Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 6
по дисциплине «Анализ алгоритмов»

Тема Методы решения задачи коммивояжёра

Студент Ильченко Е.А.

Группа ИУ7-54Б

Преподаватель Волкова Л.Л.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: сравнительный анализ метода полного перебора и метода на основе муравьиного алгоритма для поиска кратчайшего незамкнутого пути в неориентированном взвешенном графе.

Для достижения поставленной цели необходимо решить следующие задачи:

— описать метод полного перебора и метод на основе муравьиного алгоритма;

— реализовать два метода;

— выполнить оценку трудоёмкости реализованных алгоритмов по разработанным схемам;

— выполнить параметризацию алгоритма поиска минимального пути на основе муравьиного алгоритма;

— провести сравнительный анализ двух рассмотренных методов решения задачи коммивояжёра;

— по итогам исследования дать рекомендации о значениях параметров.

1 Аналитическая часть

1.1 Постановка задачи коммивояжёра

Пусть задан неориентированный взвешенный граф $G = (V, E)$, где $V = \{v_1, v_2, \ldots, v_n\}$ — множество вершин, соответствующих городам, а $E \subseteq V \times V$ — множество рёбер, соответствующих возможным путям между городами. Каждому ребру $(i, j) \in E$ сопоставлен вес $w_{ij} \in \mathbb{R}^+$, определяющий время перемещения между вершинами i и j. Для неориентированного графа матрица весов $W = \|w_{ij}\|_{n \times n}$ является симметричной, то есть $w_{ij} = w_{ji}$.

Классическая задача коммивояжёра формулируется как поиск гамильтонова цикла минимальной длины, то есть замкнутого маршрута, проходящего через каждую вершину графа ровно один раз и возвращающегося в исходную вершину [1].

В настоящей работе рассматривается модификация задачи: требуется найти **гамильтонов путь** минимальной длины, то есть **незамкнутый маршрут**, проходящий через каждую вершину графа ровно один раз без возврата в начальный город.

Пусть $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ — перестановка номеров вершин графа, определяющая порядок их посещения. Тогда длина (время) незамкнутого маршрута определяется как

$$L(\pi) = \sum_{i=1}^{n-1} w_{\pi_i \pi_{i+1}}. \tag{1.1}$$

Необходимо найти такую перестановку π^*, что

$$L(\pi^*) = \min_{\pi} L(\pi). \tag{1.2}$$

1.2 Метод полного перебора

Алгоритм полного перебора (brute force) представляет собой точный метод решения задачи коммивояжёра, основанный на анализе всех возможных маршрутов [2].

Основные этапы алгоритма:

1) сгенерировать все возможные перестановки вершин графа (размер пространства поиска составляет $(n-1)!$ маршрутов);

2) для каждой перестановки π вычислить длину незамкнутого маршрута $L(\pi)$;

3) выбрать перестановку с минимальным значением функции $L(\pi)$.

В отличие от классической постановки с циклом, в данной работе длина маршрута вычисляется только по переходам между последовательными вершинами пути и не включает возврат из последней вершины в первую.

Алгоритм полного перебора гарантированно находит оптимальное решение, однако имеет факториальную трудоёмкость $O(n! \cdot n)$ и потому пригоден только для графов небольшой размерности.

1.3 Муравьиный алгоритм

Муравьиный алгоритм относится к классу стохастических оптимизационных методов, инспирированных поведением муравьиных колоний при поиске кратчайших путей между гнездом и источниками пищи [3]. Муравьи взаимодействуют опосредованно, через феромонные следы, усиливая успешные маршруты и постепенно «забывая» неудачные.

В данной работе используется модификация муравьиного алгоритма для поиска гамильтонова пути без использования элитных муравьёв.

1.3.1 Феромонная модель и эвристика

Каждому ребру (i, j) сопоставлен уровень феромона τ_{ij}, отражающий накопленный «опыт» успешного использования этого ребра муравьями.

Эвристическая информация задаётся как *видимость* ребра

$$\eta_{ij} = \frac{1}{w_{ij}}, \tag{1.3}$$

которая тем больше, чем короче ребро.

1.3.2 Правило выбора следующей вершины

При построении маршрута каждый муравей находится в некоторой текущей вершине i и имеет множество допустимых для перехода вершин $J_k(i)$, не содержащих уже посещённые вершины. Вероятность перехода муравья k из вершины i в вершину j определяется соотношением [3]:

$$P_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum\limits_{l \in J_k(i)} [\tau_{il}]^\alpha \cdot [\eta_{il}]^\beta}, \tag{1.4}$$

где:

— $\alpha \geqslant 0$ — параметр влияния феромона;

— $\beta \geqslant 0$ — параметр влияния эвристики;

— $J_k(i)$ — множество непосещённых вершин, допустимых для перехода.

