Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное 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»

Отчет по лабораторной работе № 6 по курсу «Анализ алгоритмов»

Тема Методы решения задачи коммивояжера

Студент Жаворонкова А. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

Целью данной работы является изучение способов решения задачи коммивояжера. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать алгоритм полного перебора и муравьиного алгоритма;

— реализовать указанные алгоритмы;

— провести тестирование по методу черного ящика для реализаций указанных алгоритмов;

— провести сравнительный анализ по времени и по памяти реализаций указанных алгоритмов;

— провести параметризацию муравьиного алгоритма;

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1 Аналитическая часть

В данном разделе будут рассмотрены задача коммивояжера и используемые для ее решения алгоритмы: алгоритм полного перебора и муравьиный алгоритм.

1.1 Задача коммивояжера

Коммивояжер (фр. commis voyageur) — бродячий торговец. Коммивояжеру, чтобы распродать товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Суть задачи коммивояжера — определить наиболее выгодный маршрут объезда [3]. В качестве меры выгодности маршрута может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута. В описываемой задаче рассматривается несколько городов и матрица затрачиваемого времени на дорогу между ними, причем суммарное время пути не должно превышать 80 дней в человеческом измерении.

1.2 Алгоритм полного перебора

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них [5].

Преимуществом данного алгоритма является гарантия нахождения глобального оптимума. Недостатком — трудоемкость $O(n!)$.

1.3 Муравьиный алгоритм

Муравьиный алгоритм — метод решения задачи коммивояжера, в основе которого лежит моделирование поведения колонии муравьев [4]. Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он ощущает во время прохождения, каждый муравей

оставляет феромон на своем пути, чтобы остальные муравьи могли по нему ориентироваться. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Пусть муравей имеет следующие характеристики:

— зрение — способен определить длину ребра;

— память — запоминает пройденный маршрут;

— обоняние — чувствует феромон.

Также введем целевую функцию (1.1) и формулу вычисления вероятности перехода в заданную точку (1.2).

$$\eta_{ij} = 1/D_{ij}, \tag{1.1}$$

где D_{ij} — время, затрачиваемое на дорогу из текущего пункта i до заданного пункта j.

$$P_{kij} = \begin{cases} \frac{\tau_{ij}^a \eta_{ij}^b}{\sum_{q=1}^{m} \tau_{iq}^a \eta_{iq}^b}, \text{вершина не была посещена ранее муравьем k,} \\ 0, \text{иначе} \end{cases} \tag{1.2}$$

где a — коэффициент жадности решения, b — коэффициент стадности решения, τ_{ij} — расстояния от города i до j, η_{ij} — количество феромонов на ребре ij.

После завершения движения всех муравьев, формула обновляется феромон по формуле (1.3):

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}. \tag{1.3}$$

При этом

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \tau_{ij}^k, \tag{1.4}$$

где

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k, \text{ребро посещено k-ым муравьем,} \\ 0, \text{иначе} \end{cases} \tag{1.5}$$

Если феромон на дуге обнулится, то обнулится и вероятность перехода в соответствующий город, что недопустимо. Поэтому в конце модификации феромона требуется выполнить дополнительную проверку: если феромон на дуге упал ниже малой константы, то его требуется откатить до этой константы.

Преимуществом данного алгоритма является трудоемкость:

$$O(t_{max} * \max(m, n))$$

где t_{max} — время жизни колонии, m — количество муравьев, n – размер графа. Недостатком — отсутствие гарантии нахождения глобального оптимума.

1.4 Вывод

В данном разделе были теоретически разобраны алгоритмы полного перебора и муравьиного решения задачи коммивояжера.

В таблице 1.1 представлены результаты сравнения указанных алгоритмов.

Таблица 1.1 – Сравнение алгоритмов

Критерий	Алгоритм полного перебора	Муравьиный алгоритм
Гарантия нахождения глобального оптимума	Да	Нет
Трудоемкость	$O(n!)$	$O(t_{max} * \max(m, n))$

К разрабатываемой программе предъявляются следующие требования:

1. Программа должна предоставлять функциональность рассчета минимального времени пути, а также самого пути;
2. Реализуемое ПО будет работать в двух режимах — пользовательском, в котором можно выбрать алгоритм и вывести для него результат, а также экспериментальном режиме, в котором можно произвести сравнение

реализаций алгоритмов по времени работы на различных входных данных;

3. В первом режиме в качестве входных данных в программу будет подаваться файл с матрицей времен, затрачиваемых на путь из одного города в другой, также реализовано меню для вызова алгоритмов и замеров времени. Программа должна корректно обрабатывать случай ввода несуществующего имени файла;

4. Во втором режиме будет происходить измерение процессорного времени работы программы, будут построены зависимости времени работы от размера матрицы. Матрица будет сгенерирована автоматически для заданного размера матрицы.

