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 Комми вояжёр

Студент Фам Минь Хиеу

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л.Л., Строганов Д.В.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Муравьиный алгоритм — алгоритм для нахождения приближённых решений задач оптимизации на графах, таких, как задача коммивояжера, транспортная задача и аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию.

Целью данной лабораторной работы является изучение задачи коммивояжера, которая решается муравьиным алгоритмом и полным перебором.

Для достижения данной цели необходимо решить следующие задачи:

— описать метод полного перебора и муравьиный алгоритм для решения задачи коммивояжёра;

— построить схемы рассматриваемых алгоритмов;

— создать программное обеспечение, реализующее перечисленные алгоритмы;

— провести параметризацию метода, основанного на муравьином алгоритме;

— провести сравнительный анализ по времени реализованных алгоритмов;

— подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будет рассмотрена задача коммивояжера и описаны алгоритмы её решения.

1.1 Задача коммивояжера

Цель задачи коммивояжера заключается в нахождении самого выгодного маршрута (кратчайшего, самого быстрого, наиболее дешевого), проходящего через все заданные точки (пункты, города) по одному разу [1].

Решение методом грубой силы не подходило из-за вычислительной сложности. Была предпринята попытка реализовать метод ветвления и границ с отсечением в глубину. В целом, подход себя оправдывал, но иногда при некоторых специфических входных данных алгоритм выдавал решение далёкое от оптимального.

1.2 Алгоритм полного перебора для решения задачи коммивояжера

Пусть даны n городов и матрица расстояний между ними. Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них. Смысл перебора состоит в том, что мы перебираем все варианты объезда городов и выбираем оптимальный. Однако, при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n (сложность алгоритма равна $n!$).

Алгоритм полного перебора гарантирует точное решение задачи, однако, уже при небольшом числе городов будут большие затраты по времени выполнения.

1.3 Муравьиный алгоритм для решения задачи коммивояжера

Муравьиный алгоритм — это метаэвристический алгоритм, вдохновленный поведением муравьев при поиске пути к источнику пищи. Он был успешно применен для решения задачи коммивояжера, которая заключается в поиске кратчайшего пути, проходящего через все города один раз и возвращающегося в исходный город.

Муравьи действуют согласно следующим правилам.

1) Муравей запоминает посещенные города, причем каждый город может быть посещен только один раз. Обозначим через $J_{i,k}$ список городов, которые посетил муравей k, находящийся в городе i.

2) Муравей обладает видимостью η_{ij} - эвристическим желанием посетить город j, если муравей находится в городе i, причем

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

где D_{ij} — стоимость пути из города i в город j.

3) Муравей может улавливать след феромона - специального химического вещества. Число феромона на пути из города i в город j - τ_{ij}.

Муравей выполняет следующую последовательность действий, пока не посетит все города.

1) Муравей выбирает следующий город назначения, основываясь на вероятностно - пропорциональном правиле (1.2), в котором учитываются видимость и число феромона.

$$P_{ij,k} = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l=1}^{m}\tau_{il}^{\alpha}\eta_{il}^{\beta}}, \text{если город j необходимо посетить;} \\ 0, \text{иначе,} \end{cases} \tag{1.2}$$

где α - параметр влияния феромона, β - параметр влияния видимости пути, τ_{ij} - число феромона на ребре (ij), η_{ij} - эвристическое желание

посетить город j, если муравей находится в городе i. Выбор города является вероятностным, данное правило определяет ширину зоны города j, в общую зону всех городов $J_{i,k}$ бросается случайное число, которое и определяет выбор муравья.

2) Муравей проходит путь (ij) и оставляет на нем феромон.

Информация о числе феромона на пути используется другими муравьями для выбора пути. Те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащен феромоном. Следующие муравьи будут предпочитать именно этот путь, продолжая обогащать его феромоном.

После прохождения маршрутов всеми муравьями значение феромона на путях обновляется в соответствии со следующим правилом (1.3).

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

где ρ - коэффициент испарения. Чтобы найденное локальное решение не было единственным, моделируется испарение феромона.

При этом

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \tau_{ij,k}(t), \tag{1.4}$$

где m - число муравьев,

$$\Delta\tau_{ij,k} = \begin{cases} Q/L_k, \text{если k-ый муравей прошел путь (i,j);} \\ 0, \text{иначе.} \end{cases} \tag{1.5}$$

Вывод

Была описана задача коммивояжёра. Были рассмотрены подходы к решению задачи коммивояжера.

2 Конструкторская часть

В этом разделе представлены схемы алгоритма полного перебора и муравьиного алгоритма.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора, а на рисунках 2.2–2.6 представлена схема муравьиного алгоритма.