1.3.3 Правило обновления феромонов

После того как все муравьи построили свои маршруты, выполняется обновление уровней феромонов на рёбрах. Пусть L_k — длина маршрута k-го муравья, а m — число муравьёв. Тогда суммарное количество феромона, добавляемое на ребро (i, j), вычисляется как

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \begin{cases} \dfrac{Q}{L_k}, & \text{если ребро } (i,j) \text{ входит в маршрут муравья } k, \\ 0, & \text{иначе,} \end{cases} \tag{1.5}$$

где Q — параметр, пропорциональный среднему весу рёбер графа.

Обновление феромона задаётся правилом

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \Delta\tau_{ij}, \tag{1.6}$$

где $\rho \in (0, 1]$ — коэффициент испарения феромона.

1.4 Вывод

В данном разделе была сформулирована задача коммивояжёра в постановке незамкнутого маршрута на неориентированном графе и рассмотрены теоретические основы двух подходов к её решению. Алгоритм полного перебора обеспечивает точное нахождение минимального по длине гамильтонова пути, но обладает факториальной трудоёмкостью и применим только для графов малой размерности. Муравьиный алгоритм использует феромонную модель и эвристическую видимость рёбер для вероятностного построения маршрутов и позволяет получать приближённые решения при существенно меньшей трудоёмкости.

2 Конструкторская часть

2.1 Требования к реализации

Разрабатываемое программное обеспечение представляет собой консольное приложение на языке Python, работающее в нескольких режимах:

— **режим одиночного запуска** — пользователь выбирает источник данных, задаёт параметры муравьиного алгоритма, после чего программа выводит найденный маршрут и его длину для алгоритма полного перебора и/или муравьиного алгоритма;

— **режим замера времени** — по заданному диапазону размеров графа выполняется серия запусков обоих алгоритмов, измеряется время работы и формируется таблица для последующего анализа и построения графиков;

— **режим параметризации** — проводится исследование влияния параметров муравьиного алгоритма.

2.2 Описание алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора, на рисунках 2.2–2.3 — схема муравьиного алгоритма.