2 Конструкторская часть

В этом разделе будут представлены схемы алгоритмов полного перебора и муравьиного.

2.1 Разработка алгоритмов

На рисунках 2.1 — 2.2 представлены схемы алгоритмов полного перебора и муравьиного.

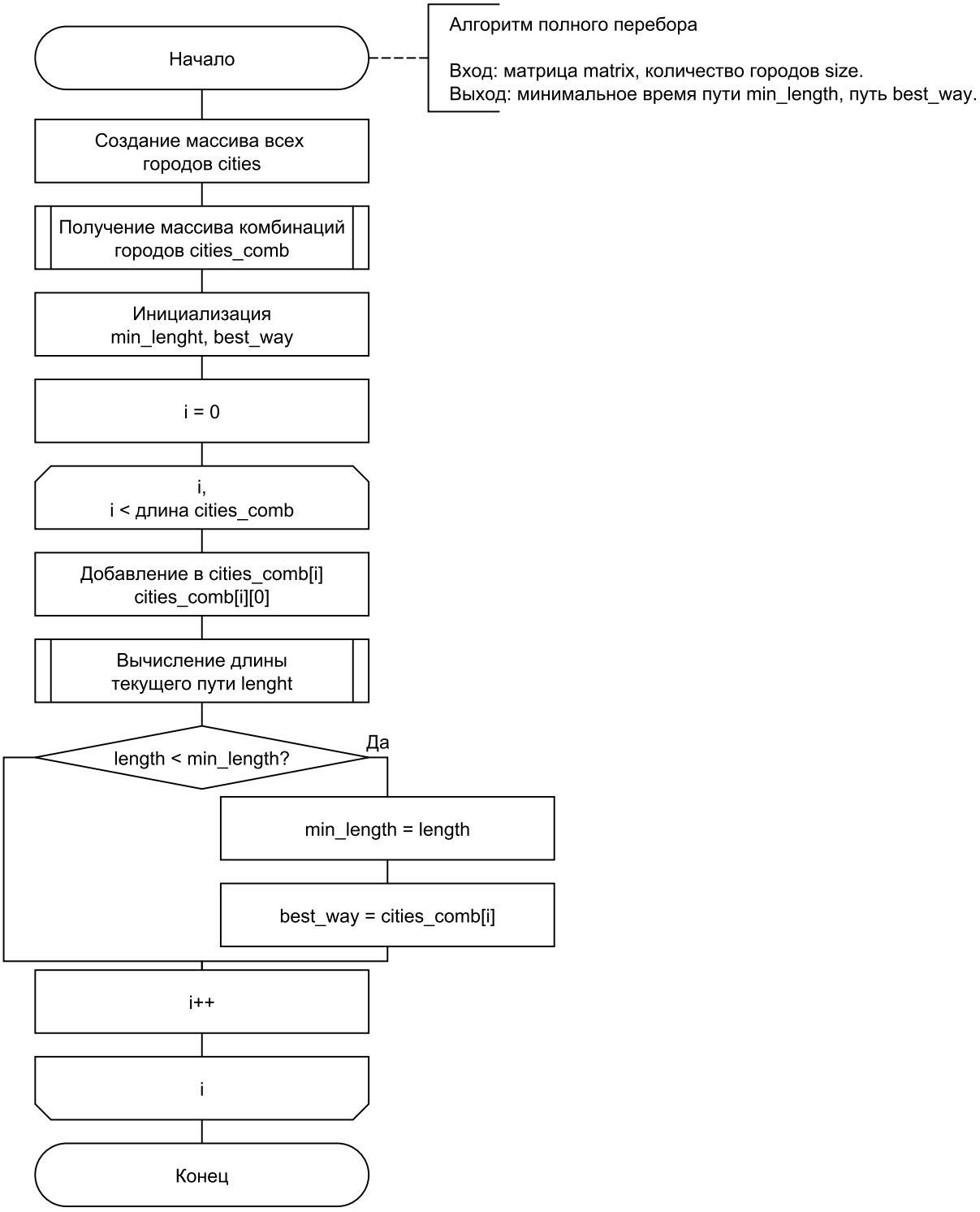

Рисунок 2.1 – Схема алгоритма полного перебора

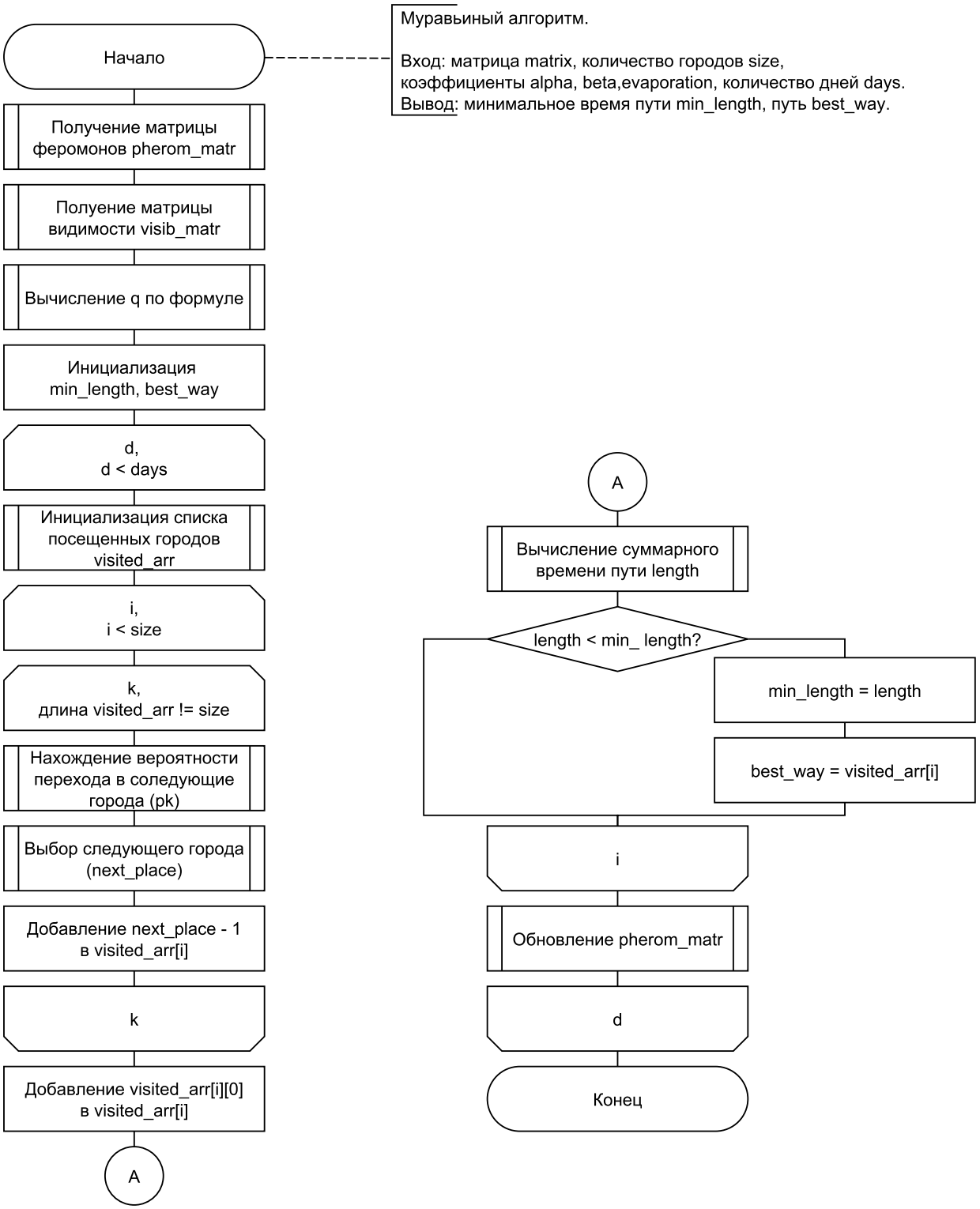

Рисунок 2.2 – Схема муравьиного алгоритма

2.2 Классы эквивалентности тестирования

Для тестирования выделены следующие классы эквивалентности:

1. Неверно выбран пункт меню — не число или число, меньшее 0 или большее 5;
2. Неверно введены коэффициенты α, β, *evaporation_koef* - не число или число, меньшее 0;
3. Неверно введено количество дней — не число или число, меньшее 0;
4. Неверно введен размер матрицы — не число или число, меньшее 2;
5. Корректный ввод всех параметров.

