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 коммивояжёра»

Студент ИУ7-51Б (Группа) _____ (Подпись, дата) А. А. Кузин (И. О. Фамилия)

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

2023 г.

Содержание

ВВЕДЕНИЕ

Цель лабораторной работы – реализация методов решения задачи коммивояжера на основе муравьиного алгоритма.

Задачи данной лабораторной:

— описать задачу коммивояжера;

— описать методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма;

— реализовать выбранные алгоритмы;

— провести функциональное тестирование разработанных алгоритмов;

— исследовать время работы реализаций алгоритмов.

1 Аналитическая часть

В этом разделе будут рассмотрена информация, касающаяся задачи коммивояжера, а также способы их решения.

1.1 Задача коммивояжера

Задача коммивояжёра (англ. *traveling salesman problem*) — (задача о бродячем торговце) одна из самых важных задач всей транспортной логистики, в которой рассматриваются вершины графа, а также матрица смежности (для расстояния между вершинами) [1]. Задача заключается в том, чтобы найти такой порядок посещения вершин графа, при котором путь будет минимален, каждая вершина будет посещена лишь один раз, а возврат произойдет в начальную вершину.

1.2 Метод полного перебора для решения задачи коммивояжёра

Полный перебор для задачи коммивояжёра [2] имеет высокую сложность алгоритма ($n!$), где n — количество городов. Суть в полном переборе всех возможных путей в графе и выбор наименьшего из них. Решение будет получено, но имеются большие затраты по времени выполнения при уже небольшом количестве вершин в графе.

1.3 Метод на основе муравьиного алгоритма

Муравьиный алгоритм (англ. *ant colony optimization*) [2] — метод решения задачи оптимизации, основаный на принципе поведения колонии муравьев.

Муравьи действуют, руководствуясь органами чувств. Каждый муравей оставляет на своём пути феромоны, чтобы другие могли ориентироваться. При большом количестве муравьев наибольшее количество феромона остаётся на наиболее посещаемом пути, посещаемость же может быть связана с длинами рёбер.

Суть в том, что отдельно взятый муравей мало что может, поскольку он способен выполнять только максимально простые задачи. Но при большом числе других таких муравьев они могут выступать самостоятельными вычислительными единицами. Муравьи используют непрямой обмен информацией

через окружающую среду посредством феромона.

Пусть муравей имеет следующие характеристики:

1. зрение — способность определить длину ребра;
2. память — способность запомнить пройденный маршрут;
3. обоняние — способность чуять феромон.

Также введем целевую функцию (1.1), характеризующую привлекательность ребра, определяемую благодаря зрению.

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

где D_{ij} — расстояние от текущего пункта i до заданного пункта j.

Также понадобится формула вычисления вероятности перехода в заданную точку (1.2).

$$p_{k,ij} = \begin{cases} \frac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\sum_{q \notin J_k} \eta_{iq}^{\alpha} \cdot \tau_{iq}^{\beta}}, j \notin J_k \\ 0, j \in J_k \end{cases} \tag{1.2}$$

где a — параметр влияния длины пути, b — параметр влияния феромона, τ_{ij} — количество феромонов на ребре ij, η_{ij} — привлекательность ребра ij, J_k — список посещённых за текущий день городов.

После завершения движения всех муравьев (ночью, перед наступлением следующего дня), феромон обновляется по формуле (1.3).

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

При этом

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

где

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q/L_k, \text{ребро посещено муравьем } k \text{ в текущий день } t, \\ 0, \text{иначе} \end{cases} \tag{1.5}$$

Поскольку вероятность перехода в заданную точку 1.2 не должна быть

равна нулю, необходимо обеспечить неравенство $\tau_{ij}(t)$ нулю посредством введения дополительного минимально возможного значения феромона τ_{min} и в случае, если $\tau_{ij}(t+1)$ принимает значение, меньшее τ_{min}, откатывать феромон до этой величины.

Путь выбирается по следующей схеме.

1) Каждый муравей имеет список запретов — список уже посещенных городов (вершин графа).

2) Муравьиное зрение отвечает за эвристическое желание посетить вершину.

3) Муравьиное обоняние отвечает за ощущение феромона на определенном пути (ребре). При этом количество феромона на пути (ребре) в день t обозначается как $\tau_{i,j}(t)$.

4) После прохождения определенного ребра муравей откладывает на нем некотрое количество феромона, которое показывает оптимальность сделанного выбора, это количество вычисляется по формуле (1.5).

Вывод

В этом разделе была рассмотрена задача коммивояжёра, а также полный перебор для её решения и муравьиный алгоритм.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы конвейерной и линейной реализаций алгоритмов обработки матриц.

