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»

на тему: «Методы решения задачи коммивояжера»

Вариант № 21

Студент ИУ7-51Б (Группа) _____ (Подпись, дата) Д. В. Шубенина (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Л. Л. Волкова (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является параметризация метода решения задачи коммивояжера на основе муравьиного метода.

Для достижения поставленной цели, необходимо решить следующие задачи:

1) описать задачу коммивояжера;
2) описать методы решения задачи коммивояжера: метод полного перебора и метод на основе муравьиного алгоритма;
3) реализовать данные алгоритмы;
4) сравнить реализованные алгоритмы по времени выполнения;
5) провести параметризацию муравьиного алгоритма.

1 Аналитическая часть

Задача коммивояжера — задача комбинаторной оптимизации, цель которой — нахождение самого выгодного маршрута, проходящего через указанные точки по одному разу с возвращением в исходную точку [1].

Задача рассматривается как в симметричном, так и в ассиметричном варианте. В асимметричном варианте задача представляется в виде ориентированного графа, где веса рёбер между одними и теми же вершинами зависят от направления движения [1].

1.1 Методы решения задачи коммивояжера

Полный перебор. Этот метод заключается в переборе всех возможных маршрутов в графе и выборе кратчайшего из них. Сложность такого алгоритма — $O(n!)$ [2], где n — количество городов.

Муравьиный алгоритм. Данный метод основан на принципах поведения колонии муравьев [2].

Муравьи, двигаясь из муравейника в поисках пищи, откладывают феромоны на своем пути. При этом, чем больше плотность феромона, тем короче путь, и, соответственно, чем длиннее путь, тем быстрее феромон испарится, и его плотность будет меньше [3].

Со временем муравьи оставят наибольшее количество феромонов на самом коротком участке пути, что приведет к тому, что большинство муравьев выберет этот самый короткий путь, и, следовательно, они оставят еще больше феромонов на нем, что уменьшит вероятность выбора других маршрутов [3].

При сведении алгоритма к математическим формулам, сначала определяется целевая функция:

$$n = \frac{1}{D}, \tag{1.1}$$

где D — расстояние до заданного пункта [3].

Далее вычисляются вероятности перехода в заданную точку:

$$P = \frac{t^e \cdot n^b}{\sum_{i=1}^{m} t_i^a \cdot n_i^b}, \tag{1.2}$$

где a, b — настраиваемые параметры,
t — концентрация феромона [3].

При $a = 0$ выбирается ближайший город и алгоритм становится «жадным» (выбирает только оптимальные или самые короткие расстояния), при $b = 0$ будут учитываться только след феромона, что может привести к сужению пространства поиска оптимального решения [3].

Последним производится обновление феромона:

$$t_{i+1} = (1 - p) \cdot t_i + \frac{Q}{L_0}, \tag{1.3}$$

где p настраивает скорость испарения феромона,
Q настраивает концентрацию нанесения феромона,
L_0 — длина пути на определенном участке [3].

Последовательность вышеизложенных действий повторяется, пока не будет найден оптимальный маршрут.

Одной из модификаций муравьиного алгоритма является элитарная муравьиная система. При таком подходе искусственно вводятся «элитные» муравьи, усиливающие уровень феромонов, оптимального на данный момент маршрута [4].

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также способы её решения: полным перебором и муравьиным алгоритмом.

2 Конструкторская часть

В данном разделе будут описаны требования к программному обеспечению и представлены схемы алгоритма полного перебора и муравьиного алгоритма.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

1) наличие интерфейса для выбора действий;
2) возможность сохранения сгенерированной матрицы смежности в файл и загрузки готовых матриц из файла;
3) возможность выбора алгоритма решения задачи коммивояжера.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— размер матрицы смежности — целое число;

— имя файла — строка;

— коэффициенты α, β, $k\_evaporation$, $elite\_deposit$ — вещественные числа;

— матрица смежности — матрица вещественных чисел.

2.3 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора, а на рисунке 2.2 — схема муравьиного алгоритма поиска путей.