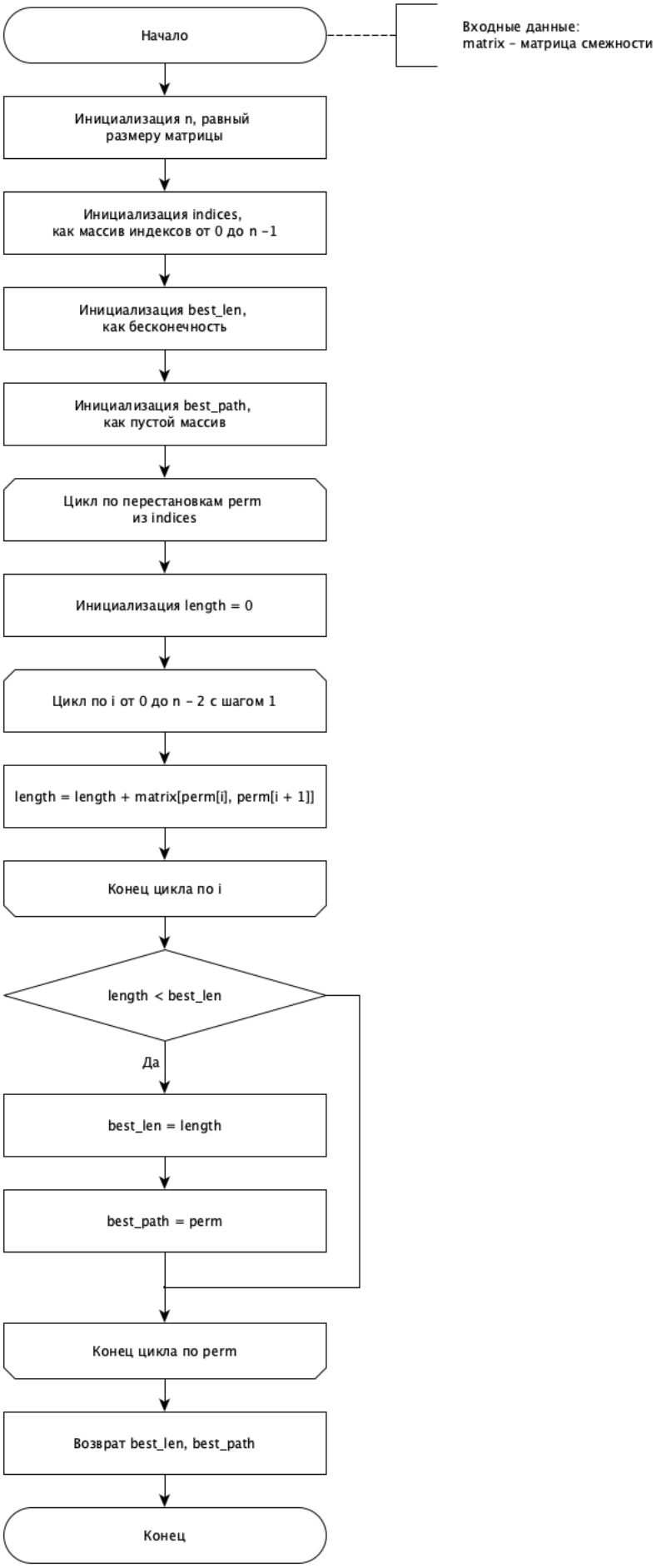

Рисунок 2.1 — Описание алгоритма полного перебора

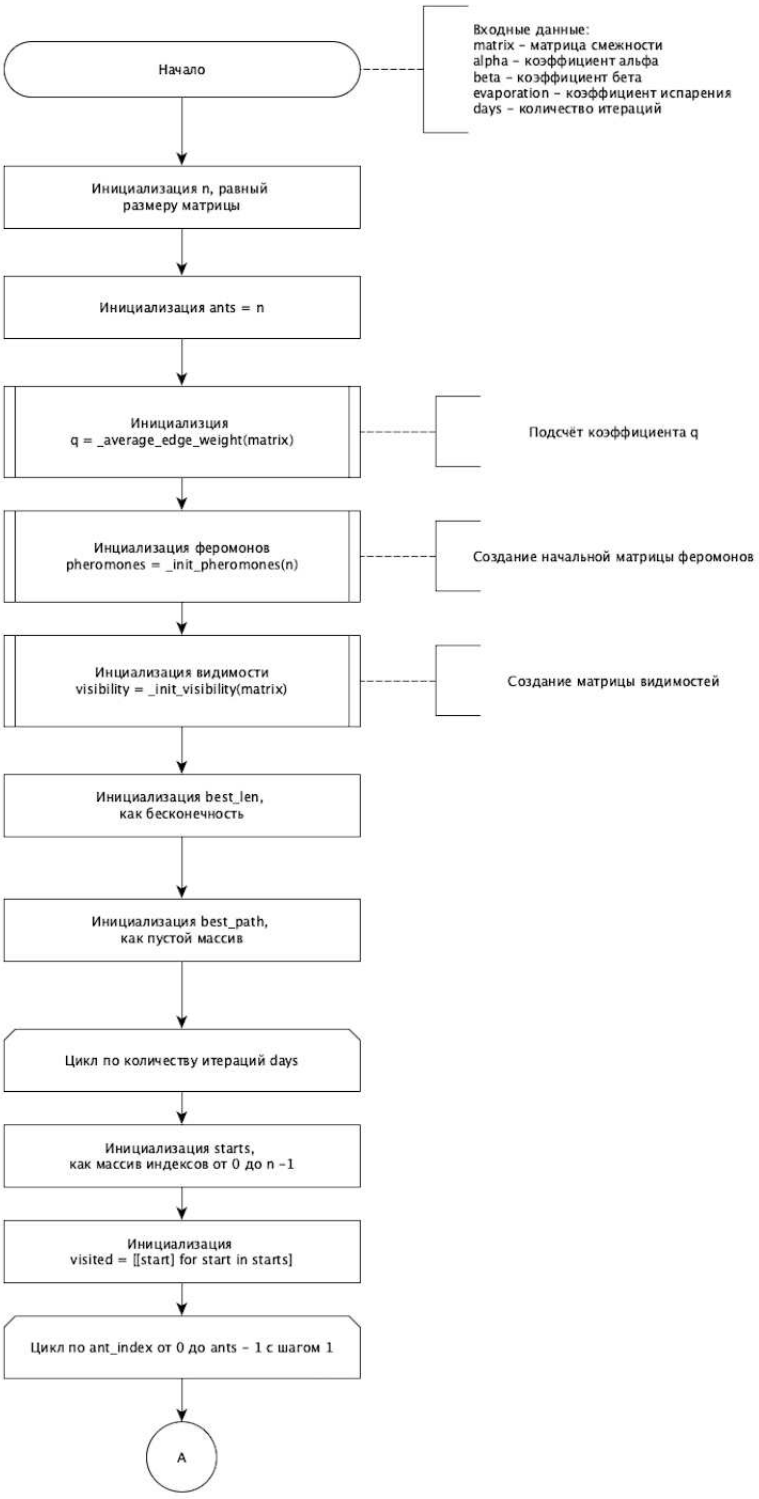

Рисунок 2.2 — Муравьиный алгоритм, часть 1

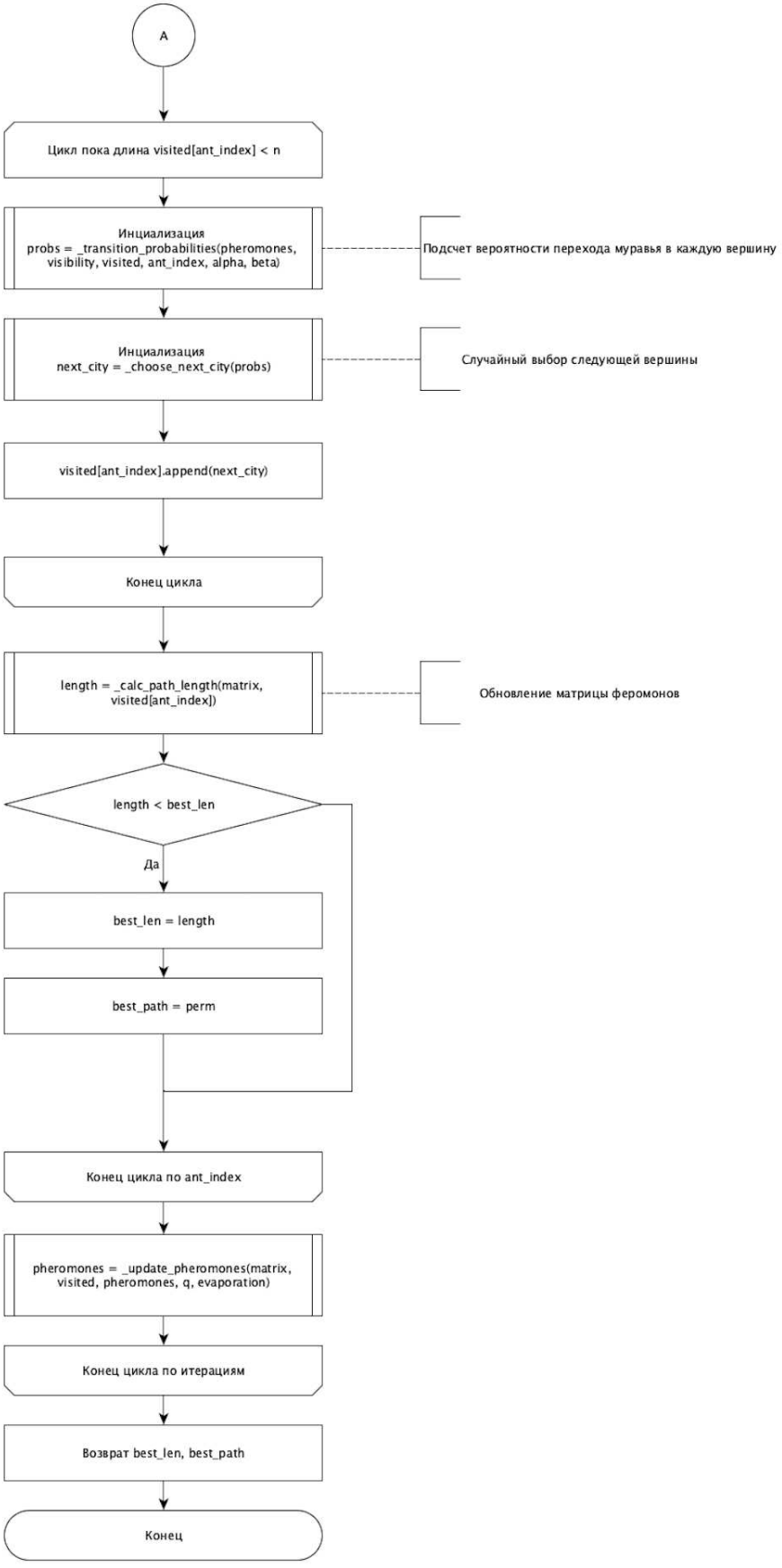

Рисунок 2.3 — Муравьиный алгоритм, часть 2

Вывод

В данном разделе были представлены схемы для алгоритма полного перебора и муравьиного алгоритма поиска кратчайших путей в графе.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов был выбран язык Python 3, так как он соответствует требованиям лабораторной работы с использованием следующих библиотек:

Замеры времени выполнения проводились с использованием функции `process_time` модуля `time`, измеряющей процессорное время текущего процесса. Разработка проводилась в интегрированной среде разработки `PyCharm`.

3.2 Реализация алгоритмов

В листинге 3.1 приведён фрагмент кода алгоритма полного перебора, а реализация муравьиного алгоритма вынесена в приложение (листинг 4.1).

```
def brute_force_path(matrix: np.ndarray) -> tuple[float, list[int]]:
    n = matrix.shape[0]
    indices = list(range(n))

    best_len = float("inf")
    best_path: list[int] = []

    for perm in it.permutations(indices):
        length = 0.0
        for i in range(n - 1):
            length += float(matrix[perm[i], perm[i + 1]])

        if length < best_len:
            best_len = length
            best_path = list(perm)

    return best_len, best_path
```

Листинг 3.1 — Алгоритм полного перебора

3.3 Подготовка входных данных

Для построения матрицы расстояний между странами Африки задаётся фиксированный набор из 11 стран, формируется квадратная матрица 11×11 с нулями на диагонали. Для каждой пары стран по их координатам (широта, долгота) считается расстояние и умножается на коэффициент местности: для двух пустынных государств расстояние увеличивается в $1{,}5$ раза, для маршрутов с участием Мадагаскара уменьшается до $0{,}7$ от сухопутного аналога. В результате получаются значения от 950 до 5571: например, «Алжир — Ливия» даёт 2326,

«Алжир — Египет» — 4309, «Мали — Египет» — 5571, «Нигер — Нигерия» — 950, «Нигер — Чад» — 1740. Для связей с Мадагаскаром значения заметно меньше: «Кения — Мадагаскар» даёт 1612, «Эфиопия — Мадагаскар» — 2227. Округлённая до целых матрица сохраняется в текстовый файл и далее используется как входные данные для функциональных тестов и параметризации.