2.3 Вывод

В данном разделе были представлены схемы алгоритмов, рассматриваемых в лабораторной работе.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций алгоритмов полного перебора и муравьиного.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [2]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time* из библиотеки *time* [1].

3.2 Описание используемых типов данных

При реализации будут использованы следующие типы и структуры данных:

— размер матрицы смежности — целое число типа *int*;

— название файла — строка типа *str*;

— коэффициенты α, β, *evaporation_koef* — числа типа *float*;

— матрица времен, затрачиваемых на дорогу из одного города в другой — матрица типа *int*.

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

— *main.py* — файл, содержащий весь служебный код;

— *algorythms.py* — файл, содержащий реализации алгоритмов.

3.4 Реализация алгоритмов

В листингах 3.1 — 3.2 представлены реализации алгоритмов полного перебора и муравьиного.

Листинг 3.1 – Реализация алгоритма полного перебора

```
def full_combinations(matrix, size):
    places = np.arange(size)
    places_combs = list()
    for combination in it.permutations(places):
        comb_arr = list(combination)
        places_combs.append(comb_arr)
    min_dist = float("inf")

    best_way = None
    for i in range(len(places_combs)):
        places_combs[i].append(places_combs[i][0])
        cur_dist = 0

        for j in range(size):
            start_city = places_combs[i][j]
            end_city = places_combs[i][j + 1]
            cur_dist += matrix[start_city][end_city]

        if cur_dist < min_dist:
            min_dist = cur_dist
            best_way = places_combs[i]

    return min_dist, best_way
```

Листинг 3.2 – Реализация муравьиного алгоритма

```
def ant_algorythm(matrix, places, alpha, beta, k_evaporation, days):
    q = calc_q(matrix, places)

    best_way = []
    min_length = float("inf")

    pheromones = get_pheromones(places)
    visibility = get_visibility(matrix, places)

```

```
10      ants = places
11
12      for day in range(days):
13          route = np.arange(places)
14
15          visited = get_visited_places(route, ants)
16
17          for ant in range(ants):
18              while len(visited[ant]) != ants:
19                  pk = find_posibilyties(pheromones, visibility,
                        visited, places, ant, alpha, beta)
20                  chosen_place = choose_next_place_by_posibility(pk)
21                  visited[ant].append(chosen_place - 1)
22              visited[ant].append(visited[ant][0])
23              # Длина текущего маршрута
24              cur_length = calc_length(matrix, visited[ant])
25              if cur_length < min_length:
26                  min_length = cur_length
27                  best_way = visited[ant]
28          # Обновить значения
29          pheromones = update_pheromones(matrix, places, visited,
                pheromones, q, k_evaporation)
30
31      return min_length, best_way
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы полного перебора и муравьиного. В качестве результата ожидается число и массив: суммарное время пути и маршрут. Тесты *для всех алгоритмов* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица	Ожидаемый результат	Результат программы
$\begin{pmatrix} 0 & 4 & 2 & 1 & 7 \\ 4 & 0 & 3 & 7 & 2 \\ 2 & 3 & 0 & 10 & 3 \\ 1 & 7 & 10 & 0 & 9 \\ 7 & 2 & 3 & 9 & 0 \end{pmatrix}$	15, [0, 2, 4, 1, 3, 0]	15, [0, 2, 4, 1, 3, 0]
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$	4, [0, 1, 2, 0]	4, [0, 1, 2, 0]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$	64, [0, 1, 2, 3, 0]	64, [0, 1, 2, 3, 0]

3.6 Вывод

Были представлены реализации алгоритмов полного перебора и муравьиного, которые были описаны в предыдущем разделе. Также в данном разделе была приведена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

— операционная система: Windows 11, x64;

— оперативная память: 8 Гб;

— процессор: AMD Ryzen 5 5500U с видеокартой Radeon Graphics 2.10 ГГц.

Во время замеров времени ноутбук был нагружен только встроенными приложениями окружения.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.

```
1. Полный перебор
2. Муравьиный алгоритм
3. Параметризация
4. Замерить время
5. Сгенерировать новую матрицу
0. Выход

Выберите команду: 2
Введите имя файла: matr1.txt


Введите коэффициент alpha: 0.2
Введите коэффициент испарения: 0.6
Введите кол-во дней: 200


Минимальная сумма пути =  16
Путь:  [5, 7, 6, 8, 4, 9, 0, 2, 1, 3, 5]
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени *process_time* из библиотеки *time* на *Python*. Функция возвращает пользовательское процессорное время типа *float*.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для размеров матрицы от 2 до 10 с шагом 1 по 10 раз на различных входных данных.