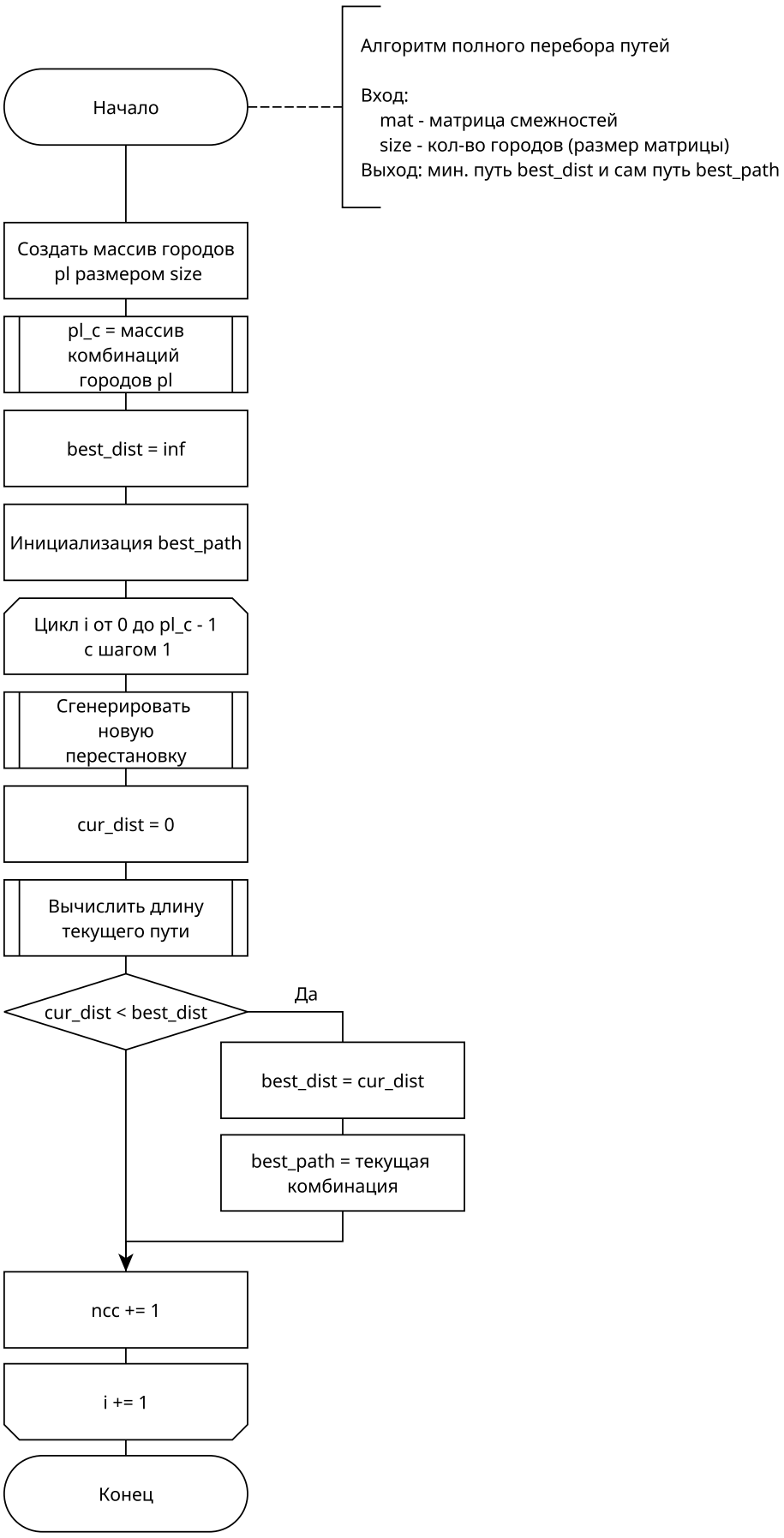

Рисунок 2.1 – Схема алгоритма полного перебора

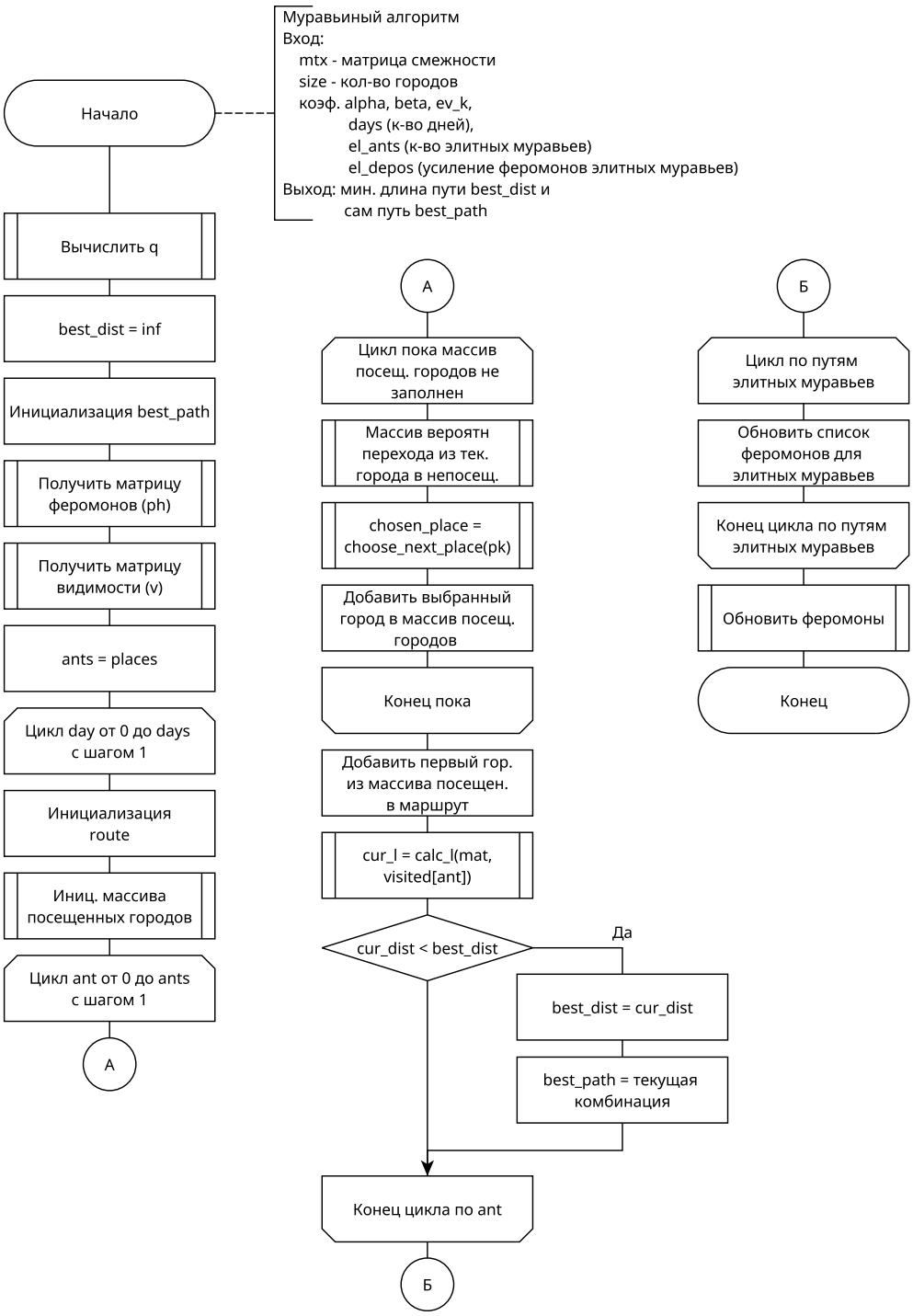

Рисунок 2.2 – Схема муравьиного алгоритма поиска путей

На рисунке 2.3 представлена схема алгоритма выбора следующего города, на рисунке 2.4 — схема алгоритма нахождения массива вероятностей переходов и на рисунке 2.5 показана схема алгоритма обновления матрицы феромонов.