3.4 Функциональные тесты

Для проверки корректности работы программной реализации были проведены функциональные тесты на матрицах расстояний между странами Африки, построенных по географическим координатам и скорректированных с учётом типа местности.

В таблице 3.1 приведены примеры тестов.

Таблица 3.1 — Функциональные тесты на матрицах карты Африки

Матрица расстояний	Страны	Результат
$\begin{pmatrix} 0 & 2326 & 4309 & 1949 & 1999 & 3369 \\ 2326 & 0 & 2025 & 3587 & 2028 & 1830 \\ 4309 & 2025 & 0 & 5571 & 3822 & 2665 \\ 1949 & 3587 & 5571 & 0 & 1920 & 3649 \\ 1999 & 2028 & 3822 & 1920 & 0 & 1740 \\ 3369 & 1830 & 2665 & 3649 & 1740 & 0 \end{pmatrix}$	1 — Алжир 2 — Ливия 3 — Египет 4 — Мали 5 — Нигер 6 — Чад	Оптимальный незамкнутый маршрут: $1 \to 4 \to 5 \to 6 \to 2 \to 3$ Минимальная длина: 9464
$\begin{pmatrix} 0 & 1740 & 3646 & 950 & 3806 & 3624 \\ 1740 & 0 & 1907 & 1302 & 2720 & 2464 \\ 3646 & 1907 & 0 & 2388 & 1665 & 1195 \\ 950 & 1302 & 2388 & 0 & 3391 & 3492 \\ 3806 & 2720 & 1665 & 3391 & 0 & 1059 \\ 3624 & 2464 & 1195 & 3492 & 1059 & 0 \end{pmatrix}$	1 — Нигер 2 — Чад 3 — Судан 4 — Нигерия 5 — Кения 6 — Эфиопия	Оптимальный незамкнутый маршрут: $1 \to 4 \to 2 \to 3 \to 6 \to 5$ Минимальная длина: 6413
Матрица 11x11 (см. Приложение А.3)	1 — Алжир 2 — Ливия 3 — Египет 4 — Мали 5 — Нигер 6 — Чад 7 — Судан 8 — Нигерия 9 — Кения 10 — Эфиопия 11 — Мадагаскар	Оптимальный незамкнутый маршрут: $1 \to 4 \to 5 \to 8 \to 6 \to 2 \to 3 \to$ $7 \to 10 \to 9 \to 11$ Минимальная длина: 16172

Во всех трёх тестах результаты муравьиного алгоритма при подобранных параметрах совпали или оказались очень близки к результатам полного перебора, что подтверждает корректность реализации.

3.5 Оценка трудоёмкости алгоритмов

3.5.1 Алгоритм полного перебора

Трудоёмкость алгоритма полного перебора рассчитывается следующим образом:

1) инициализация переменных: $f_{init} = 3$;
2) генерация всех перестановок ($(n-1)!$ итераций):

$$f_{permutations} = (n-1)! \cdot f_{iteration}$$

3) обработка одной перестановки:

$$f_{iteration} = f_{length} + f_{compare} + f_{update}$$

4) вычисление длины пути: $f_{length} = n - 1$;
5) сравнение с минимальной длиной: $f_{compare} = 1$;
6) обновление лучшего пути: $f_{update} = n$.

Для каждой перестановки:

$$f_{iteration} = (n-1) + 1 + n = 2n$$

Общая трудоёмкость:

$$f_{brute\_force} = 3 + (n-1)! \cdot (2n) \approx O(n! \cdot n)$$

3.5.2 Муравьиный алгоритм

Трудоёмкость муравьиного алгоритма рассчитывается следующим образом:

1) инициализация матрицы феромонов: $f_{init\_pheromone} = n^2$;
2) инициализация матрицы видимости: $f_{init\_visibility} = n^2$;
3) основной цикл (t_{max} итераций):

$$f_{main\_loop} = t_{max} \cdot (f_{ants} + f_{update})$$

4) цикл по муравьям (n муравьёв):

$$f_{ants} = n \cdot f_{ant\_path}$$

5) построение пути одного муравья ($n - 1$ шагов):

$$f_{ant\_path} = (n-1) \cdot (f_{probabilities} + f_{next}) + f_{calc\_length}$$

6) вычисление вероятностей перехода: $f_{probabilities} = 2n$;

7) выбор следующей вершины: $f_{next} = n$;

8) расчёт длины пути: $f_{calc\_length} = n$;

9) обновление феромонов: $f_{update} = n^2 \cdot n = n^3$.