Результаты замеров приведены в таблице 4.1 (время в с).

Таблица 4.1 – Результаты замеров времени

Размер матрицы	Полный перебор, с	Муравьиный алгоритм, с
2	0.000019	0.004641
3	0.000053	0.013441
4	0.000075	0.032406
5	0.000313	0.065612
6	0.001941	0.132931
7	0.025000	0.226562
8	0.153125	0.371875
9	1.481250	0.556250
10	16.020312	0.801562

На рисунке 4.2 приведена визуализация результатов замеров.

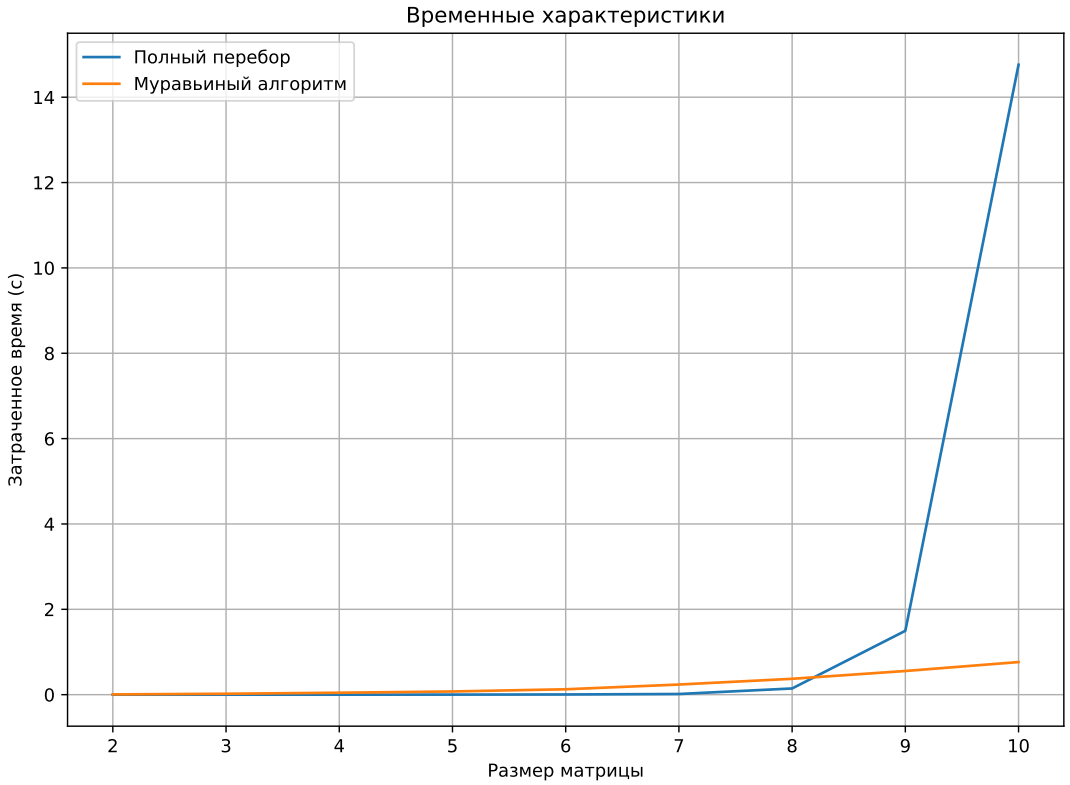

Рисунок 4.2 – Визуализация результатов замеров

4.4 Параметризация

Параметризация была проведена на классе данных, состоящем из трех матриц размером 10×10. Муравьиный алгоритм был запущен для всех значений $\alpha, \rho \in [0; 1]$ с шагом 0.2. В качестве эталонного значения был взят результат работы алгоритма полного перебора.

Далее будут представлены матрицы, на которых происходила параметризация и таблицы с результатами её выполнения.