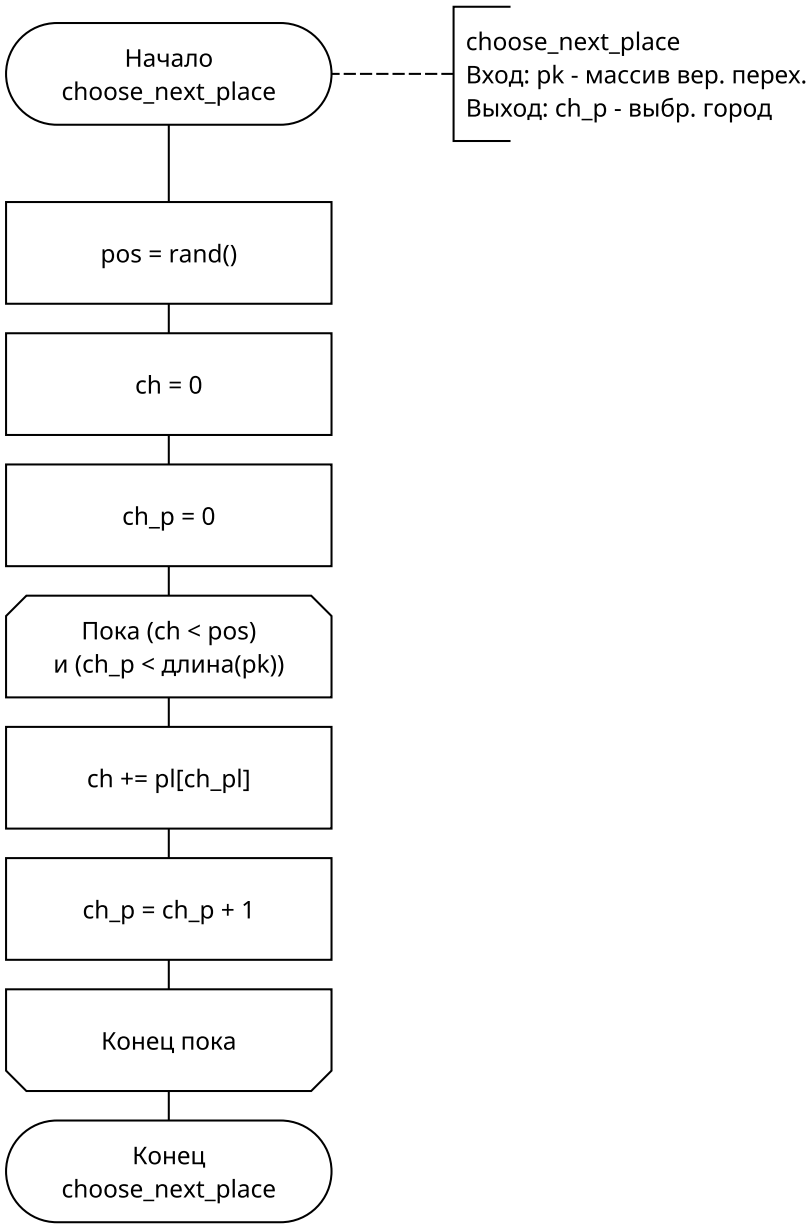

Рисунок 2.3 – Схема алгоритма выбора следующего города

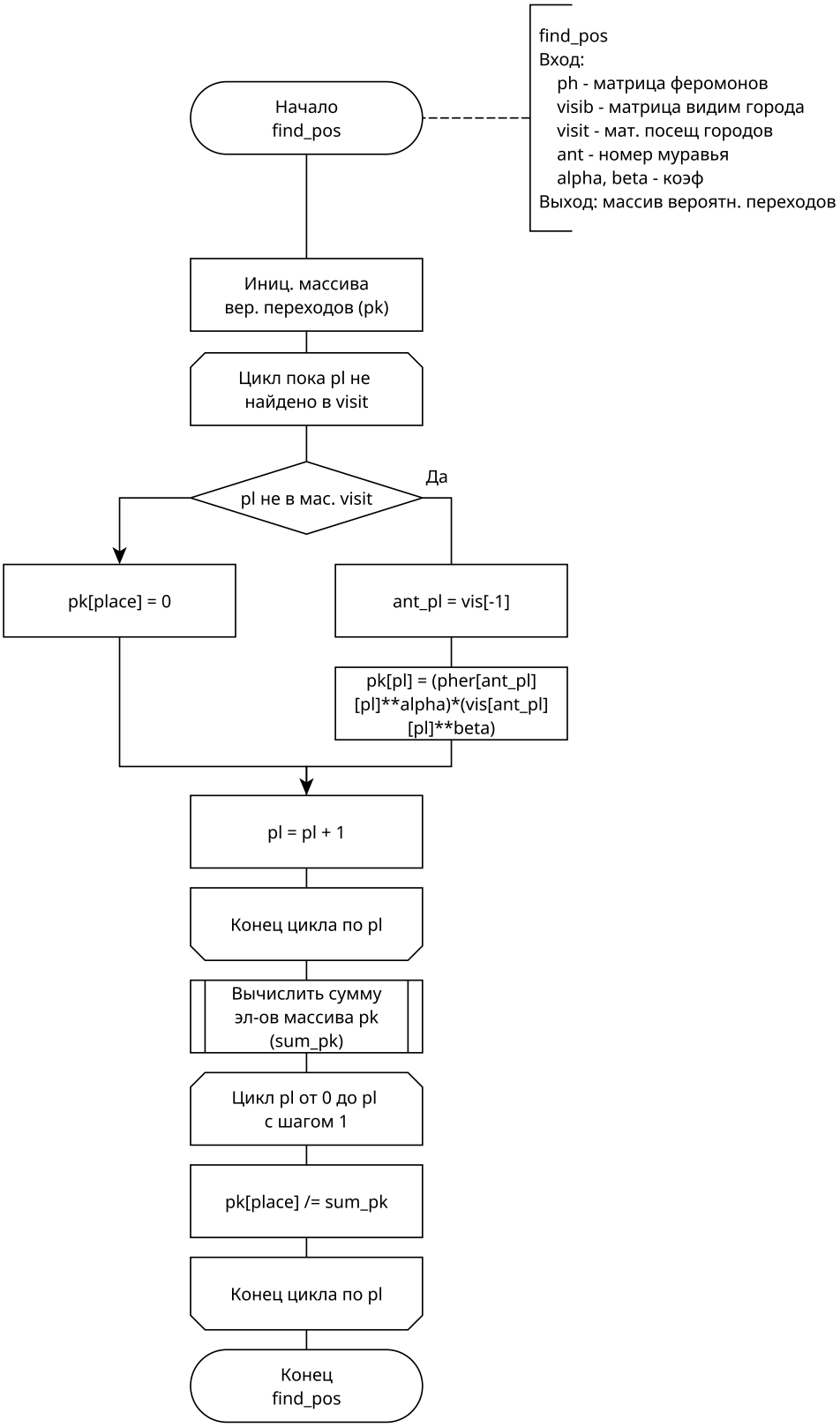

Рисунок 2.4 – Схема алгоритма нахождения массива вероятностей переходов

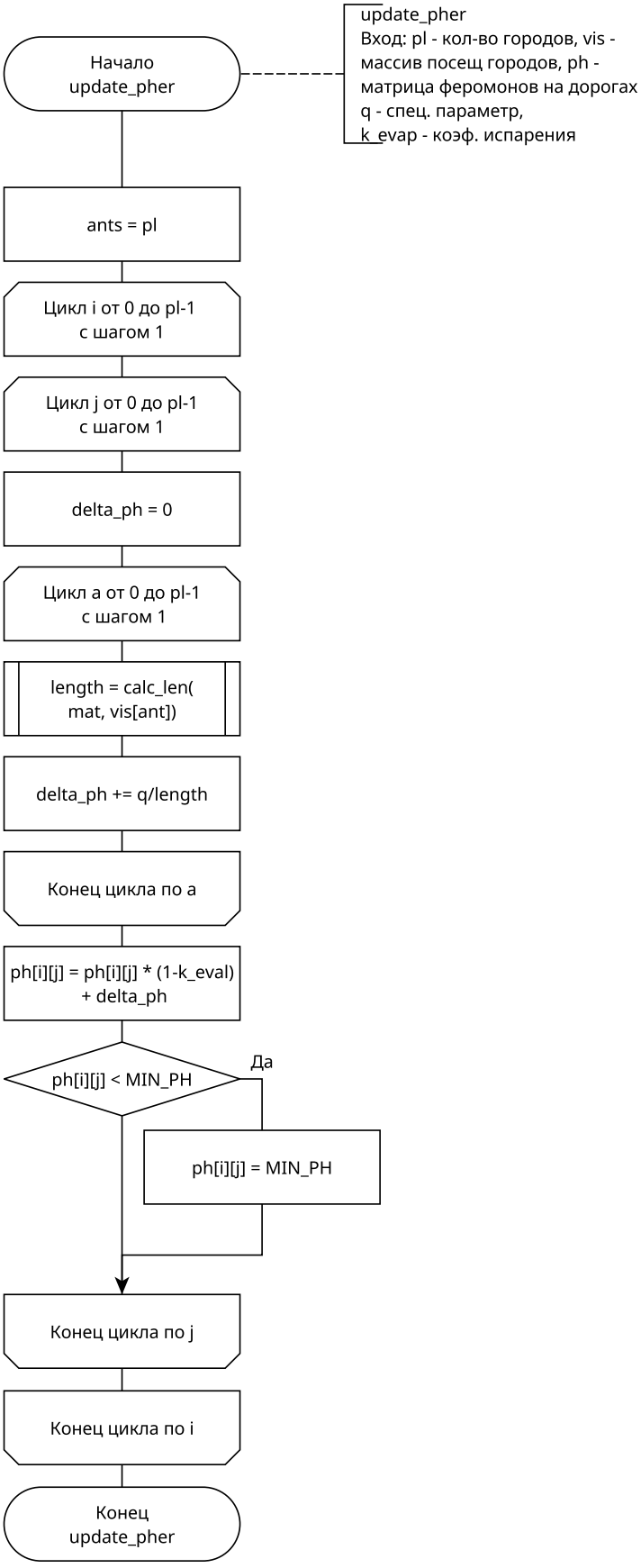

Рисунок 2.5 – Схема алгоритма обновления матрицы феромонов

Вывод

В данном разделе были перечислены требования к программному обеспечению и построены схемы алгоритмов решения задачи коммивояжера.

3 Технологическая часть

3.1 Средства реализации

В данной работе для реализации был выбран язык *Python* [5], поскольку стандартная библиотека данного языка содержит все инструменты, требующиеся для реализации программного обеспечения.

3.2 Средства замера времени

Для выполнения измерений процессорного времени работы выполняемой программы использовалась функция *process_time* из модуля *time* [6]. На листинге 3.1 приведен пример замера процессорного времени с помощью функции *process_time*.

Листинг 3.1 – Пример замера затраченного времени

```
REPS = 100
t = 0
for i in range(0, REPS):
    beg = process_time()
    func(...)
    end = process_time()
    t += end - beg
print('Время:', t / REPS)
```

3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

— `main.py` — файл, содержащий точку входа в программу;

— `utils.py` — файл, содержащий вспомогательные функции;

— `bruteforce.py` — файл, содержащий реализацию алгоритма полного перебора;

— `ants.py` — файл, содержащий реализацию муравьиного алгоритма;

— `test.py` — файл, содержащий функции замера процессорного времени работы алгоритмов и подбора параметров.

3.4 Реализация алгоритмов

В листинге 3.2 представлена реализация алгоритма полного перебора.

Листинг 3.2 – Реализация алгоритма полного перебора

```
def brute_force(matrix, size):
    places = np.arange(size)
    places_combs = list()

    for combination in it.permutations(places):
        combs = list(combination)
        places_combs.append(combs)

    best_dist = float("inf")

    for i in range(len(places_combs)):
        cur_dist = 0
        for j in range(size - 1):
            start = places_combs[i][j]
            end = places_combs[i][j + 1]
            cur_dist += matrix[start][end]

        if (cur_dist < best_dist):
            best_dist = cur_dist
            best_path = places_combs[i]

    return best_dist, best_path
```

В листинге 3.3 представлена реализация муравьиного алгоритма с элитными муравьями, в листинге 3.4 — реализация алгоритма обновления феромонов с учетом элитных муравьев, а в листинге 3.5 — реализация алгоритма нахождения массива вероятностей переходов в непосещенные города.