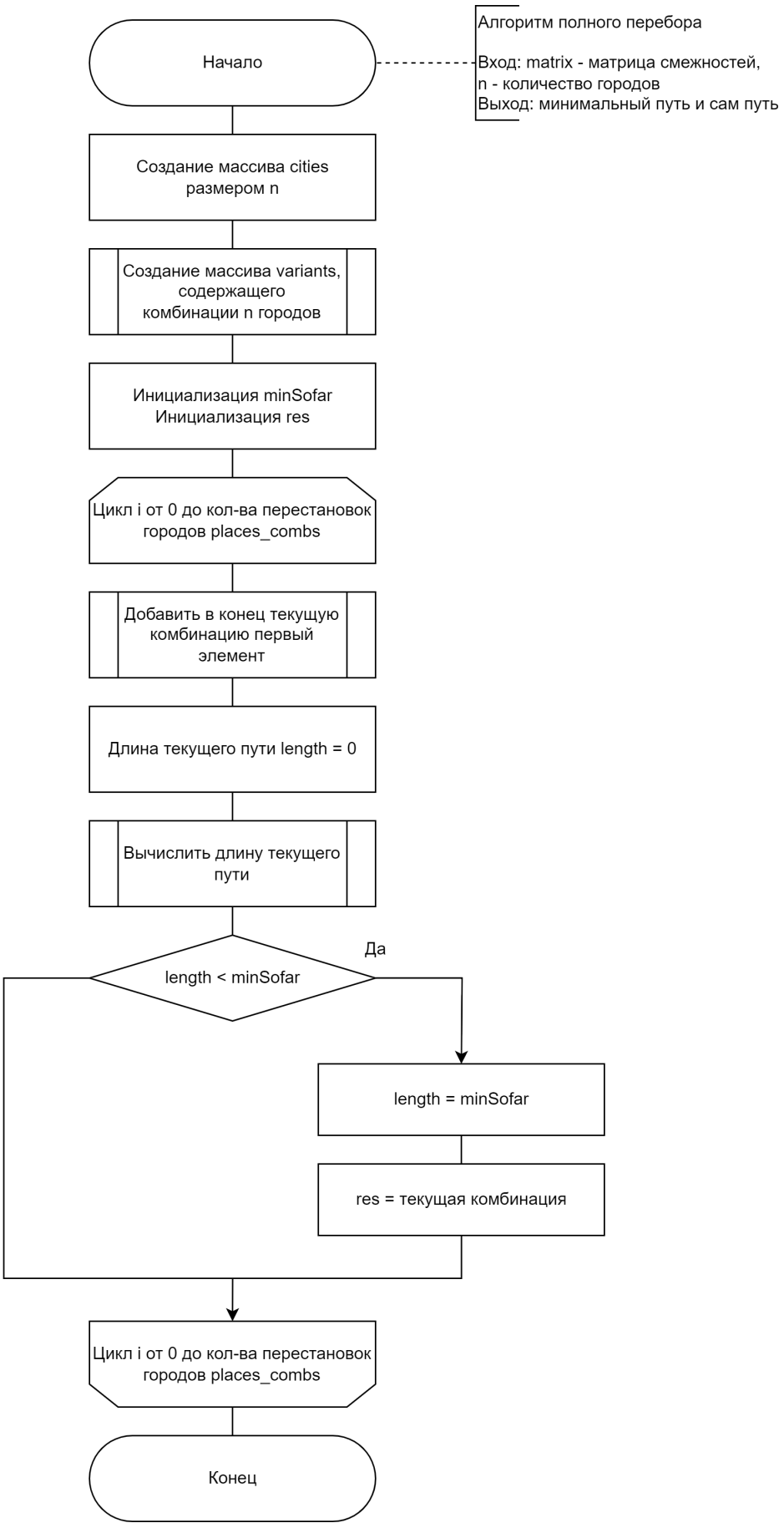

Рисунок 2.1 – Схема алгоритма полного перебора путей

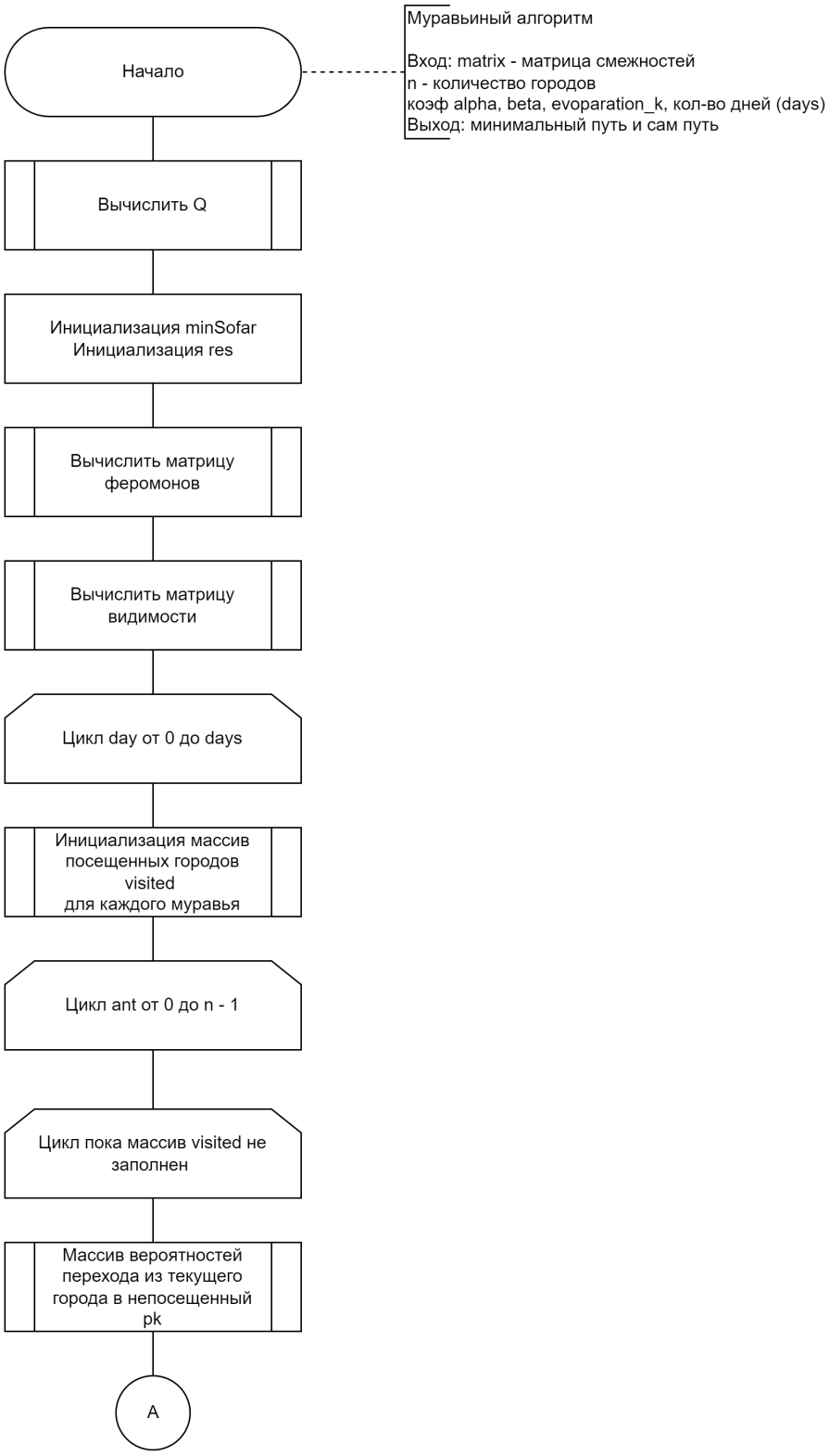

Рисунок 2.2 – Схема муравьиного алгоритма (часть 1)