В качестве класса данных были взяты матрицы, в которых все значения незначительно отличаются друг от друга и находятся в диапазоне [1, 8].

$$M_1 = \begin{pmatrix} 0 & 1 & 3 & 1 & 1 & 2 & 7 & 6 & 4 & 5 \\ 8 & 0 & 8 & 7 & 3 & 7 & 6 & 2 & 7 & 7 \\ 5 & 1 & 0 & 5 & 4 & 4 & 4 & 8 & 4 & 3 \\ 8 & 7 & 7 & 0 & 3 & 6 & 1 & 2 & 8 & 2 \\ 6 & 2 & 2 & 7 & 0 & 8 & 1 & 7 & 3 & 2 \\ 5 & 8 & 8 & 7 & 2 & 0 & 2 & 5 & 2 & 7 \\ 7 & 2 & 4 & 8 & 7 & 4 & 0 & 8 & 6 & 5 \\ 5 & 7 & 2 & 2 & 7 & 4 & 1 & 0 & 6 & 5 \\ 7 & 6 & 3 & 3 & 1 & 7 & 2 & 2 & 0 & 8 \\ 4 & 2 & 6 & 6 & 1 & 8 & 4 & 3 & 8 & 0 \end{pmatrix} \tag{4.1}$$

$$M_2 = \begin{pmatrix} 0 & 7 & 4 & 7 & 1 & 3 & 1 & 1 & 2 & 3 \\ 6 & 0 & 5 & 5 & 3 & 2 & 6 & 8 & 3 & 3 \\ 7 & 6 & 0 & 2 & 1 & 1 & 5 & 7 & 1 & 8 \\ 7 & 7 & 5 & 0 & 1 & 3 & 8 & 8 & 6 & 8 \\ 8 & 4 & 7 & 5 & 0 & 4 & 8 & 1 & 4 & 5 \\ 7 & 7 & 7 & 3 & 8 & 0 & 3 & 3 & 4 & 3 \\ 6 & 6 & 6 & 2 & 4 & 3 & 0 & 4 & 4 & 3 \\ 4 & 2 & 3 & 6 & 2 & 4 & 3 & 0 & 8 & 2 \\ 1 & 5 & 4 & 8 & 8 & 8 & 4 & 3 & 0 & 4 \\ 4 & 1 & 4 & 3 & 2 & 3 & 3 & 7 & 5 & 0 \end{pmatrix} \tag{4.2}$$

$$M_1 = \begin{pmatrix} 0 & 2 & 7 & 3 & 5 & 8 & 2 & 3 & 4 & 1 \\ 4 & 0 & 4 & 2 & 3 & 4 & 7 & 2 & 7 & 2 \\ 7 & 2 & 0 & 6 & 6 & 3 & 2 & 3 & 4 & 8 \\ 7 & 6 & 1 & 0 & 4 & 3 & 7 & 3 & 6 & 2 \\ 5 & 1 & 8 & 4 & 0 & 5 & 1 & 8 & 5 & 3 \\ 3 & 7 & 3 & 6 & 1 & 0 & 5 & 1 & 2 & 7 \\ 2 & 1 & 5 & 8 & 5 & 5 & 0 & 1 & 3 & 1 \\ 6 & 3 & 7 & 7 & 8 & 5 & 8 & 0 & 7 & 5 \\ 6 & 1 & 3 & 3 & 4 & 6 & 5 & 4 & 0 & 7 \\ 6 & 5 & 4 & 6 & 3 & 2 & 2 & 7 & 5 & 0 \end{pmatrix} \tag{4.3}$$

В таблице 4.2 представлены результаты параметризации.

Таблица 4.2 – Результаты параметризации

α	ρ	Количество дней	Результат 1	Результат 2	Результат 3	max
0.0	0.0	1	12	12	8	12
0.0	0.0	10	5	5	3	5
0.0	0.0	50	1	0	1	1
0.0	0.0	100	0	0	2	2
0.0	0.0	500	0	0	0	0
0.0	0.2	1	13	16	10	16
0.0	0.2	10	2	4	2	4
0.0	0.2	50	1	1	2	2
0.0	0.2	100	0	2	0	2
0.0	0.2	500	0	0	0	0
0.0	0.4	1	12	10	9	12
0.0	0.4	10	4	7	3	7
0.0	0.4	50	0	0	2	2
0.0	0.4	100	1	0	2	2
0.0	0.4	500	0	0	0	0
0.0	0.6	1	3	8	2	8
0.0	0.6	10	3	3	4	4
0.0	0.6	50	0	0	1	1
0.0	0.6	100	1	0	1	1
0.0	0.6	500	0	0	0	0
0.0	0.8	1	10	10	11	11
0.0	0.8	10	2	0	3	3
0.0	0.8	50	1	0	2	2
0.0	0.8	100	1	0	0	1
0.0	0.8	500	0	0	0	0
0.0	1.0	1	10	8	2	10
0.0	1.0	10	1	4	4	4
0.0	1.0	50	2	2	2	2
0.0	1.0	100	1	2	0	2
0.0	1.0	500	0	0	0	0
0.2	0.0	1	12	13	15	15