Листинг 3.3 – Реализация муравьиного алгоритма с элитными муравьями

```
def ant_algorithm(matrix, size, alpha, beta, k_evaporation,
   days, elite_ants, elite_pheromone_deposit=1.0):
    q = calc_q(matrix, size)
    best_path = []
    best_dist = float("inf")
    pheromones = calc_pheromones(size)
    ants = size

    elite_ant_paths = []
```

```
9
10      matrices = [matrix, matrix]
11      season = 0
12
13      visibilities = [
14          calc_visibility(matrices[0], size),
15          calc_visibility(matrices[1], size),
16      ]
17
18      elite_paths_updated = False
19
20      for day in range(days):
21          if day % 60 == 0:  # Update season
22              season = 1 - season
23              elite_paths_updated = False  # Reset elite paths
                    update flag
24
25          route = np.arange(size)
26          visited = calc_visited_places(route, ants)
27
28          for ant in range(ants):
29              while len(visited[ant]) != size:
30                  pk = find_paths(
31                      pheromones, visibilities[season], visited,
                            size, ant, alpha, beta
32                  )
33                  chosen_loc = choose_next_loc_by_possibility(pk)
34                  visited[ant].append(chosen_loc - 1)
35                  if ant < elite_ants:
36                      elite_ant_paths.append(visited[ant])
37
38              cur_length = calc_length(matrix, visited[ant])
39              if cur_length < best_dist:
40                  best_dist = cur_length
41                  best_path = visited[ant]
42
43              if ant < elite_ants and not elite_paths_updated:
44                  for elite_path in elite_ant_paths:
45                      for i in range(len(elite_path) - 1):
46                          pheromones[elite_path[i]][elite_path[i +
                                1]] *= (
```

```
                                1 + elite_pheromone_deposit
                            )
                    elite_paths_updated = True

            if day % 10 == 0:
                pheromones = update_pheromones_with_elite(
                    matrices[season],
                    size,
                    visited,
                    pheromones,
                    q,
                    k_evaporation,
                    elite_ant_paths,
                    elite_pheromone_deposit,
                )
                pheromones /= np.linalg.norm(pheromones)

    return best_dist, best_path
```

Листинг 3.4 – Реализация алгоритма обновления феромонов с учетом элитных муравьев

```
def update_pheromones_with_elite(
    matrix,
    places,
    visited,
    pheromones,
    q,
    k_evaporation,
    elite_ant_paths,
    elite_pheromone_deposit,
):
    ants = places
    for i in range(places):
        for j in range(places):
            delta = 0
            for ant in range(ants):
                length = calc_length(matrix, visited[ant])
                delta += q / length
            for elite_path in elite_ant_paths:
                if (i, j) in zip(elite_path, elite_path[1:]):
                    delta += elite_pheromone_deposit
            pheromones[i][j] *= 1 - k_evaporation
```

```
22                pheromones[i][j] += delta
23                if pheromones[i][j] < MIN_PHEROMONE:
24                    pheromones[i][j] = MIN_PHEROMONE
25        return pheromones
```

Листинг 3.5 – Реализация алгоритма нахождения массива вероятностей переходов в непосещенные города

```
1  def find_paths(pheromones, visibility, visited, places, ant,
       alpha, beta):
2      pk = [0] * places
3      for place in range(places):
4          if place not in visited[ant]:
5              ant_place = visited[ant][-1]
6              pk[place] = pow(pheromones[ant_place][place], alpha)
                   * pow(
7                  visibility[ant_place][place], beta
8              )
9          else:
10             pk[place] = 0
11     sum_pk = sum(pk)
12     for place in range(places):
13         pk[place] /= sum_pk
14     return pk
```

В листинге 3.6 показана реализация алгоритма нахождения массива вероятностей переходов в непосещенные города.

Листинг 3.6 – Реализация алгоритма нахождения массива вероятностей переходов в непосещенные города

```
1  def calc_pheromones(size):
2      min_phero = 1
3      pheromones = [[min_phero for _ in range(size)] for _ in
         range(size)]
4      return np.array(pheromones, dtype=np.float128)
```

В листинге 3.7 показана реализация алгоритма алгоритма выбора следующего города.

Листинг 3.7 – Реализация алгоритма алгоритма выбора следующего города

```
1  def choose_next_loc_by_possibility(pk):
2      posibility = random()
3      choice = 0
4      chosen_loc = 0
```

```
    while (choice < posibility) and (chosen_loc < len(pk)):
        choice += pk[chosen_loc]
        chosen_loc += 1
    return chosen_loc
```

3.5 Функциональные тесты

В таблице 3.1 приведены результаты функционального тестирования реализованных алгоритмов. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица смежности	Полный перебор	Муравьиный алгоритм
$\begin{pmatrix} 0 & 4 & 2 & 1 & 7 \\ 4 & 0 & 3 & 7 & 2 \\ 2 & 3 & 0 & 10 & 3 \\ 1 & 7 & 10 & 0 & 9 \\ 7 & 2 & 3 & 9 & 0 \end{pmatrix}$	15, [0, 2, 4, 1, 3]	15, [0, 2, 4, 1, 3]
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$	4, [0, 1, 2]	4, [0, 1, 2]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$	64, [0, 1, 2, 3]	64, [0, 1, 2, 3]

Вывод

Были представлены листинги всех реализаций алгоритмов — полного перебора и муравьиного. Также в данном разделе была приведена информации о выбранных средствах для разработки алгоритмов и сведения о модулях программы, проведено функциональное тестирование.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— процессор: AMD Ryzen 7 5800X (16) @ 3.800 ГГц.

— оперативная память: 32 ГБайт.

— операционная система: Manjaro Linux x86_64 (версия ядра Linux 6.5.12-1-MANJARO).