Рисунок 2.3 – Схема муравьиного алгоритма (часть 2)

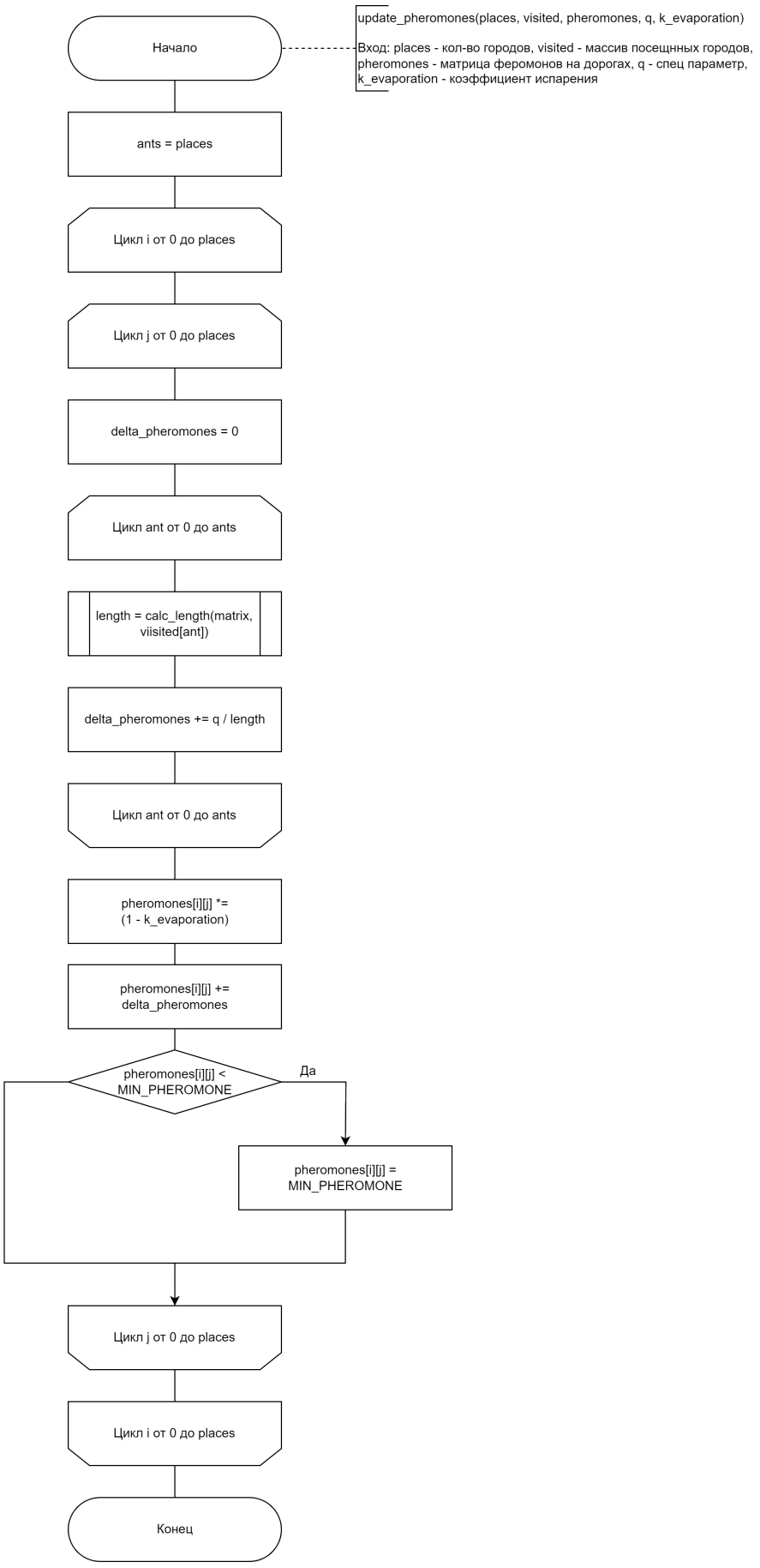

Рисунок 2.4 – Схема подпрограммы обновления матрицы феромона

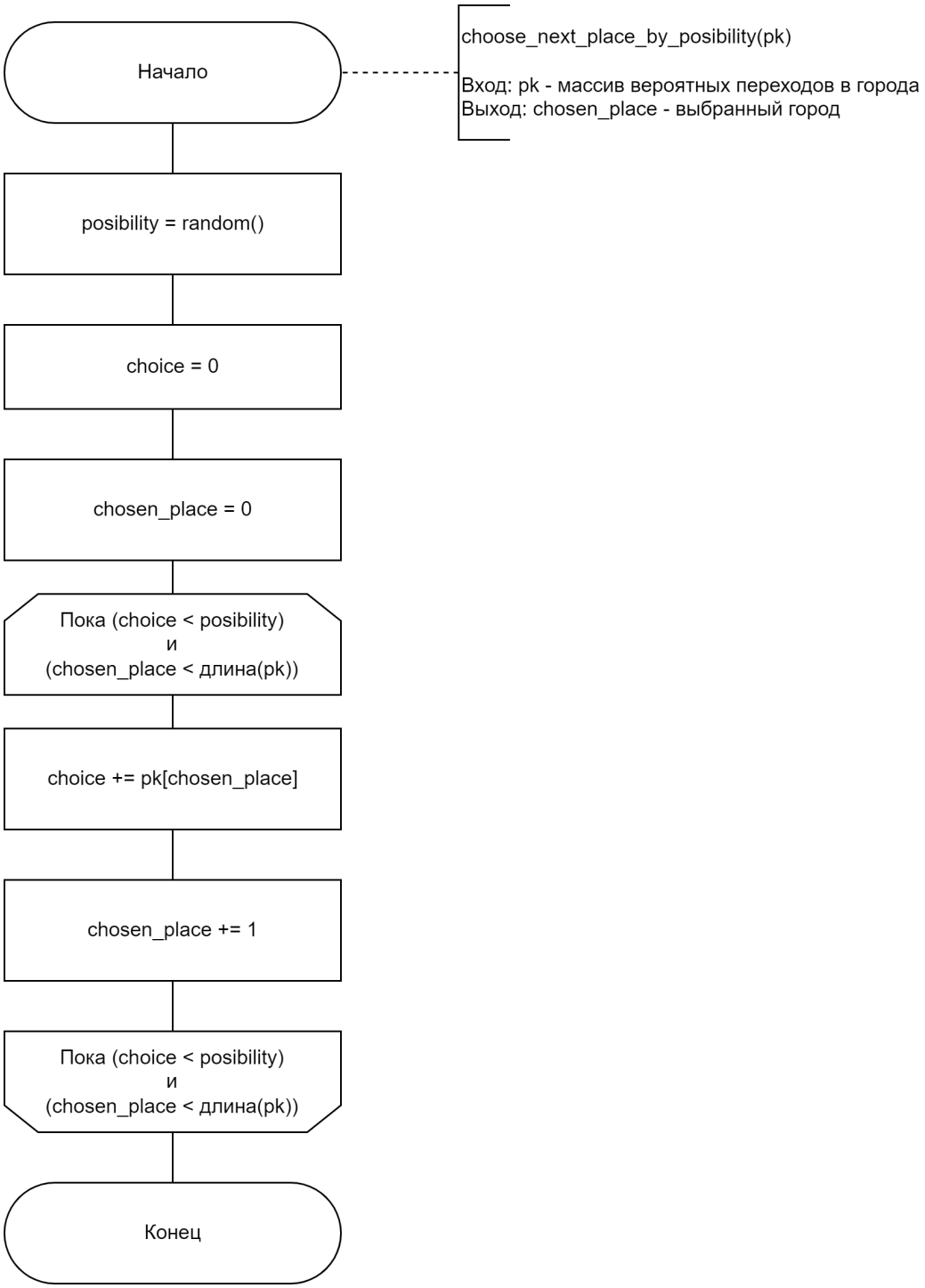

Рисунок 2.5 – Схема подпрограммы выбора следующего города

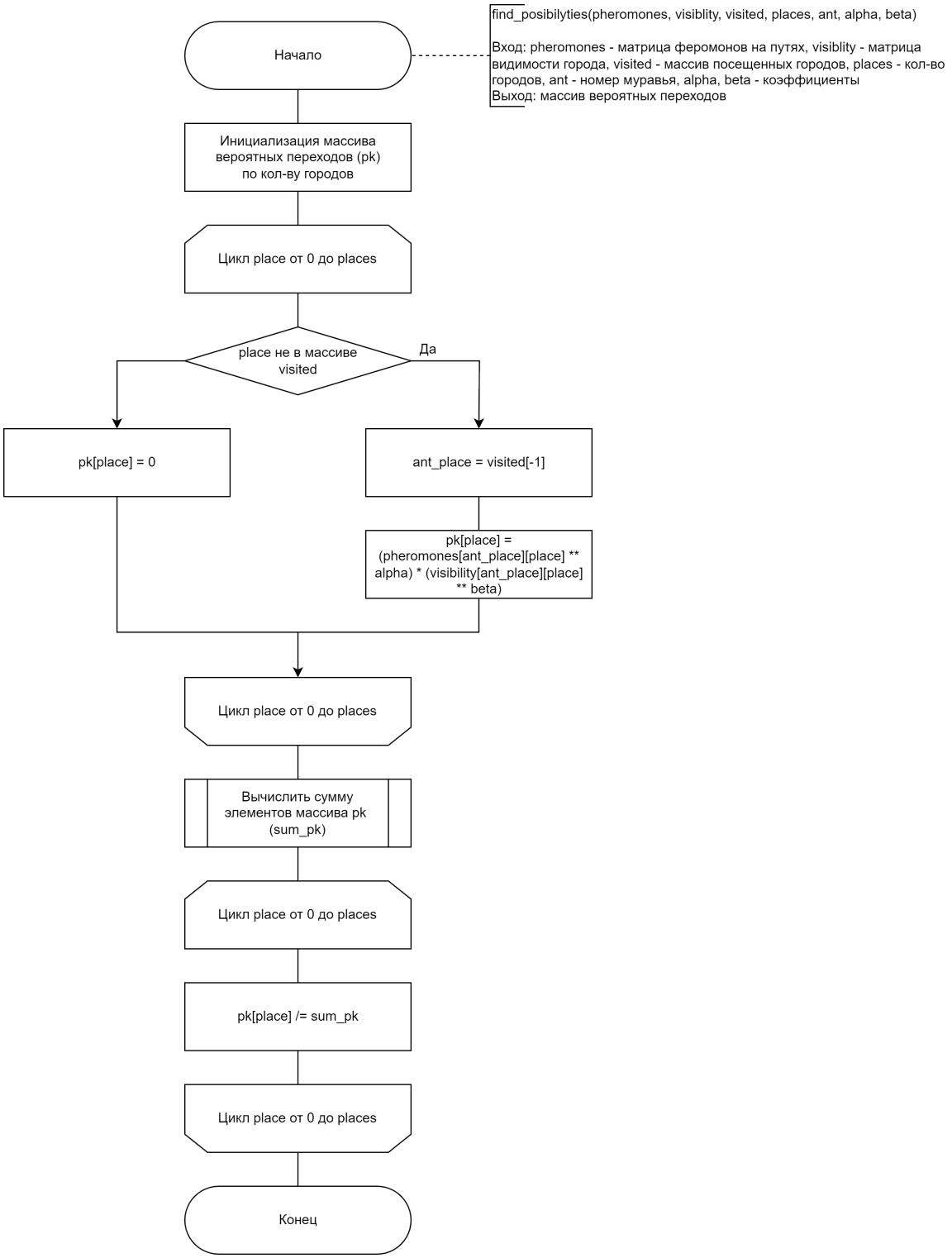

Рисунок 2.6 – Схема подпрограммы вычисления возможности объезда для соседних городов

Вывод

В данном разделе были построены схемы алгоритмов, рассматриваемых в данной работе.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программе, средства реализации, представлены листинги рассматриваемых алгоритмов, а также функциональные тесты.

