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»

Модификация – «Блуждающая колония»

Студент гр. 23Б15-пу
Сериков К.Г.

Преподаватель
Дик А.Г.

Санкт-Петербург
2025 г.

Оглавление

Цель работы

Целью данной лабораторной работы является изучение и реализация муравьиного алгоритма для решения задачи коммивояжёра (TSP). В рамках данной работы была разработана программа, которая находит кратчайший гамильтонов цикл при помощи муравьиного алгоритма и предоставляет возможность переключения между обычной и модифицированной версиями алгоритма.

Описание задачи (формализация задачи)

В данной лабораторной работе необходимо исследовать эффективность муравьиного алгоритма для решения задачи коммивояжёра (TSP).

Формализация задачи включает следующие компоненты:

1. **Изучение особенностей муравьиного алгоритма:** Исследовать принципы работы муравьиного алгоритма, включая его применение для решения задачи коммивояжера. Рассмотреть эффективность алгоритма.
2. **Разработка алгоритма для нахождения кратчайшего гамильтонова цикла:** Создать программу, реализующую муравьиный алгоритм для поиска решения задачи коммивояжера. Реализовать визуализацию исходного и итогового графов.
3. **Тестирование программы на взвешенном ориентированном графе:** Провести тестирование программы на контрольных примерах. Оценить точность и эффективность муравьиного алгоритма для решения задачи коммивояжёра (TSP).
4. **Анализ эффективности алгоритма**: Провести сравнительный анализ работы муравьиного алгоритма с модификацией и без неё, а также с алгоритмом ближайшего соседа и алгоритмом имитации отжига. Исследовать влияние структуры графа на эффективность. Оценить влияние модификации на скорость и точность нахождения кратчайшего гамильтонова цикла.

Теоретическая часть

Задача коммивояжера

Задача коммивояжёра (Traveling Salesman Problem) является одной из классических задач комбинаторной оптимизации. Она формулируется следующим образом: имеется множество городов и известны расстояния между каждой парой городов. Требуется найти кратчайший маршрут, который проходит через каждый город ровно один раз и возвращается в начальную точку.

Муравьиный алгоритм

Муравьиный алгоритм — это метаэвристический метод оптимизации, вдохновлённый поведением муравьёв в природе, которые находят кратчайшие пути до источников пищи с помощью феромонов. Алгоритм особенно эффективен для решения задач дискретной оптимизации, таких как TSP.

Блуждающая колония

Это специальная модификация муравьиного алгоритма, где вся колония начинает движение из одного случайно выбранного города на каждой итерации. Это позволяет более интенсивно исследовать окрестности разных частей графа и уменьшает вероятность преждевременной сходимости к субоптимальным решениям.

Принцип работы

1. Инициализация популяции феромонов для всех рёбер
2. Построение решений с помощью формулы.
3. Обновление феромонов:
4. Рассчитывается разница в длине маршрута
5. Цикл повторяется до достижение максимального числа итераций

Основные шаги программы

1. **Запуск программы:** Инициализируется графический интерфейс с элементами управления.
2. **Настройка параметров:** Пользователь в интерфейсе может задать граф вручную или загрузить JSON файл, также пользователь выбирает между стандартным и модифицированным режимами алгоритма.
3. **Запуск алгоритма:** При нажатии на кнопку "Рассчитать" программа запускает выполнение алгоритма имитации отжига на заданном графе.
4. **Начало работы алгоритма:** Создаётся популяция из n муравьёв и инициализируются феромоны.
5. **Построение маршрута:** В обычном режиме каждый муравей стартует из случайного узла. В режиме "блуждающая колония" выбирается общий стартовый узел. Каждый муравей строит путь. Начинает из заданного города. Последовательно выбирает следующий город по формуле. Посещает все города ровно по одному разу.
6. **Завершение цикла:** Алгоритм заканчивает выполнение, когда достигается максимальное число итераций. Лучший найденный путь сохраняется как результат.
7. **Отображение результатов:** Программа отображает кратчайший гамильтонов цикл в виде графа и длину это пути и вершины маршрута.