Измерения проводились на стационарном компьютере. Во время проведения измерений устройство было нагружено только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 продемонстрирована работа программы для случая, когда пользователь выбрал пункт 2 «Муравьиный алгоритм», выбрал датасет «cities.csv» и ввел следующие параметры:

— коэффициент α: 0;

— коэффициент испарения: 0.5;

— количество дней: 180;

— количество элитных муравьев: 2;

— коэффициент усиления феромонов элитных муравьев: 0.5.

```
Меню
    1. Полный перебор
    2. Муравьиный алгоритм
    3. Все алгоритмы
    4. Параметризация
    5. Замерить время
    6. Обновить данные
    7. Распечатать матрицу
    0. Выход
Выбор: 2
Доступные датасеты:
    1. cities.csv
    2. param_low_diff.csv
    3. param_high_diff.csv

Выберите файл: 1
    Введите коэффициент alpha: 0
    Введите коэффициент испарения: 0.5
    Введите кол-во дней: 180
    Введите количество элитных муравьев: 2
    Введите коэффициент усиления феромонов элитных муравьев: 0.5


Минимальная длина пути = 3725
Путь:  [1, 0, 2, 5, 3, 9, 4, 6, 7, 8]
Меню
    1. Полный перебор
    2. Муравьиный алгоритм
    3. Все алгоритмы
    4. Параметризация
    5. Замерить время
    6. Обновить данные
    7. Распечатать матрицу
    0. Выход
Выбор: 0
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Исследование реализаций алгоритмов по времени выполнения производилось при варьировании размера N матриц смежности от 2 до 11 с шагом 1.

В силу того, что время работы реализаций алгоритмов может колебаться в связи с различными процессами, происходящими в системе, для обеспече-

ния более точных результатов измерения для каждой реализации алгоритма повторялись 10 раз, а затем бралось их среднее арифметическое значение.

На рисунке 4.2 представлены результаты исследования реализаций алгоритмов по времени работы в зависимости от размера матриц смежности; таблица 4.1 содержит данные, по которым строились данные графики.

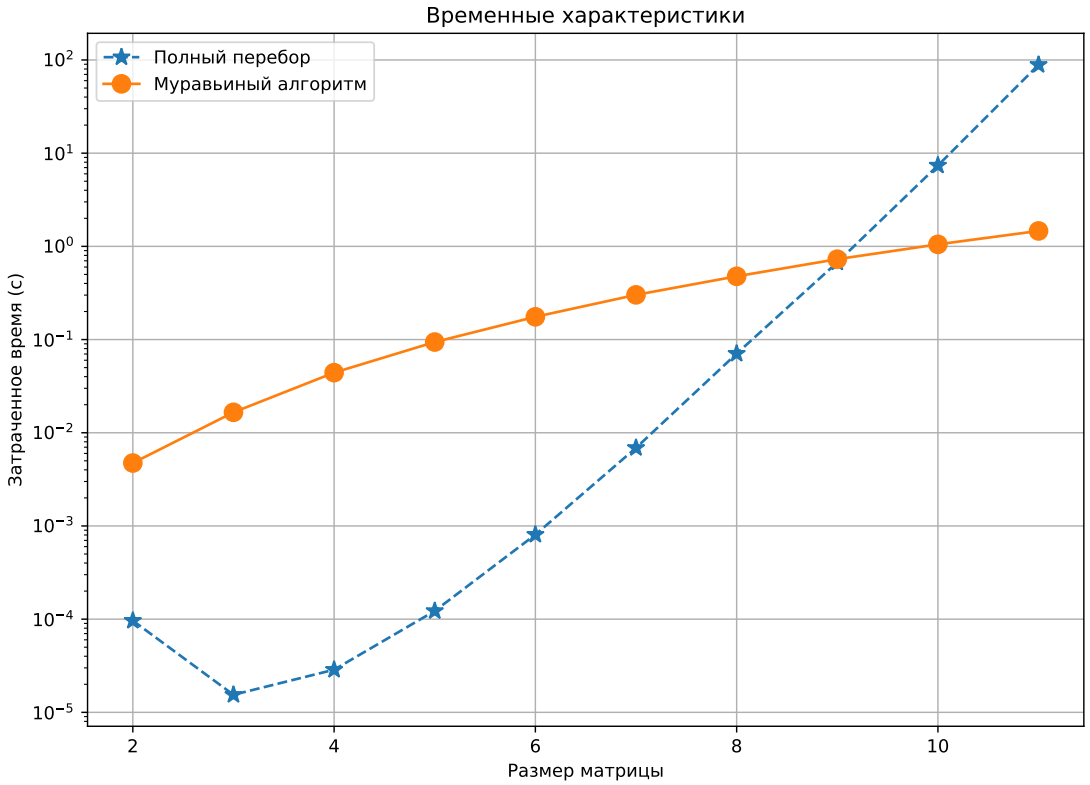

Рисунок 4.2 – Сравнение реализаций алгоритмов по времени работы при изменении размера матрицы смежности

Таблица 4.1 – Результаты измерений реализаций алгоритмов при изменении размера матрицы смежности

Размер матрицы	Полный перебор	Муравьиный алгоритм
2	0.000096	0.004733
3	0.000015	0.016570
4	0.000029	0.044153
5	0.000122	0.094063
6	0.000806	0.175447
7	0.006886	0.301783
8	0.070623	0.477132
9	0.673715	0.728651
10	7.379569	1.050827
11	88.714126	1.463882

4.4 Постановка исследования

Автоматическая параметризация была проведена на трех классах данных: 4.4.1, 4.4.2 и 4.4.3. Алгоритм будет запущен для набора значений $\alpha, \rho \in (0, 1)$.

Итоговая таблица значений параметризации будет состоять из следующих колонок:

— α — коэффициент жадности;

— ρ — коэффициент испарения;

— *days* — количество дней жизни колонии муравьёв;

— *Result* — эталонный результат, полученный методом полного перебора для проведения данного исследования;

— *Mistake* — разность полученного основанным на муравьином алгоритме методом значения и эталонного значения на данных значениях параметров, показатель качества решения.

Цель исследования — определить комбинацию параметров, которые позволяют решить задачу наилучшим образом для выбранного класса данных. Качество решения зависит от количества дней и погрешности измерений.

4.4.1 Класс данных 1

Класс данных 1 представляет собой матрицу смежности размером 10 элементов (небольшой разброс значений: от 1 до 2), которая представлена в (4.1).

$$K_1 = \begin{pmatrix} 0 & 1 & 1 & 2 & 2 & 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 2 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 0 & 2 & 2 & 1 & 1 & 2 & 2 \\ 2 & 2 & 2 & 0 & 1 & 2 & 1 & 2 & 2 \\ 2 & 1 & 2 & 1 & 0 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 2 & 0 & 1 & 1 & 2 \\ 1 & 2 & 1 & 1 & 2 & 1 & 0 & 2 & 2 \\ 1 & 1 & 2 & 2 & 1 & 1 & 2 & 0 & 2 \\ 2 & 1 & 2 & 2 & 1 & 2 & 2 & 2 & 0 \end{pmatrix} \tag{4.1}$$