3.1 Требования к программе

На вход программе должна подаваться матрица стоимостей, которая задает взвешенный неориентированный граф и количество городов. Выходные данные программы — оптимальный маршрут, проходящий через все заданные вершины по одному разу с последующим возвратом в исходную точку, и его стоимость. Программа должна работать в рамках следующих ограничений:

— стоимости путей должны быть целыми числами;

— число городов должно быть больше 1;

— число дней должно быть больше 0;

Пользователь должен иметь возможность выбора метода решения — полным перебором или муравьиным алгоритмом, и вывода результата на экран. Кроме того должна быть возможность проведения параметризации муравьиного алгоритма.

3.2 Средства реализации

Для реализации этих алгоритмов был выбран язык программирования *Python* [3]. Данный выбор обусловлен наличием у языка функции process_time измерения процессорного времени. Визуализация графиков с помощью библиотеки *Matplotlib* [4].

3.3 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма полного перебора. В листингах 3.2–3.5 представлена реализация муравьиного алгоритма.

Листинг 3.1 – Алгоритм полного перебора

```
def bruteForce(matrix, n):
    cities = np.arange(n)
    variants = []

    for comb in it.permutations(cities):
        variants.append(list(comb))

    res = []
    minSoFar = float("inf")

    for i in range(len(variants)):
        variants[i].append(variants[i][0])

        length = getDistance(matrix, n, variants[i])

        if length < minSoFar:
            minSoFar = length
            res = variants[i]

    return minSoFar, res
```

Листинг 3.2 – Муравьиный алгоритм

```
def antAlgo(matrix, size, alpha, beta, evaporation, days):
    pheromone = getPheromone(size)
    visibility = getVisibility(matrix, size)

    Q = getQ(matrix, size)

    res = []
    minSofar = float("inf")

    for _ in range(days):
        visited = getVisitedCity(np.arange(size), size)
        for i in range(size):
            while len(visited[i]) != size:
                pk = searchProbability(pheromone, visibility, visited,
                    size, i, alpha, beta)
                nextCity = getNextCity(pk)

                visited[i].append(nextCity - 1)
```

```
18
19              visited[i].append(visited[i][0])
20
21              currLen = getDistance(matrix, size, visited[i])
22
23              if currLen < minSofar:
24                  minSofar = currLen
25                  res = visited[i]
26
27          pheromone = update_pheromone(matrix, size, visited, pheromone,
                Q, evaporation)
28
29      return minSofar, res
```

Листинг 3.3 – Функция для обновления феромонов

```
1  def update_pheromone(matrix, size, visited, pheromone, q, evaporation):
2      ants = size
3
4      for i in range(size):
5          for j in range(size):
6              delta = 0
7
8              for ant in range(ants):
9                  length = getDistance(matrix, size, visited[ant])
10                 delta += q / length
11
12             pheromone[i][j] *= (1 - evaporation)
13             pheromone[i][j] += delta
14
15             if pheromone[i][j] < MIN_PHEROMONE:
16                pheromone[i][j] = MIN_PHEROMONE
17
18     return pheromone
```

Листинг 3.4 – Функция для нахождения вероятней перехода в каждый из городов

```
1  def searchProbability(pheromone, visibility, visited, size, ant,
       alpha, beta):
2      pk = [0] * size
3
4      for i in range(size):
5          if i not in visited[ant]:
6              ant_i = visited[ant][-1]
7
8              pk[i] = pow(pheromone[ant_i][i], alpha) * \
9                      pow(visibility[ant_i][i], beta)
10         else:
```

```
11              pk[i] = 0
12
13      pk_sum = sum(pk)
14
15      for i in range(size):
16          pk[i] /= pk_sum
17
18      return pk
```

Листинг 3.5 – Функция выбора следующего города

```
1  def getNextCity(pk):
2      size = len(pk)
3      numb = 0
4      i = 0
5
6      probability = random()
7
8      while numb < probability and i < size:
9          numb += pk[i]
10         i += 1
11
12     return i
```

3.4 Функциональные тесты

В таблице 3.1 приведены тесты для алгоритмов решения задачи коммивояжера (муравьиного алгоритма и алгоритма полного перебора). Все тесты пройдены успешно.

Вывод

В данном разделе были рассмотрены средства реализации лабораторной работы, представлены реализации алгоритмов, а также функциональные тесты.

Таблица 3.1 – Функциональные тесты

Входные данные	Выходные данные	
Матрица смежности	Результат	Ожидаемый результат
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix}$	6, [0, 1, 2, 0]	6, [0, 1, 2, 0]
$\begin{pmatrix} 0 & 3 & 4 & 1 \\ 3 & 0 & 1 & 1 \\ 4 & 1 & 0 & 2 \\ 1 & 1 & 2 & 0 \end{pmatrix}$	7, [0, 1, 2, 3, 0]	7, [0, 1, 2, 3, 0]
$\begin{pmatrix} 0 & 11 & 12 & 14 & 13 \\ 11 & 0 & 15 & 10 & 10 \\ 12 & 15 & 0 & 14 & 13 \\ 14 & 10 & 14 & 0 & 10 \\ 13 & 10 & 13 & 10 & 0 \end{pmatrix}$	56, [0, 1, 3, 4, 2, 0]	56, [0, 1, 3, 4, 2, 0]

4 Исследовательская часть

В данном разделе будут предоставлена информация о технических характеристиках устройства, представлены демонстрации работы программы и проведены замеры процессорного времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— операционная система Window 10 Home Single Language;

— память 8 Гб;

— процессор 11th Gen Intel(R) Core(TM) i7-1165G7 2.80 ГГц, 4 ядра.