Блок схема программы

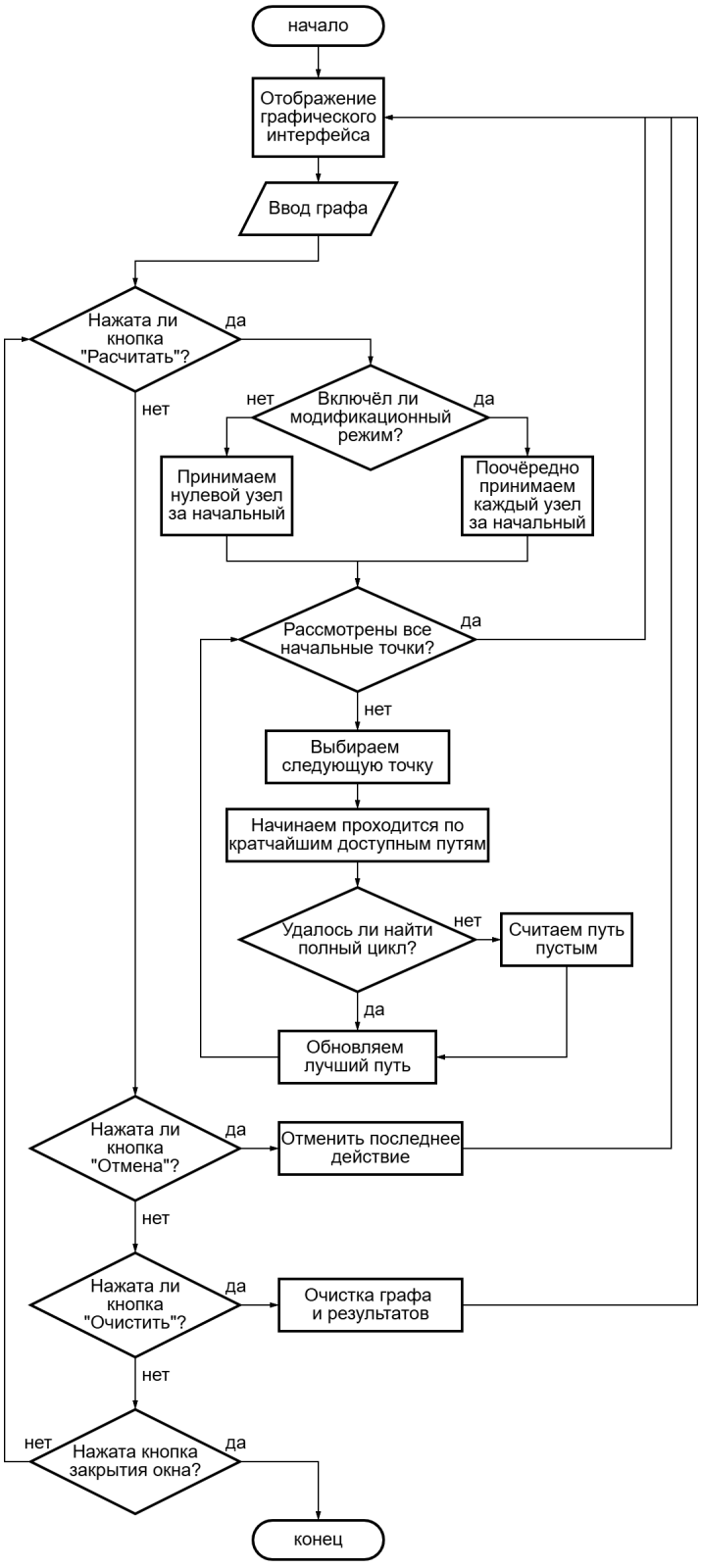

Рис 1. Блок-схема основной программы

Описание программы

Программная реализация написана на языке Python 3.13.0 с использованием следующих библиотек: tkinter [1], numpy [2], time [3], math [4], json [5] и random [6]. Программа организована в виде графического интерфейса для решения задачи коммивояжёра (TSP) с помощью алгоритма ближайшего соседа. В программе реализован только один класс – TSPApp, который отвечает за работу всей программы. В процессе разработки программы использовался main.py, включающий 17 функций, каждая из которых имеет чётко определённое назначение:

Таблица 1. main.py

Функция	Описание	Возвращаемое значение
__init__	Инициализация интерфейса.	None
add_node_or_edge	Добавляет вершину или ребро в зависимости от места клика	None
add_node	Добавляет новую вершину в заданной позиции	None
add_edge	Добавляет ребро между двумя вершинами с вычислением евклидова расстояния	None
find_clicked_node	Находит вершину по координатам клика	int или None
redraw_graph	Перерисовывает граф и обновляет таблицу рёбер	None
get_distance_matrix	Создает матрицу расстояний между всеми вершинами графа	numpy.ndarray
calculate_path_length	Вычисляет общую длину гамильтонова цикла	float
solve_tsp	Реализует муравьиный алгоритм	None

generate_ant_paths	Генерирует пути для всех муравьев из одного стартового города (для модификации "блуждающая колония")	List
generate_ant_path	Генерирует путь для одного муравья по правилам муравьиного алгоритма	List
select_next_city	Выбирает следующий город для муравья на основе вероятностного правила	int
visualize_solution	Визуализирует найденный маршрут на отдельном холсте	None
load_graph	Загружает граф из JSON-файла	None
get_distance	Возвращает расстояние между двумя вершинами	float
undo_action	Отменяет последнее действие	None
clear_graph	Полностью очищает граф, историю изменений и результаты	None

Рекомендации пользователя

Программа позволяет запустить муравьиный алгоритм для решения задачи коммивояжёра (TSP) с использованием графического интерфейса.

1. **Запуск программы**: Откройте программу с помощью Python 3.13.0, чтобы инициализировать интерфейс. Интерфейс позволяет нарисовать граф и выбирать режим: стандартный или модифицированный.
2. **Настройка параметров**: В графическом интерфейсе реализованы следующий функции:
   - Добавление узлов: Для создания нового узла кликните левой кнопкой мыши на область графа.
   - Добавление ребер: Для создания ребра между узлами кликните сначала на один узел, затем на второй. Длина ребра будет автоматически подсчитана.

- Выбор модификации: Для использования модификации «Блуждающая колония» установите галочку в флажок «Использовать модификацию».

3. **Запуск алгоритма**: Нажмите кнопку «Рассчитать» для выполнения алгоритма. При этом будет вычислен и отображён кратчайший гамильтонов цикл. Программа отобразит последовательность вершин кратчайшего гамильтонова цикла и построенный маршрут с выделенными ребрами.
4. **Дополнительные элементы интерфейса**: Для отмены последнего действия нажмите кнопку «Отмена». Для сброса графа нажмите кнопку «Очистить».

Рекомендации программиста

Актуальность версии Python: Используйте обновленную версию Python, tkinter [1], numpy [2], time [3], math [4], json [5] и random [6]. Уделяйте внимание четкому именованию переменных и функций. Тщательно тестируйте на различных графах, чтобы удостовериться в корректной работе алгоритма и визуализации. Убедитесь, что интерфейс программы в Tkinter легко понимается пользователями. Разделите входные данные, управление алгоритмом и отображение результатов по разным секциям.

Исходный код программы

https://github.com/romplle/spbu-algorithms-and-data-structures/

Контрольный пример

1. Запуск программы и ввод параметров

   Для запуска программы откройте main.py. Программа откроет графический интерфейс для настройки и запуска алгоритма имитации отжига, предназначенного для решения задачи коммивояжёра (TSP) (Рис. 2).