Для данного класса данных приведена таблица 4.2 с выборкой параметров, которые наилучшим образом решают поставленную задачу. Использованы следующие обозначения: Days — количество дней, Result — результат работы, Mistake — ошибка как отклонение решения от эталонного.

Таблица 4.2 – Параметры для класса данных 1

α	ρ	Days	Result	Mistake
0.1	0.3	10	9	0
0.1	0.3	50	9	0
0.1	0.3	100	9	0
0.1	0.3	300	9	0
0.1	0.3	500	9	0
0.1	0.4	10	9	0
0.1	0.4	50	9	0
0.1	0.4	100	9	0
0.1	0.4	300	9	0
0.1	0.4	500	9	0
0.1	0.7	10	9	0
0.1	0.7	50	9	0

0.1	0.7	100	9	0
0.1	0.7	300	9	0
0.1	0.7	500	9	0
0.2	0.5	10	9	0
0.2	0.5	50	9	0
0.2	0.5	100	9	0
0.2	0.5	300	9	0
0.2	0.5	500	9	0
0.2	0.7	10	9	0
0.2	0.7	50	9	0
0.2	0.7	100	9	0
0.2	0.7	300	9	0
0.2	0.7	500	9	0
0.3	0.4	10	9	0
0.3	0.4	50	9	0
0.3	0.4	100	9	0
0.3	0.4	300	9	0
0.3	0.4	500	9	0
0.3	0.5	10	9	0
0.3	0.5	100	9	0
0.3	0.5	300	9	0
0.3	0.5	500	9	0
0.4	0.5	10	9	0
0.4	0.5	50	9	0
0.4	0.5	100	9	0
0.4	0.5	300	9	0
0.4	0.5	500	9	0
0.6	0.1	10	9	0
0.6	0.1	50	9	0
0.6	0.1	100	9	0
0.6	0.1	300	9	0
0.6	0.1	500	9	0

4.4.2 Класс данных 2

Класс данных 2 представляет собой матрицу смежности размером 9 элементов (большой разброс значений: от 1000 до 9999), которая представлена далее.

$$K_2 = \begin{pmatrix} 0 & 9271 & 8511 & 2010 & 1983 & 7296 & 7289 & 3024 & 1011 \\ 9271 & 0 & 7731 & 4865 & 5494 & 6812 & 4755 & 7780 & 7641 \\ 8511 & 7731 & 0 & 1515 & 9297 & 7506 & 5781 & 5804 & 7334 \\ 2010 & 4865 & 1515 & 0 & 3662 & 9597 & 2876 & 8188 & 9227 \\ 1983 & 5494 & 9297 & 3662 & 0 & 8700 & 4754 & 7445 & 3834 \\ 7296 & 6812 & 7506 & 9597 & 8700 & 0 & 4216 & 5553 & 8215 \\ 7289 & 4755 & 5781 & 2876 & 4754 & 4216 & 0 & 4001 & 4715 \\ 3024 & 7780 & 5804 & 8188 & 7445 & 5553 & 4001 & 0 & 9522 \\ 1011 & 7641 & 7334 & 9227 & 3834 & 8215 & 4715 & 9522 & 0 \end{pmatrix} \quad (4.2)$$

Для данного класса данных приведена таблица 4.3 с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.3 – Параметры для класса данных 2

α	ρ	Days	Result	Mistake
0.1	0.3	100	34192	0
0.1	0.3	300	34192	0
0.1	0.3	500	34192	0
0.1	0.7	100	34192	0
0.1	0.7	300	34192	0
0.1	0.7	500	34192	0
0.2	0.1	100	34192	0
0.2	0.1	300	34192	0
0.2	0.1	500	34192	0
0.2	0.2	100	34192	0
0.2	0.2	300	34192	0
0.2	0.2	500	34192	0
0.2	0.5	100	34192	0

0.2	0.5	300	34192	0
0.2	0.5	500	34192	0
0.2	0.8	100	34192	0
0.2	0.8	300	34192	0
0.2	0.8	500	34192	0
0.3	0.1	100	34192	0
0.3	0.1	300	34192	0
0.3	0.1	500	34192	0
0.3	0.2	5	34192	0
0.3	0.2	50	34192	0
0.3	0.2	100	34192	0
0.3	0.2	300	34192	0
0.3	0.2	500	34192	0
0.4	0.5	50	34192	0
0.4	0.5	300	34192	0
0.4	0.5	500	34192	0
0.5	0.2	100	34192	0
0.5	0.2	300	34192	0
0.5	0.2	500	34192	0
0.6	0.2	100	34192	0
0.6	0.2	300	34192	0
0.6	0.2	500	34192	0
0.6	0.3	300	34192	0
0.6	0.3	500	34192	0
0.6	0.4	100	34192	0
0.6	0.4	500	34192	0
0.6	0.5	100	34192	0
0.6	0.5	300	34192	0
0.6	0.5	500	34192	0

4.4.3 Класс данных 3

Класс данных 3 представляет собой матрицу расстояний между следующими городами: Москва, Великий Новгород, Тверь, Ярославль, Великий

Устюг, Коломна, Нижний Новгород, Казань, Астрахань, Вологда [7].

$$K_3 = \begin{pmatrix} 0 & 490 & 160 & 250 & 750 & 100 & 400 & 715 & 1270 & 410 \\ 490 & 0 & 330 & 515 & 880 & 590 & 800 & 1115 & 1760 & 700 \\ 160 & 330 & 0 & 255 & 740 & 265 & 500 & 820 & 1435 & 350 \\ 250 & 515 & 255 & 0 & 505 & 290 & 290 & 600 & 1370 & 180 \\ 750 & 880 & 740 & 505 & 0 & 770 & 510 & 575 & 1605 & 450 \\ 100 & 590 & 265 & 290 & 770 & 0 & 355 & 660 & 1170 & 465 \\ 400 & 800 & 500 & 290 & 510 & 355 & 0 & 325 & 1145 & 405 \\ 715 & 1115 & 820 & 600 & 575 & 660 & 325 & 0 & 1055 & 670 \\ 1270 & 1760 & 1435 & 1370 & 1605 & 1170 & 1145 & 1055 & 0 & 1530 \\ 410 & 700 & 350 & 180 & 450 & 465 & 405 & 670 & 1530 & 0 \end{pmatrix} \quad (4.3)$$

Для данного класса данных приведена таблица 4.4 с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.4 – Параметры для класса данных 3

α	ρ	Days	Result	Mistake
0.1	0.3	100	3725	0
0.1	0.3	300	3725	0
0.1	0.3	500	3725	0