Для одного муравья:

$$f_{ant\_path} = (n-1) \cdot (2n+n) + n = (n-1) \cdot 3n + n = 3n^2 - 2n$$

Для всех муравьёв на одной итерации:

$$f_{ants} = n \cdot (3n^2 - 2n) = 3n^3 - 2n^2$$

Общая трудоёмкость одной итерации:

$$f_{day} = (3n^3 - 2n^2) + n^3 = 4n^3 - 2n^2$$

Общая трудоёмкость алгоритма:

$$f_{ant} = 2n^2 + t_{max} \cdot (4n^3 - 2n^2) \approx O(t_{max} \cdot n^3)$$

Вывод

В технологической части были выбраны средства реализации, приведены фрагменты кода реализованных алгоритмов, проведены функциональные тесты и выполнена оценка трудоёмкости методов полного перебора и муравьиного алгоритма. Алгоритм полного перебора обладает факториальной сложностью и подходит лишь для малых графов, тогда как муравьиный алгоритм имеет полиномиальную сложность $O(t_{\max} \cdot n^3)$ и может применяться к задачам существенно большей размерности.

4 Исследовательская часть

4.1 Характеристики ЭВМ

Замеры проводились на устройстве со следующими характеристиками:

— процессор: Apple M4 Pro;

— количество логических ядер: 12;

— количество ядер: 12;

— оперативная память: 24 Гб;

— операционная система: macOS Sequoia 15.6.1.

Замеры времени проводились, когда ноутбук был загружен только системными приложениями.

4.2 Сравнение времени работы

Для оценки временной эффективности алгоритмов полного перебора и муравьиного алгоритма были проведены замеры на случайно сгенерированных полносвязных неориентированных графах с целочисленными весами рёбер. Для каждого размера графа выполнялось по нескольку запусков и усреднялось процессорное время.

Результаты приведены в таблице 4.1.

Таблица 4.1 — Сравнение алгоритмов по времени выполнения

Размер	Время полного перебора, с	Время муравьиного алгоритма, с
2	0,000010	0,000316
3	0,000003	0,001013
4	0,000010	0,002391
5	0,000043	0,004671
6	0,000284	0,008640
7	0,002464	0,013845
8	0,020792	0,021348
9	0,209074	0,031641
10	2,313401	0,045135

На основе полученных данных был построен график зависимости времени работы алгоритмов от количества вершин (рисунок 4.1).