Продолжение таблицы

0.2	0.0	10	7	10	5	10
0.2	0.0	50	4	1	2	4
0.2	0.0	100	1	0	1	1
0.2	0.0	500	1	0	0	1
0.2	0.2	1	10	10	11	11
0.2	0.2	10	5	7	3	7
0.2	0.2	50	1	2	2	2
0.2	0.2	100	0	1	1	1
0.2	0.2	500	0	0	1	1
0.2	0.4	1	14	9	10	14
0.2	0.4	10	4	5	3	5
0.2	0.4	50	1	2	2	2
0.2	0.4	100	1	0	1	1
0.2	0.4	500	0	0	0	0
0.2	0.6	1	10	7	5	10
0.2	0.6	10	4	3	5	5
0.2	0.6	50	3	0	2	3
0.2	0.6	100	0	2	1	2
0.2	0.6	500	0	0	0	0
0.2	0.8	1	10	17	6	17
0.2	0.8	10	7	1	4	7
0.2	0.8	50	1	4	2	4
0.2	0.8	100	1	0	2	2
0.2	0.8	500	1	0	0	1
0.2	1.0	1	9	18	8	18
0.2	1.0	10	5	4	3	5
0.2	1.0	50	2	3	4	4
0.2	1.0	100	1	3	2	3
0.2	1.0	500	0	0	1	1
0.4	0.0	1	8	13	9	13
0.4	0.0	10	1	6	5	6
0.4	0.0	50	2	6	2	6

Продолжение таблицы

0.4	0.0	100	0	5	0	5
0.4	0.0	500	1	1	1	1
0.4	0.2	1	11	21	6	21
0.4	0.2	10	3	9	5	9
0.4	0.2	50	2	2	1	2
0.4	0.2	100	4	2	1	4
0.4	0.2	500	1	0	0	1
0.4	0.4	1	13	14	5	14
0.4	0.4	10	8	3	5	8
0.4	0.4	50	1	4	1	4
0.4	0.4	100	3	2	2	3
0.4	0.4	500	1	0	1	1
0.4	0.6	1	18	5	9	18
0.4	0.6	10	1	2	7	7
0.4	0.6	50	2	0	3	3
0.4	0.6	100	0	2	2	2
0.4	0.6	500	1	0	0	1
0.4	0.8	1	14	10	9	14
0.4	0.8	10	1	4	4	4
0.4	0.8	50	5	1	1	5
0.4	0.8	100	1	3	3	3
0.4	0.8	500	1	2	0	2
0.4	1.0	1	10	19	9	19
0.4	1.0	10	1	8	5	8
0.4	1.0	50	5	4	2	5
0.4	1.0	100	3	0	2	3
0.4	1.0	500	0	3	2	3
0.6	0.0	1	11	12	9	12
0.6	0.0	10	1	10	4	10
0.6	0.0	50	2	7	5	7
0.6	0.0	100	3	5	1	5
0.6	0.0	500	1	2	1	2

Продолжение таблицы

0.6	0.2	1	8	10	15	15
0.6	0.2	10	6	15	7	15
0.6	0.2	50	6	7	3	7
0.6	0.2	100	1	0	2	2
0.6	0.2	500	1	1	1	1
0.6	0.4	1	9	15	7	15
0.6	0.4	10	6	5	5	6
0.6	0.4	50	3	3	4	4
0.6	0.4	100	3	5	2	5
0.6	0.4	500	1	0	1	1
0.6	0.6	1	6	11	7	11
0.6	0.6	10	6	9	3	9
0.6	0.6	50	5	5	4	5
0.6	0.6	100	2	4	3	4
0.6	0.6	500	1	3	2	3
0.6	0.8	1	11	14	9	14
0.6	0.8	10	9	11	5	11
0.6	0.8	50	5	7	4	7
0.6	0.8	100	2	5	3	5
0.6	0.8	500	1	2	0	2
0.6	1.0	1	13	10	3	13
0.6	1.0	10	4	10	5	10
0.6	1.0	50	1	0	2	2
0.6	1.0	100	2	0	1	2
0.6	1.0	500	1	4	1	4
0.8	0.0	1	13	20	10	20
0.8	0.0	10	7	12	5	12
0.8	0.0	50	2	6	1	6
0.8	0.0	100	3	8	2	8
0.8	0.0	500	2	3	1	3
0.8	0.2	1	16	16	18	18
0.8	0.2	10	15	4	7	15

