 сек |
| 2 | 50 | 59 | 0.0423 сек | | 52 | 1.9910 сек |
| 3 | 50 | 58 | 0.0396 сек | | 52 | 1.9427 сек |
| 4 | 50 | Не удалось найти путь | | | 54 | 2.0333 сек |

Таблица 3. Тесты алгоритма имитации отжига

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер графа | Вершины | Расстояние без модификации | Время без модификации | Расстояние с модификацией | Время с модификацией |
| 1 | 6 | 13 | 0.1877 сек | 13 | 0.2042 сек |
| 2 | 6 | 13 | 0.1578 сек | 13 | 0.1731 сек |
| 3 | 6 | 9 | 0.1638 сек | 10 | 0.1622 сек |
| 4 | 6 | 9 | 0.1805 сек | 9 | 0.1876 сек |
| 1 | 15 | 19 | 0.8301 сек | 20 | 0.8162 сек |
| 2 | 15 | 18 | 0.8063 сек | 19 | 0.8463 сек |
| 3 | 15 | 18 | 0.7855 сек | 17 | 0.8522 сек |
| 4 | 15 | 21 | 0.8295 сек | 21 | 0.8012 сек |
| 1 | 30 | 35 | 4.6611 сек | 33 | 4.6212 сек |
| 2 | 30 | 35 | 4.6814 сек | 37 | 4.7954 сек |
| 3 | 30 | 34 | 4.5799 сек | 32 | 4.6005 сек |
| 4 | 30 | 34 | 4.6718 сек | 36 | 4.4452 сек |
| 1 | 50 | 55 | 19.3073 сек | 56 | 19.0850 сек |
| 2 | 50 | 55 | 19.4550 сек | 57 | 19.0367 сек |
| 3 | 50 | 56 | 19.2024 сек | 57 | 19.5447 сек |
| 4 | 50 | 54 | 19.2696 сек | 57 | 19.8725 сек |

Тесты проводились с количеством муравьев = 10, количеством итераций = 20, Alpha = 1, Beta = 3, Коэффициентом испарения = 0.5 и количеством феромона = 100

Таблица 4. Тесты муравьиного алгоритма

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер графа | Вершины | Расстояние без модификации | Время без модификации | Расстояние с модификацией | Время с модификацией |
| 1 | 6 | 12 | 0.0076 сек | 12 | 0.0076 сек |
| 2 | 6 | 13 | 0.0084 сек | 13 | 0.0088 сек |
| 3 | 6 | 9 | 0.0072 сек | 9 | 0.0073 сек |
| 4 | 6 | 9 | 0.0082 сек | 9 | 0.0085 сек |
| 1 | 15 | 18 | 0.1454 сек | 18 | 0.1415 сек |
| 2 | 15 | 17 | 0.1427 сек | 17 | 0.1423 сек |
| 3 | 15 | 16 | 0.1412 сек | 15 | 0.1410 сек |
| 4 | 15 | 18 | 0.1483 сек | 19 | 0.1457 сек |
| 1 | 30 | 31 | 2.0268 сек | 30 | 1.9863 сек |
| 2 | 30 | 30 | 1.8967 сек | 31 | 1.8985 сек |
| 3 | 30 | 30 | 1.9810 сек | 30 | 1.9481 сек |
| 4 | 30 | 30 | 1.9378 сек | 31 | 1.9523 сек |
| 1 | 50 | 51 | 14.4465 сек | 53 | 14.4295 сек |
| 2 | 50 | 50 | 15.2289 сек | 52 | 14.5623 сек |
| 3 | 50 | 52 | 15.2923 сек | 52 | 14.6222 сек |
| 4 | 50 | 51 | 14.5528 сек | 51 | 14.3516 сек |

Таблица 5. Средние значения методов

|  |  |  |  |
| --- | --- | --- | --- |
| Вершины | Метод/Алгоритм | Расстояние без модификации | Расстояние с модификацией |
| 6 | Ближайших сосед | 10.67 | 11.25 |
| 15 | Ближайших сосед | 24 | 18.75 |
| 30 | Ближайших сосед | 40 | 32.75 |
| 50 | Ближайших сосед | 58.5 | 52.5 |
| 6 | Имитация отжига | 11 | 11.25 |
| 15 | Имитация отжига | 19 | 19.25 |
| 30 | Имитация отжига | 34.5 | 34.5 |
| 50 | Имитация отжига | 55 | 56.75 |
| 6 | Муравьиный | 10.75 | 10.75 |
| 15 | Муравьиный | 17.25 | 17.25 |
| 30 | Муравьиный | 30.25 | 30.5 |
| 50 | Муравьиный | 51 | 52 |

Муравьиный алгоритм, как и алгоритм имитации отжига показывает результаты по расстоянию значительно лучше, чем метод ближайшего соседа и слегка превосходит его модификацию, но затрачивает намного больше времени из-за необходимости большого числа расчетов и итераций. В сравнении с имитацией отжига муравьиный алгоритм находит решение лучше, делая это за меньшее время (особенно на небольших графах).

Модификация шаблонов почти идентична стандартному алгоритму и немного хуже себя проявляет на больших графах, скорее всего, это происходит из-за более быстрого схождения, которое иногда попадает в решение чуть хуже. Поскольку отличие модификации — это учет частоты вхождения ребер в пути и дополнительная прибавка феромона на частые, то фактическое время в обоих случаях сопоставимо.

# Вывод

В ходе работы был реализован муравьиный алгоритм для решения задачи коммивояжера, а также создан графический интерфейс для визуализации построения графа и вывода решения. Экспериментальные результаты показали, что метод эффективнее справляется с поиском оптимального решения нежели метод ближайшего соседа и немного лучше, чем алгоритм имитации отжига, затрачивая при этом меньше времени.

Преимуществом алгоритма являются его точность и скорость схождения. Его модификация позволяет сходится к решению быстрее, слегка жертвуя точностью решения.

# Источники

1. NumPy documentation // numpy.org URL: https://numpy.org/doc/stable/ (дата обращения: 04.03.2025).
2. PyQt5 Reference Guide // www.riverbankcomputing.com URL: https://www.riverbankcomputing.com/static/Docs/PyQt5/ (дата обращения: 04.03.2025).
3. Алгоритм имитации отжига // ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Алгоритм\_имитации\_отжига (дата обращения: (04.03.2025).