0.1	0.7	100	3725	0
0.1	0.7	300	3725	0
0.1	0.7	500	3725	0
0.2	0.1	100	3725	0
0.2	0.1	300	3725	0
0.2	0.1	500	3725	0
0.2	0.2	100	3725	0
0.2	0.2	300	3725	0
0.2	0.2	500	3725	0
0.2	0.5	100	3725	0
0.2	0.5	300	3725	0
0.2	0.5	500	3725	0

0.2	0.8	100	3725	0
0.2	0.8	300	3725	0
0.2	0.8	500	3725	0
0.3	0.1	100	3725	0
0.3	0.1	300	3725	0
0.3	0.1	500	3725	0
0.3	0.2	5	3725	0
0.3	0.2	50	3725	0
0.3	0.2	100	3725	0
0.3	0.2	300	3725	0
0.3	0.2	500	3725	0
0.4	0.5	50	3725	0
0.4	0.5	300	3725	0
0.4	0.5	500	3725	0
0.5	0.2	100	3725	0
0.5	0.2	300	3725	0
0.5	0.2	500	3725	0
0.6	0.2	100	3725	0
0.6	0.2	300	3725	0
0.6	0.2	500	3725	0
0.6	0.3	300	3725	0
0.6	0.3	500	3725	0
0.6	0.4	100	3725	0
0.6	0.4	500	3725	0
0.6	0.5	100	3725	0
0.6	0.5	300	3725	0
0.6	0.5	500	3725	0

4.5 Вывод

В результате исследования было получено, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так, при размере матрицы, равном 2, муравьиный алгоритм медленнее алгоритма

полного перебора в 49 раз, а при размере матрицы, равном 10, муравьиный алгоритм быстрее алгоритма полного перебора в 7 раз, а при размере в 11 — уже в 60.5 раз. Следовательно, при размерах матриц больше 9 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует получения глобального оптимума при решении задачи.

Для класса данных 3 (см. п. 4.4.3) было получено, что наилучшим образом алгоритм работает на значениях параметров, которые представлены далее:

- $\alpha = 0.1, \rho = 0.3, 0.7$;
- $\alpha = 0.2, \rho = 0.1, 0.2, 0.5, 0.8$;
- $\alpha = 0.3, \rho = 0.1, 0.2$;
- $\alpha = 0.4, \rho = 0.5$;
- $\alpha = 0.5, \rho = 0.2$;
- $\alpha = 0.6, \rho = 0.2, 0.3, 0.4$.

Для этого класса данных рекомендуется использовать данные параметры.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы поставленная ранее цель была достигнута, решены все задачи, а именно:

1) описана задача коммивояжера;

2) описаны методы решения задачи коммивояжера: метод полного перебора и метод на основе муравьиного алгоритма;

3) реализованы данные алгоритмы;

4) проведено сравнение по времени реализованных алгоритмов и сделаны следующие выводы:

   — при размере матрицы, равном 2, муравьиный алгоритм медленнее алгоритма полного перебора в 49 раз, а при размере матрицы, равном 10, муравьиный алгоритм быстрее алгоритма полного перебора в 7 раз, а при размере в 11 — уже в 60.5 раз.

5) проведена параметризация муравьиного алгоритма и сделаны следующие выводы:

   — для класса данных 3 было получено, что наилучшим образом алгоритм работает на значениях параметров, которые представлены далее:

   — $\alpha = 0.1, \rho = 0.3, 0.7$;
   — $\alpha = 0.2, \rho = 0.1, 0.2, 0.5, 0.8$;
   — $\alpha = 0.3, \rho = 0.1, 0.2$;
   — $\alpha = 0.4, \rho = 0.5$;
   — $\alpha = 0.5, \rho = 0.2$;
   — $\alpha = 0.6, \rho = 0.2, 0.3, 0.4$.

   Для этого класса данных рекомендуется использовать данные параметры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Киселева А. А.* Обзор методов решения задачи коммивояжера //. — Издательство «Научно-исследовательские публикации» (ООО «Вэлборн»), 2019.
2. *Борознов В. О.* Исследование решения задачи коммивояжера. — АГТУ, Вестник Астраханского государственного технического университета. — Режим доступа: `https://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera/viewer` (дата обращения: 12.12.2023).
3. *Дробот К. С*, *Едемская Е. Н.* Решение задачи коммивояжера с помощью муравьиного алгоритма // Сборник материалов XI Международной научно-технической конференции в рамках VI Международного Научного форума Донецкой Народной Республики. — 2020. — С. 344—348.
4. *Коцюбинская С. А.* Задача глобальной оптимизации. Муравьиный алгоритм // Modern Science. — 2020. — С. 537—540.
5. Welcome to Python [Электронный ресурс]. — URL: `https://www.python.org`.
6. time — Time access and conversions [Электронный ресурс]. — URL: `https://docs.python.org/3/library/time.html#functions`.
7. *Рыбаков Б. А.* Русские карты Московии XV - начала XVI века. — Наука, 1974.