Продолжение таблицы

0.8	0.2	50	4	9	2	9
0.8	0.2	100	2	4	4	4
0.8	0.2	500	0	4	1	4
0.8	0.4	1	11	11	5	11
0.8	0.4	10	8	11	9	11
0.8	0.4	50	5	6	4	6
0.8	0.4	100	5	6	3	6
0.8	0.4	500	1	3	2	3
0.8	0.6	1	18	5	12	18
0.8	0.6	10	8	11	8	11
0.8	0.6	50	6	6	6	6
0.8	0.6	100	2	4	3	4
0.8	0.6	500	2	5	1	5
0.8	0.8	1	16	17	11	17
0.8	0.8	10	3	13	8	13
0.8	0.8	50	4	5	2	5
0.8	0.8	100	2	6	4	6
0.8	0.8	500	1	5	2	5
0.8	1.0	1	15	15	12	15
0.8	1.0	10	8	10	6	10
0.8	1.0	50	6	9	4	9
0.8	1.0	100	3	6	2	6
0.8	1.0	500	4	2	1	4
1.0	0.0	1	8	23	9	23
1.0	0.0	10	8	9	7	9
1.0	0.0	50	9	7	6	9
1.0	0.0	100	5	8	5	8
1.0	0.0	500	4	4	0	4
1.0	0.2	1	5	16	10	16
1.0	0.2	10	9	9	5	9
1.0	0.2	50	7	4	3	7
1.0	0.2	100	4	8	5	8

Продолжение таблицы

1.0	0.2	500	5	6	3	6
1.0	0.4	1	10	21	13	21
1.0	0.4	10	8	12	4	12
1.0	0.4	50	7	7	4	7
1.0	0.4	100	7	4	3	7
1.0	0.4	500	3	6	4	6
1.0	0.6	1	18	20	11	20
1.0	0.6	10	8	12	7	12
1.0	0.6	50	6	1	5	6
1.0	0.6	100	4	6	5	6
1.0	0.6	500	3	6	3	6
1.0	0.8	1	16	18	16	18
1.0	0.8	10	13	10	4	13
1.0	0.8	50	7	7	5	7
1.0	0.8	100	5	4	5	5
1.0	0.8	500	5	4	2	5
1.0	1.0	1	12	17	12	17
1.0	1.0	10	8	10	5	10
1.0	1.0	50	8	7	3	8
1.0	1.0	100	4	4	2	4
1.0	1.0	500	3	3	2	3

4.5 Вывод

Как видно из графика 4.2, реализация алгоритма полного перебора работает быстрее реализации муравьиного алгоритма при размерах матрицы меньше 9.

В связи с результатами параметризации можно сделать вывод, что оптимальными значениями параметров являются $\alpha = 0.2, \rho = 0.4, \ = 500$.

Заключение

В результате было получено, что реализация муравьиного алгоритма работает быстрее реализации алгоритма полного перебора при размерах матрицы больше 8. А оптимальными параметрами для муравьиного алгоритма являются $\alpha = 0.2, \rho = 0.4, = 500$.

Цель, которая была поставлена в начале лабораторной работы была достигнута: исследованы алгоритмы полного перебора и муравьиного. В ходе выполнения были решены все задачи:

— описаны алгоритм полного перебора и муравьиный алгоритм;

— реализованы указанные алгоритмы;

— проведено тестирование по методу черного ящика для реализаций указанных алгоритмов;

— проведен сравнительный анализ по времени и по памяти реализаций указанных алгоритмов;

— описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

Список используемых источников

1. time — Time access and conversions [Электронный ресурс]. — URL: https://docs.python.org/3/library/time.html#functions (дата обр. 17.10.2023).
2. Welcome to Python [Электронный ресурс]. — URL: https://www.python.org (дата обр. 17.10.2023).
3. Задача коммивояжёра [Электронный ресурс]. — URL: http://mech.math.msu.su/~shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml (дата обр. 01.12.2023).
4. Муравьиные алгоритмы [Электронный ресурс]. — URL: https://blog.bullgare.com/wp-content/uploads/2019/05/aca.pdf (дата обр. 01.12.2023).
5. Решаем задачу коммивояжёра простым перебором [Электронный ресурс]. — URL: https://thecode.media/path-js/ (дата обр. 01.12.2023).