4.2 Демонстрации работы программы

На рисунках 4.1–4.2 представлены результаты работы программы.

```
abcd2@HIEURUSSIA MINGW64 ~/OneDrive/Desktop/5-SEM-HIEU/AA/lab_06/src
$ python main.py

Меню
1. Полный перебор
2. Муравьиный алгоритм
3. Параметризация
4. Замеры времени
0. Выход

Введите команду:1

   0  10  10   1   1
  10   0   4   8   6
  10   4   0   7   5
   1   8   7   0   8
   1   6   5   8   0

Минимальная сумма пути: 19
Минимальный путь:  [0, 3, 1, 2, 4, 0]
```

Рисунок 4.1 – Демонстрация работы программы при полном переборе

```
Меню
1. Полный перебор
2. Муравьиный алгоритм
3. Параметризация
4. Замеры времени
0. Выход

Введите команду:2

   0  10  10   1   1
  10   0   4   8   6
  10   4   0   7   5
   1   8   7   0   8
   1   6   5   8   0

Коэффициент alpha =  0.6
Коэффициент beta  =  0.4
Коэффициент evaporation =  0.6

Введите кол-во дней: 80

Минимальная сумма пути:  19
Минимальный путь:  [0, 4, 2, 1, 3, 0]
```

Рисунок 4.2 – Демонстрация работы программы при муравьином алгоритме

4.3 Временные характеристики

В таблице 4.1 приведены результаты замеров по времени рассматриваемых алгоритмов.

Таблица 4.1 – Результаты замеров времени (неотсортированные массивы)

Количество городов	**Время, мс**	
	Полный перебор	**Муравьиный алгоритм**
2	0.0	0.015625
3	0.0	0.015625
4	0.0	0.03125
5	0.0	0.09375
6	0.0	0.171875
7	0.0	0.296875
8	0.125	0.46875
9	1.140625	0.71875
10	12.734375	1.078125

По таблице 4.1 был построен график, который иллюстрирует зависимость процессорного времени, затраченного реализациями алгоритмов , от количества городов — рис. 4.3. Исходя из этих данных можно понять, что при количестве городов больше 8 муравьиный алгоритм работает быстрее алгоритма полного перебора.

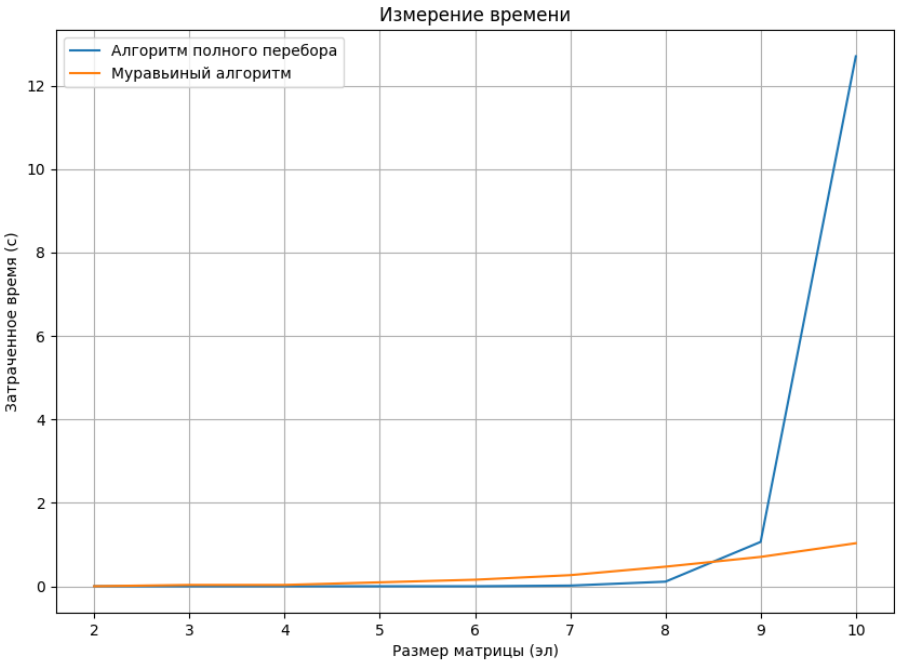

Рисунок 4.3 – Зависимость времени работы реализаций алгоритмов от размера матрицы

4.4 Автоматическая параметризация

Автоматическая параметризация была проведена на двух классах данных. Для проведение эксперимета были взяты матрицы размером 10x10. Муравьиный алгоритм был запущен для всех значений $\alpha, \rho \in (0, 1)$ с шагом 0.1.

В качестве эталонного значения был взят результат работы алгоритма полного перебора.

Далее будут представлены матрицы смежности (матрица 4.1 для класса данных, на котором происходила параметризация и таблицы с результатами её выполнения.

4.4.1 Класс данных

В качестве класса данных была взята матрица смежности, в которой все значения незначительно отличаются друг от друга, находятся в диапазоне [1, 3]. Таблица с результатами параметризации представлена в приложении А.

$$M_1 = \begin{pmatrix} 0 & 2 & 1 & 1 & 3 & 3 & 2 & 2 & 3 & 3 \\ 2 & 0 & 3 & 3 & 3 & 1 & 3 & 2 & 2 & 3 \\ 1 & 3 & 0 & 2 & 3 & 3 & 1 & 1 & 2 & 3 \\ 1 & 3 & 2 & 0 & 2 & 3 & 2 & 3 & 1 & 3 \\ 3 & 3 & 3 & 2 & 0 & 3 & 2 & 3 & 1 & 1 \\ 3 & 1 & 3 & 3 & 3 & 0 & 1 & 1 & 2 & 1 \\ 2 & 3 & 1 & 2 & 2 & 1 & 0 & 2 & 1 & 2 \\ 2 & 2 & 1 & 3 & 3 & 1 & 2 & 0 & 2 & 3 \\ 3 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 0 & 2 \\ 3 & 3 & 3 & 3 & 1 & 1 & 2 & 3 & 2 & 0 \end{pmatrix} \tag{4.1}$$

4.5 Вывод

В этом разделе были указаны технические характеристики машины, на которой происходило сравнение времени работы алгоритмов (муравьиного алгоритма и алгоритма полного перебора) решения задачи коммивояжера, также была рассмотрена автоматическая параметризация.

В результате замеров времени было установлено, что муравьиный алгоритм работает хуже алгоритма полного перебора на матрицах, размер которых меньше 9. Но при больших размерах муравьиный алгоритм существенно превосходит алгоритм полного перебора (на матрицах 9x9 он лучше в 2.1 раз, а на матрицах 10x10 уже в 15.4 раза).

На основе проведённой параметризации по двум классам данных можно сделать следующие выводы.