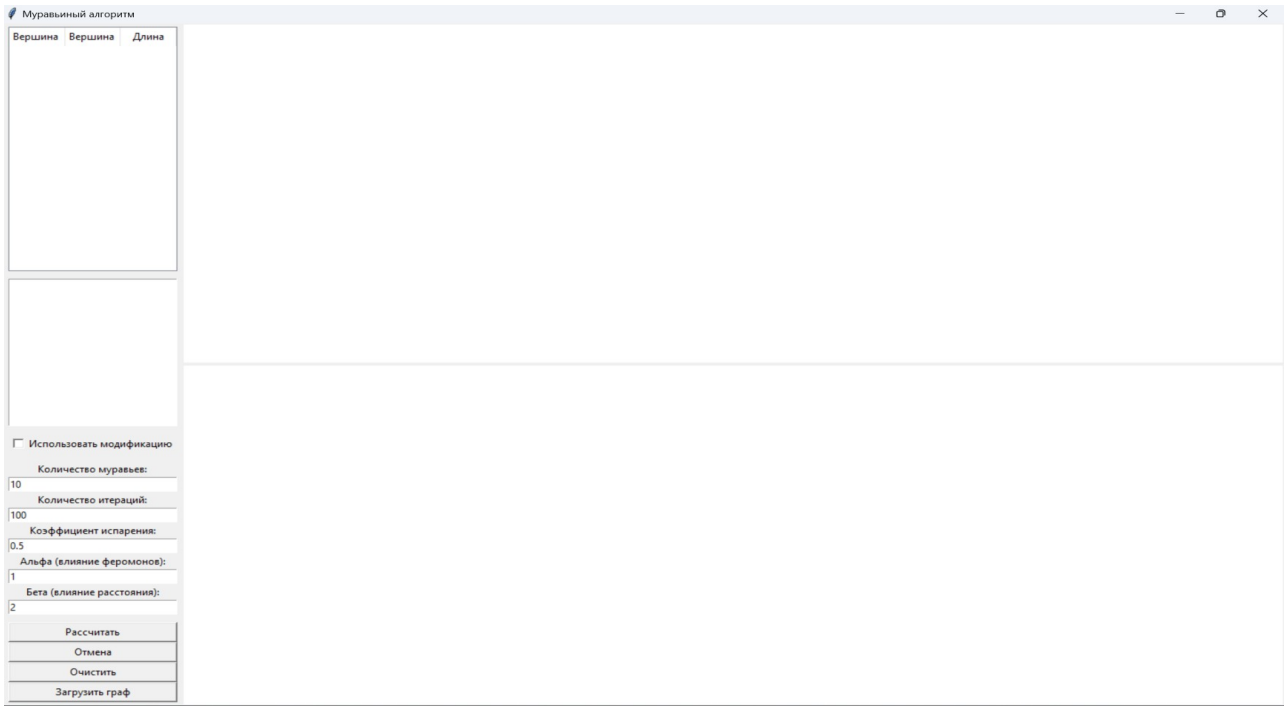

Рис. 2. Начальное окно программы

2. Ввод исходный данных

   После запуска программы можно добавить вершины и соединить их ребрами вручную, а можно нажать кнопку «Загрузить граф» и выбрать JSON-файл для импортирования готового графа. Вы также можете выбрать режим работы (стандартный или модифицированный) (Рис. 3).