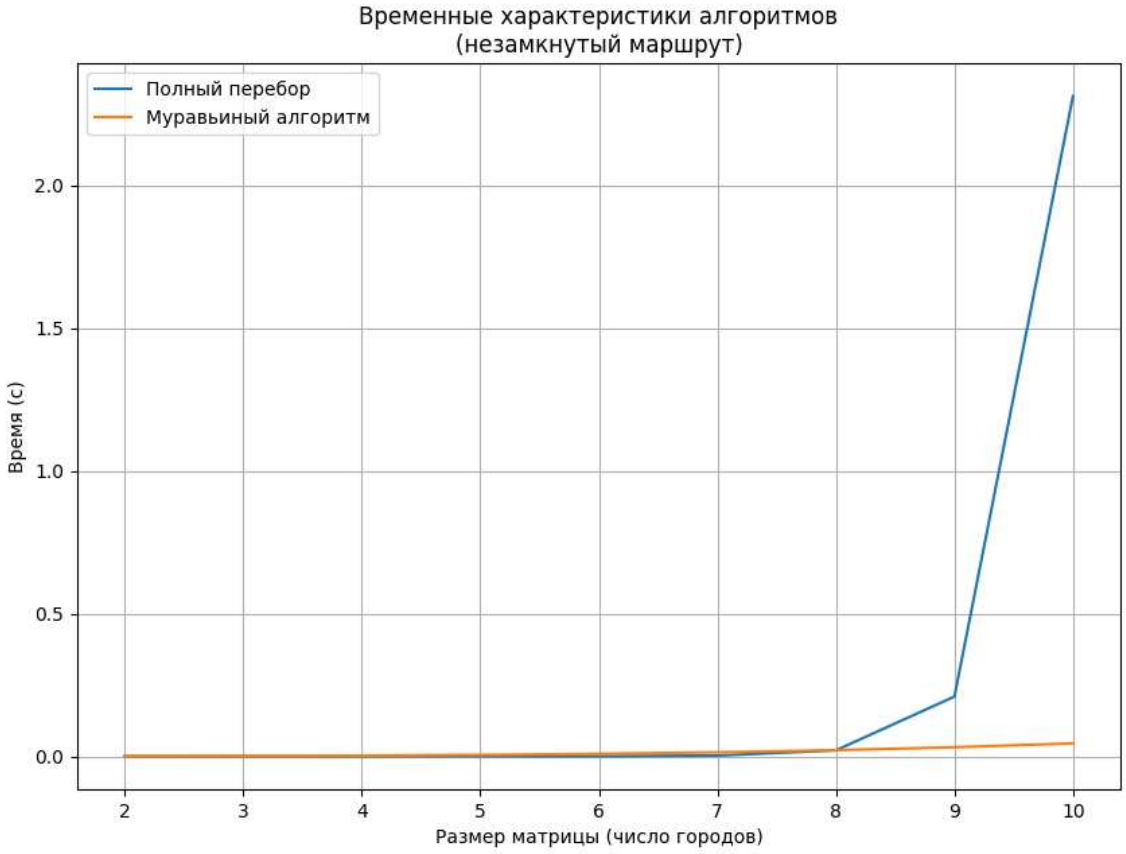

Рисунок 4.1 — Сравнение алгоритмов по времени выполнения

4.3 Параметризация муравьиного алгоритма

Была выполнена параметризация муравьиного алгоритма по трём основным параметрам (коэффициенты влияния феромона и эвристики, коэффициент испарения феромона), а также по числу итераций. Замеры проводились на фиксированном полносвязном неориентированном графе малого размера; для каждой комбинации параметров вычислялось отклонение длины маршрута, найденного муравьиным алгоритмом, от оптимального значения, полученного методом полного перебора. Результаты параметризации приведены в таблице А.1 Приложения А.

4.4 Вывод

При увеличении числа вершин трудоёмкость алгоритма полного перебора быстро возрастает и делает практическое применение невозможным. Муравьиный алгоритм на тех же тестовых данных показывает значительно более плавный рост времени выполнения и, начиная примерно с $n \approx 8$ вершин, становится сопоставим по скорости или быстрее полного перебора, а при дальнейшем увеличении размера графа его преимущество по времени только усиливается.

По полученной таблице параметризации наилучшие результаты дают наборы с $\alpha \in [0{,}1; 0{,}5]$, $\beta = 1 - \alpha$, $\rho \in [0{,}1; 0{,}5]$ и $t_{max} \in [300; 500]$, при которых во многих случаях удаётся получить маршрут с нулевым отклонением.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы поставленная цель была достигнута, а также были решены следующие задачи:

— сформулирована задача коммивояжёра для полносвязного неориентированного графа с требованием нахождения гамильтонова пути (незамкнутого маршрута);

— разработан и реализован алгоритм полного перебора для точного решения задачи;

— разработан и реализован муравьиный алгоритм без элитных муравьёв;

— проведена параметризация муравьиного алгоритма по трём основным параметрам (α, β, ρ) и числу итераций;

— выполнена оценка трудоёмкости разработанных алгоритмов и сравнительный анализ их временных характеристик;

— сформулированы рекомендации по настройке муравьиного алгоритма и выбору метода решения в зависимости от размера задачи.

По результатам исследования установлено:

— алгоритм полного перебора обладает факториальной трудоёмкостью $O(n! \cdot n)$ и обеспечивает точное решение задачи, но практически применим только для графов малой размерности;

— муравьиный алгоритм имеет полиномиальную сложность порядка $O(t_{max} \cdot n^3)$ и позволяет находить близкие к оптимальным решения для графов существенно большей размерности;

— результаты параметризации показали, что наиболее устойчивую работу муравьиного алгоритма обеспечивают параметры $\alpha \in [0{,}1; 0{,}5]$, $\beta = 1 - \alpha$, $\rho \in [0{,}1; 0{,}5]$ и $t_{max} \in [300; 500]$.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аннабаева Н. Р. РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА // Наука и мировоззрение. 2025. Т. 1, № 59. С. 43-48.

2. Вечур А. В., Суяргулова Е. Б. Исследование закономерности расположения гамильтоновых циклов в графе // Автоматизированные системы управления и приборы автоматики. 2006. № 137. С. 41-48.

3. Телегин В. А. МУРАВЬИНЫЕ АЛГОРИТМЫ ДЛЯ РЕШЕНИЯ ЗАДАЧИ КОММИВОЯЖЁРА // Международный научно-исследовательский журнал. 2024. № 7 (145). с. 1.

ПРИЛОЖЕНИЕ А

Для каждой комбинации параметров в строке файла содержатся значения:

— коэффициент влияния феромона α;

— коэффициент влияния эвристики β;

— коэффициент испарения феромона ρ;

— число итераций t_{max};

— оптимальная длина маршрута L^*, найденная алгоритмом полного перебора;

— отклонение $L - L^*$ длины маршрута, найденного муравьиным алгоритмом, от оптимального значения.