Для класса данных 4.1 лучше всего подходят следующие параметры:

- $\alpha = 0.1, \beta = 0.9, \rho = 0.2, 0.5$;
- $\alpha = 0.2, \beta = 0.8, \rho = 0.3, 0.4, 06$;
- $\alpha = 0.3, \beta = 0.7, \rho = 0.2$;

Таким образом, можно сделать вывод о том, что для лучшей работы муравьиного алгоритма на используемых классах данных нужно использовать полученные коэффиценты.

ЗАКЛЮЧЕНИЕ

Было экспериментально подтверждено различие во временной эффективности муравьиного алгоритма и алгоритма полного перебора решения задачи коммивояжера. В результате исследований можно сделать вывод о том, что при матрицах большого размера (больше 9) стоит использовать муравьиный алгоритм решения задачи коммивояжера, а не алгоритм полного перебора (на матрице размером 10x10 он работает в 15.4 раза быстрее). Также было установлено по результатам параметризации на экспериментальных класса данных, что при коэффициенте $\alpha = 0.1, 0.2, 0.3$ муравьиный алгоритм работает наилучшим образом.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены все задачи:

— описаны метод полного перебора и муравьиный алгоритм для решения задачи коммивояжёра;

— построены схемы рассматриваемых алгоритмов;

— создано программное обеспечение, реализующее перечисленные алгоритмы;

— проведена параметризация метода, основанного на муравьином алгоритме;

— проведен сравнительный анализ по времени реализованных алгоритмов;

— подготовлен отчет о выполненной лабораторной работе.

Список использованных источников

1. Решение задачи коммивояжера [Электронный ресурс]. Режим доступа: https://math.semestr.ru/kom/index.php

2. Муравьи и Python: ищем самые короткие пути [Электронный ресурс]. Режим доступа: https://vc.ru/newtechaudit/353372-muravi-i-python-ishchem-samye-korotkie-puti

3. Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org/

4. Matplotlib documentation [Электронный ресурс]. Режим доступа: https://matplotlib.org/stable/index.html

ПРИЛОЖЕНИЕ А

В таблицах А.1 представлены результаты параметризации муравьиного алгоритма.

Таблица А.1 – Параметры для класса данных 1

α	β	ρ	Дней	Результат	Ошибка
0.1	0.9	0.1	50	12	1
0.1	0.9	0.1	100	12	1
0.1	0.9	0.1	200	12	0
0.1	0.9	0.2	50	12	0
0.1	0.9	0.2	100	12	0
0.1	0.9	0.2	200	12	0
0.1	0.9	0.3	50	12	1
0.1	0.9	0.3	100	12	1
0.1	0.9	0.3	200	12	0
0.1	0.9	0.4	50	12	1
0.1	0.9	0.4	100	12	0
0.1	0.9	0.4	200	12	0
0.1	0.9	0.5	50	12	0
0.1	0.9	0.5	100	12	0
0.1	0.9	0.5	200	12	0
0.1	0.9	0.6	50	12	1
0.1	0.9	0.6	100	12	1
0.1	0.9	0.6	200	12	0
0.1	0.9	0.7	50	12	1
0.1	0.9	0.7	100	12	0
0.1	0.9	0.7	200	12	0
0.1	0.9	0.8	50	12	1
0.1	0.9	0.8	100	12	0
0.1	0.9	0.8	200	12	0
0.2	0.8	0.1	50	12	0
0.2	0.8	0.1	100	12	1
0.2	0.8	0.1	200	12	1

Продолжение таблицы А.1

α	β	ρ	Дней	Результат	Ошибка
0.2	0.8	0.2	50	12	1
0.2	0.8	0.2	100	12	0
0.2	0.8	0.2	200	12	0
0.2	0.8	0.3	50	12	0
0.2	0.8	0.3	100	12	0
0.2	0.8	0.3	200	12	0
0.2	0.8	0.4	50	12	0
0.2	0.8	0.4	100	12	0
0.2	0.8	0.4	200	12	0
0.2	0.8	0.5	50	12	1
0.2	0.8	0.5	100	12	0
0.2	0.8	0.5	200	12	1
0.2	0.8	0.6	50	12	0
0.2	0.8	0.6	100	12	0
0.2	0.8	0.6	200	12	0
0.2	0.8	0.7	50	12	1
0.2	0.8	0.7	100	12	1
0.2	0.8	0.7	200	12	0
0.2	0.8	0.8	50	12	1
0.2	0.8	0.8	100	12	0
0.2	0.8	0.8	200	12	0
0.3	0.7	0.1	50	12	0
0.3	0.7	0.1	100	12	1
0.3	0.7	0.1	200	12	1
0.3	0.7	0.2	50	12	0
0.3	0.7	0.2	100	12	0
0.3	0.7	0.2	200	12	0
0.3	0.7	0.3	50	12	1
0.3	0.7	0.3	100	12	1
0.3	0.7	0.3	200	12	0
0.3	0.7	0.4	50	12	1
0.3	0.7	0.4	100	12	0
0.3	0.7	0.4	200	12	1
0.3	0.7	0.5	50	12	0
0.3	0.7	0.5	100	12	1
0.3	0.7	0.5	200	12	0
0.3	0.7	0.6	50	12	1
0.3	0.7	0.6	100	12	1
0.3	0.7	0.6	200	12	1