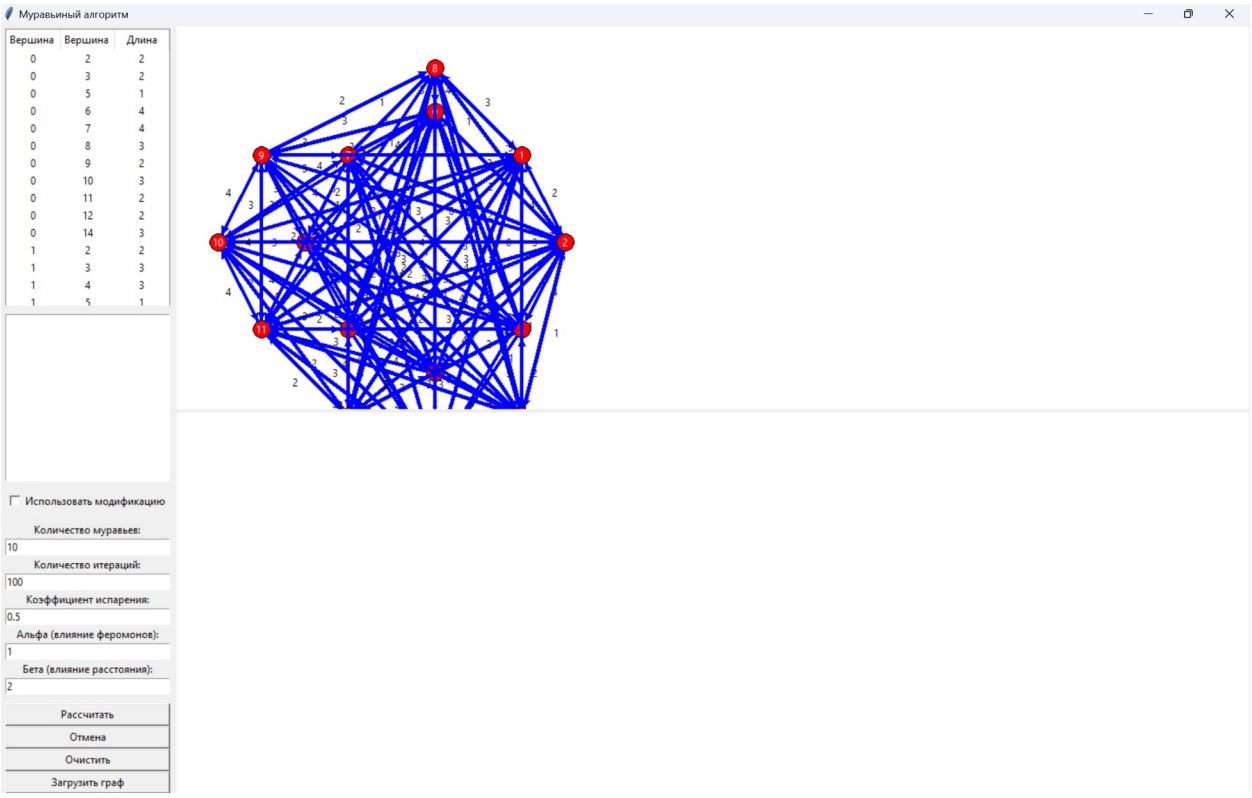

Рис. 3. Пример исходных данных программы

3. Запуск алгоритма

   После задания графа нажмите кнопку «Рассчитать», чтобы запустить алгоритм. Программа вычислит кратчайший гамильтонов цикл методом имитации отжига, обновляя интерфейс для отображения результатов (Рис. 4).