Таблица А.1 — Результаты параметризации муравьиного алгоритма

α	β	ρ	t_{max}	L^*	$L - L^*$
0.1	0.9	0.1	100	95	3
0.1	0.9	0.1	300	95	0
0.1	0.9	0.1	500	95	0
0.1	0.9	0.2	100	95	3
0.1	0.9	0.2	300	95	0
0.1	0.9	0.2	500	95	2
0.1	0.9	0.5	100	95	4
0.1	0.9	0.5	300	95	0
0.1	0.9	0.5	500	95	0
0.1	0.9	0.9	100	95	4
0.1	0.9	0.9	300	95	0
0.1	0.9	0.9	500	95	0
0.2	0.8	0.1	100	95	3
0.2	0.8	0.1	300	95	0
0.2	0.8	0.1	500	95	2
0.2	0.8	0.2	100	95	9
0.2	0.8	0.2	300	95	3
0.2	0.8	0.2	500	95	2
0.2	0.8	0.5	100	95	5
0.2	0.8	0.5	300	95	0
0.2	0.8	0.5	500	95	0
0.2	0.8	0.9	100	95	4
0.2	0.8	0.9	300	95	0
0.2	0.8	0.9	500	95	0
0.5	0.5	0.1	100	95	6
0.5	0.5	0.1	300	95	2

Таблица А.2 — Продолжение таблицы А.1

α	β	ρ	t_{max}	L^*	$L - L^*$
0.5	0.5	0.1	500	95	0
0.5	0.5	0.2	100	95	3
0.5	0.5	0.2	300	95	0
0.5	0.5	0.2	500	95	0
0.5	0.5	0.5	100	95	0
0.5	0.5	0.5	300	95	3
0.5	0.5	0.5	500	95	0
0.5	0.5	0.9	100	95	5
0.5	0.5	0.9	300	95	3
0.5	0.5	0.9	500	95	2
0.9	0.1	0.1	100	95	12
0.9	0.1	0.1	300	95	5
0.9	0.1	0.1	500	95	0
0.9	0.1	0.2	100	95	9
0.9	0.1	0.2	300	95	2
0.9	0.1	0.2	500	95	2
0.9	0.1	0.5	100	95	3
0.9	0.1	0.5	300	95	5
0.9	0.1	0.5	500	95	0
0.9	0.1	0.9	100	95	10
0.9	0.1	0.9	300	95	8
0.9	0.1	0.9	500	95	5

Таблица A.3 — Матрица расстояний для полной карты Африки

$$\begin{pmatrix}
0 & 2326 & 4309 & 1949 & 1999 & 3369 & 5109 & 2232 & 4963 & 4570 & 4985 \\
2326 & 0 & 2025 & 3587 & 2028 & 1830 & 3031 & 2120 & 3674 & 3107 & 4164 \\
4309 & 2025 & 0 & 5571 & 3822 & 2665 & 2330 & 3050 & 3080 & 2214 & 3750 \\
1949 & 3587 & 5571 & 0 & 1920 & 3649 & 5555 & 1663 & 4983 & 4891 & 4810 \\
1999 & 2028 & 3822 & 1920 & 0 & 1740 & 3646 & 950 & 3806 & 3624 & 4101 \\
3369 & 1830 & 2665 & 3649 & 1740 & 0 & 1907 & 1302 & 2720 & 2464 & 3427 \\
5109 & 3031 & 2330 & 5555 & 3646 & 1907 & 0 & 2388 & 1665 & 1195 & 2774 \\
2232 & 2120 & 3050 & 1663 & 950 & 1302 & 2388 & 0 & 3391 & 3492 & 3646 \\
4963 & 3674 & 3080 & 4983 & 3806 & 2720 & 1665 & 3391 & 0 & 1059 & 1612 \\
4570 & 3107 & 2214 & 4891 & 3624 & 2464 & 1195 & 3492 & 1059 & 0 & 2227 \\
4985 & 4164 & 3750 & 4810 & 4101 & 3427 & 2774 & 3646 & 1612 & 2227 & 0
\end{pmatrix}$$

```
def ant_algorithm_path(
    matrix: np.ndarray,
    alpha: float,
    beta: float,
    evaporation: float,
    days: int,
) -> tuple[float, list[int]]:
    n = matrix.shape[0]
    ants = n

    q = _average_edge_weight(matrix)
    pheromones = _init_pheromones(n)
    visibility = _init_visibility(matrix)

    best_length = float("inf")
    best_path: list[int] = []

    for _ in range(days):
        starts = list(range(n))
        visited = [[start] for start in starts]

        for ant_index in range(ants):
            while len(visited[ant_index]) < n:
                probs = _transition_probabilities(
                    pheromones, visibility, visited,
                    ant_index, alpha, beta,
                )
                next_city = _choose_next_city(probs)

                if next_city in visited[ant_index]:
                    unvisited = [
                        c for c in range(n)
                        if c not in visited[ant_index]
                    ]
                    if not unvisited:
                        break
                    next_city = unvisited[0]

                visited[ant_index].append(next_city)
```

```
            length = _calc_path_length(matrix, visited[ant_index])
            if length < best_length:
                best_length = length
                best_path = visited[ant_index].copy()

        pheromones = _update_pheromones(
            matrix=matrix,
            paths=visited,
            pheromones=pheromones,
            q=q,
            evaporation=evaporation,
        )

    return best_length, best_path
```

Листинг 4.1 — Муравьиный алгоритм