Продолжение таблицы А.1

α	β	ρ	Дней	Результат	Ошибка
0.3	0.7	0.7	50	12	1
0.3	0.7	0.7	100	12	1
0.3	0.7	0.7	200	12	0
0.3	0.7	0.8	50	12	1
0.3	0.7	0.8	100	12	1
0.3	0.7	0.8	200	12	1
0.4	0.6	0.1	50	12	1
0.4	0.6	0.1	100	12	0
0.4	0.6	0.1	200	12	0
0.4	0.6	0.2	50	12	1
0.4	0.6	0.2	100	12	0
0.4	0.6	0.2	200	12	1
0.4	0.6	0.3	50	12	0
0.4	0.6	0.3	100	12	0
0.4	0.6	0.3	200	12	1
0.4	0.6	0.4	50	12	1
0.4	0.6	0.4	100	12	0
0.4	0.6	0.4	200	12	0
0.4	0.6	0.5	50	12	1
0.4	0.6	0.5	100	12	1
0.4	0.6	0.5	200	12	0
0.4	0.6	0.6	50	12	0
0.4	0.6	0.6	100	12	1
0.4	0.6	0.6	200	12	0
0.4	0.6	0.7	50	12	2
0.4	0.6	0.7	100	12	1
0.4	0.6	0.7	200	12	0
0.4	0.6	0.8	50	12	1
0.4	0.6	0.8	100	12	1
0.4	0.6	0.8	200	12	0
0.5	0.5	0.1	50	12	2
0.5	0.5	0.1	100	12	0
0.5	0.5	0.1	200	12	1
0.5	0.5	0.2	50	12	1
0.5	0.5	0.2	100	12	0
0.5	0.5	0.2	200	12	1
0.5	0.5	0.3	50	12	2
0.5	0.5	0.3	100	12	1
0.5	0.5	0.3	200	12	1

Продолжение таблицы А.1

α	β	ρ	Дней	Результат	Ошибка
0.5	0.5	0.4	50	12	2
0.5	0.5	0.4	100	12	1
0.5	0.5	0.4	200	12	0
0.5	0.5	0.5	50	12	2
0.5	0.5	0.5	100	12	1
0.5	0.5	0.5	200	12	1
0.5	0.5	0.6	50	12	2
0.5	0.5	0.6	100	12	1
0.5	0.5	0.6	200	12	1
0.5	0.5	0.7	50	12	2
0.5	0.5	0.7	100	12	0
0.5	0.5	0.7	200	12	0
0.5	0.5	0.8	50	12	1
0.5	0.5	0.8	100	12	1
0.5	0.5	0.8	200	12	1
0.6	0.4	0.1	50	12	1
0.6	0.4	0.1	100	12	0
0.6	0.4	0.1	200	12	1
0.6	0.4	0.2	50	12	2
0.6	0.4	0.2	100	12	1
0.6	0.4	0.2	200	12	1
0.6	0.4	0.3	50	12	1
0.6	0.4	0.3	100	12	2
0.6	0.4	0.3	200	12	1
0.6	0.4	0.4	50	12	1
0.6	0.4	0.4	100	12	0
0.6	0.4	0.4	200	12	0
0.6	0.4	0.5	50	12	1
0.6	0.4	0.5	100	12	1
0.6	0.4	0.5	200	12	1
0.6	0.4	0.6	50	12	1
0.6	0.4	0.6	100	12	1
0.6	0.4	0.6	200	12	1
0.6	0.4	0.7	50	12	1
0.6	0.4	0.7	100	12	1
0.6	0.4	0.7	200	12	1
0.6	0.4	0.8	50	12	1
0.6	0.4	0.8	100	12	2
0.6	0.4	0.8	200	12	1

Продолжение таблицы А.1

α	β	ρ	Дней	Результат	Ошибка
0.7	0.3	0.1	50	12	2
0.7	0.3	0.1	100	12	1
0.7	0.3	0.1	200	12	1
0.7	0.3	0.2	50	12	1
0.7	0.3	0.2	100	12	1
0.7	0.3	0.2	200	12	1
0.7	0.3	0.3	50	12	2
0.7	0.3	0.3	100	12	1
0.7	0.3	0.3	200	12	1
0.7	0.3	0.4	50	12	2
0.7	0.3	0.4	100	12	1
0.7	0.3	0.4	200	12	0
0.7	0.3	0.5	50	12	2
0.7	0.3	0.5	100	12	1
0.7	0.3	0.5	200	12	1
0.7	0.3	0.6	50	12	3
0.7	0.3	0.6	100	12	2
0.7	0.3	0.6	200	12	1
0.7	0.3	0.7	50	12	2
0.7	0.3	0.7	100	12	2
0.7	0.3	0.7	200	12	1
0.7	0.3	0.8	50	12	2
0.7	0.3	0.8	100	12	1
0.7	0.3	0.8	200	12	0
0.8	0.2	0.1	50	12	1
0.8	0.2	0.1	100	12	2
0.8	0.2	0.1	200	12	1
0.8	0.2	0.2	50	12	2
0.8	0.2	0.2	100	12	2
0.8	0.2	0.2	200	12	1
0.8	0.2	0.3	50	12	1
0.8	0.2	0.3	100	12	2
0.8	0.2	0.3	200	12	1
0.8	0.2	0.4	50	12	1
0.8	0.2	0.4	100	12	2
0.8	0.2	0.4	200	12	1
0.8	0.2	0.5	50	12	2
0.8	0.2	0.5	100	12	1
0.8	0.2	0.5	200	12	0

Продолжение таблицы А.1

α	β	ρ	Дней	Результат	Ошибка
0.8	0.2	0.6	50	12	2
0.8	0.2	0.6	100	12	1
0.8	0.2	0.6	200	12	0
0.8	0.2	0.7	50	12	2
0.8	0.2	0.7	100	12	2
0.8	0.2	0.7	200	12	2
0.8	0.2	0.8	50	12	0
0.8	0.2	0.8	100	12	1
0.8	0.2	0.8	200	12	1
0.9	0.1	0.1	50	12	2
0.9	0.1	0.1	100	12	1
0.9	0.1	0.1	200	12	2
0.9	0.1	0.2	50	12	2
0.9	0.1	0.2	100	12	1
0.9	0.1	0.2	200	12	0
0.9	0.1	0.3	50	12	3
0.9	0.1	0.3	100	12	1
0.9	0.1	0.3	200	12	1
0.9	0.1	0.4	50	12	3
0.9	0.1	0.4	100	12	1
0.9	0.1	0.4	200	12	2
0.9	0.1	0.5	50	12	2
0.9	0.1	0.5	100	12	2
0.9	0.1	0.5	200	12	2
0.9	0.1	0.6	50	12	2
0.9	0.1	0.6	100	12	2
0.9	0.1	0.6	200	12	1
0.9	0.1	0.7	50	12	2
0.9	0.1	0.7	100	12	1
0.9	0.1	0.7	200	12	1
0.9	0.1	0.8	50	12	2
0.9	0.1	0.8	100	12	2
0.9	0.1	0.8	200	12	1