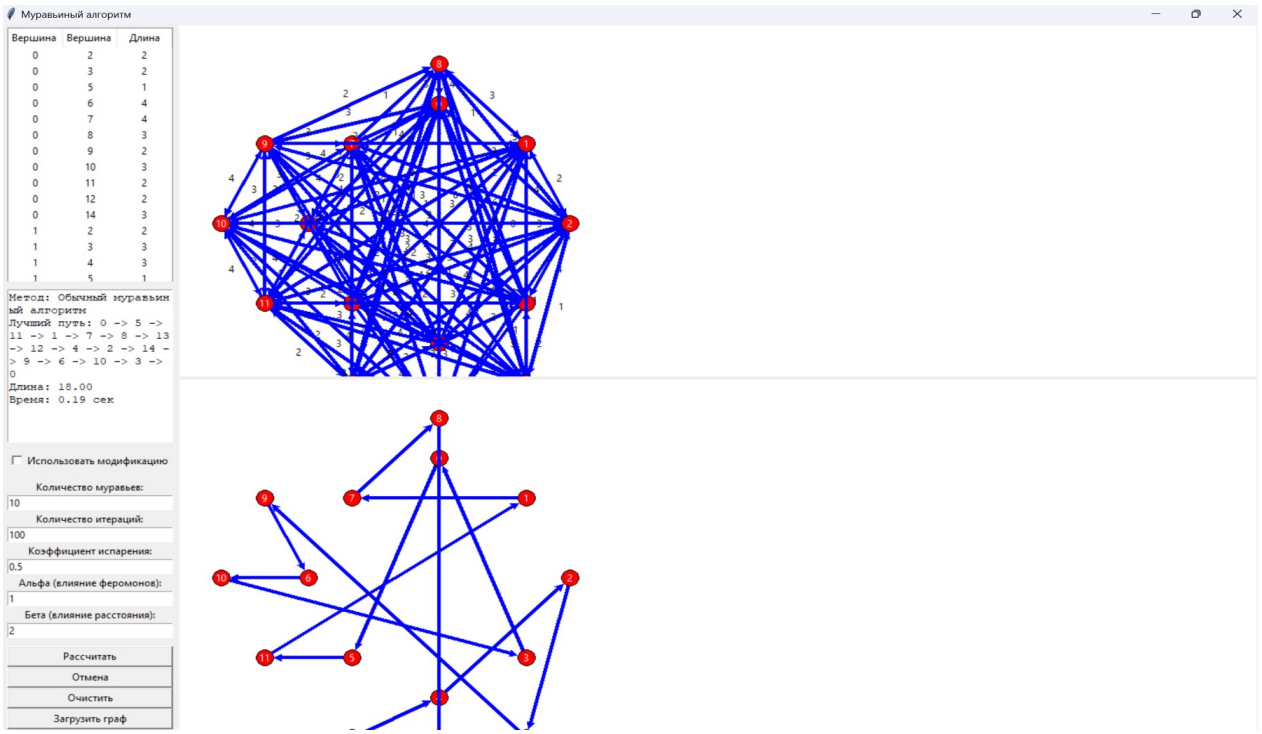

Рис. 4. Пример результатов программы

Анализ

В качестве модификации классического муравьиного алгоритма был применён метод блуждающей колонии. Для сравнения также был протестирован алгоритм ближайшего соседа и алгоритм имитации отжига. Основной целью являлось сравнение времени выполнения и качества найденного маршрута (длины пути) в зависимости от количества вершин и использования модификации.

Начальные параметры: количество муравьев — 10, количество итераций — 100, коэффициент испарения — 0.5, альфа (влияние феромонов) — 1, бета (влияние расстояния) - 2.

Таблица 2: тестирование муравьиный алгоритм без модификации

Кол-во вершин	Модификация	Время	Длина пути
6	Без модификации	0.03 сек	12
6	Без модификации	0.04 сек	12
6	Без модификации	0.04 сек	12
15	Без модификации	0.29 сек	17
15	Без модификации	0.26 сек	18
15	Без модификации	0.26 сек	18
25	Без модификации	0.67 сек	25
25	Без модификации	0.70 сек	25
25	Без модификации	0.67 сек	25

Таблица 3: тестирование муравьиный алгоритм с модификацией

Кол-во вершин	Модификация	Время	Длина пути
6	С модификацией	0.05 сек	12
6	С модификацией	0.05 сек	12
6	С модификацией	0.05 сек	12
15	С модификацией	0.24 сек	19
15	С модификацией	0.25 сек	18
15	С модификацией	0.27 сек	21
25	С модификацией	0.70 сек	25
25	С модификацией	0.69 сек	25
25	С модификацией	0.67 сек	26

Таблица 4: тестирование алгоритма имитации отжига без модификации

Кол-во вершин	Модификация	Время	Длина пути
6	Без модификации	0.07 сек	12
6	Без модификации	0.07 сек	13
6	Без модификации	0.07 сек	16
15	Без модификации	0.09 сек	28
15	Без модификации	0.09 сек	25

15	Без модификации	0.09 сек	24
25	Без модификации	0.12 сек	29
25	Без модификации	0.13 сек	32
25	Без модификации	0.12 сек	31

Таблица 5: тестирование алгоритма имитации отжига с модификацией

Кол-во вершин	Модификация	Время	Длина пути
6	С модификацией	0.07 сек	12
6	С модификацией	0.07 сек	12
6	С модификацией	0.07 сек	15
15	С модификацией	0.09 сек	21
15	С модификацией	0.09 сек	24
15	С модификацией	0.09 сек	22
25	С модификацией	0.12 сек	27
25	С модификацией	0.12 сек	27
25	С модификацией	0.11 сек	26

Таблица 6: тестирование алгоритма ближайшего соседа

Кол-во вершин	Модификация	Время	Длина пути
6	Без модификации	0.0001 сек	12
6	С модификацией	0.0002 сек	12
15	Без модификации	0.0022 сек	18
15	С модификацией	0.0258 сек	17
25	Без модификации	0.0090 сек	31
25	С модификацией	0.1414 сек	25

Общие наблюдения

Применение модификации не дало существенного улучшения результатов по сравнению с классическим вариантом. В некоторых случаях (например, при 15 вершинах) длина пути даже оказалась хуже, чем без модификации. Это может быть связано с тем, что блуждающая колония снижает качество решений на малом количестве итераций.

Обычный муравьиный алгоритм стабильно выдаёт лучшие результаты по длине пути, чем оба варианта имитации отжига, особенно на графах с 15 и 25 вершинами. Однако цена за это — значительно большее время выполнения: от 0.26 до 0.70 секунд против 0.09–0.13 секунд у отжига. Таким образом, муравьиный алгоритм более точен, но менее эффективен по времени.

Этот метод ожидаемо оказался самым быстрым. Однако, он уступает по качеству найденных маршрутов: на 25 вершинах ближайший сосед дал путь

длиной 31 (без модификации), тогда как отжиг и муравьиный алгоритм находили решения длиной 25–27.

Интересно, что добавление модификации к алгоритму ближайшего соседа значительно увеличило время выполнения (в 10–15 раз при 25 вершинах), но улучшило качество маршрута — путь сократился с 31 до 25. Это может говорить о том, что модификация превращает метод из чисто жадного в гибридный, что в отдельных случаях даёт выигрыш.

Вывод

В рамках работы был реализован муравьиный алгоритм для решения задачи коммивояжёра, улучшенный модификацией «Блуждающая колония». Также был разработан удобный графический интерфейс. В ходе анализа результатов были сделаны следующие выводы: если важна точность маршрута, особенно на больших графах, лучше использовать муравьиный алгоритм; если критична производительность, но допустим некоторый проигрыш в качестве, подойдёт имитация отжига, особенно с модификацией; а алгоритм ближайшего соседа может быть полезен как быстрая эвристика, особенно в задачах, где скорость важнее оптимальности.

Источники

1. tkinter — Python interface to Tcl/Tk // URL: https://docs.python.org/3/library/tkinter.html (дата обращения: 26.03.2025).
2. NumPy documentation // URL: https://numpy.org/doc/stable/index.html (дата обращения: 26.03.2025).
3. time — Time access and conversions // URL: https://docs.python.org/3/library/time.html (дата обращения: 27.03.2025).
4. math — Mathematical functions // URL: https://docs.python.org/3/library/math.html (дата обращения: 27.03.2025).
5. json — JSON encoder and decoder // URL: https://docs.python.org/3/library/json.html (дата обращения: 27.03.2025).
6. random — Generate pseudo-random numbers // URL: https://docs.python.org/3/library/random.html (дата обращения: 10.04.2025).