К программе предъявлен ряд функциональных требований:

— наличие интерфейса для выбора действий;

— возможность выбора линейной или конвейерной реализации алгоритма;

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— размер матрицы смежности — целое число;

— имя файла — строка;

— коэффициенты α, β, $k\_evaporation$ — действительные числа;

— матрица смежности — матрица целых чисел.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма полного перебора путей, а на рисунках 2.2 схема муравьиного алгоритма поиска путей. Также на рисунках 2.3–2.4 представлены схемы вспомогательных функций для муравьиного алгоритма.

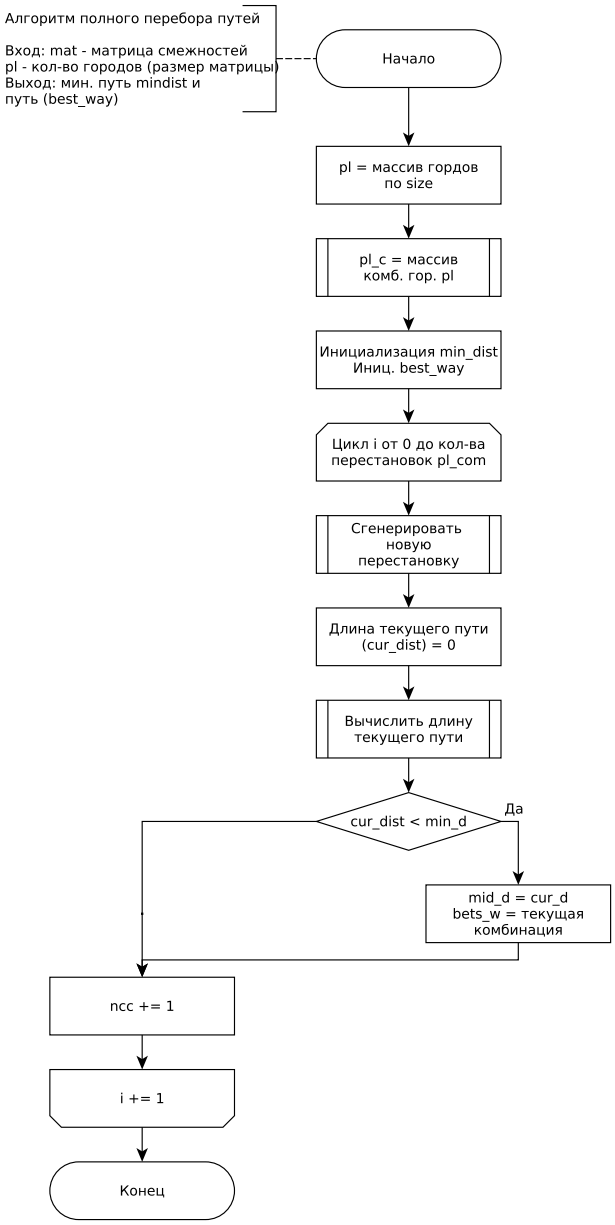

Рисунок 2.1 – Схема алгоритма полного перебора путей

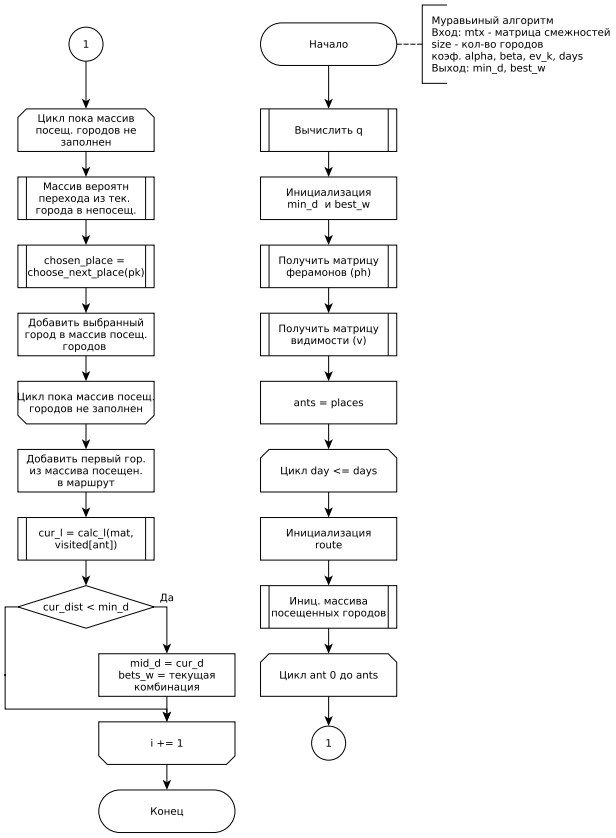

Рисунок 2.2 – Схема муравьиного алгоритма

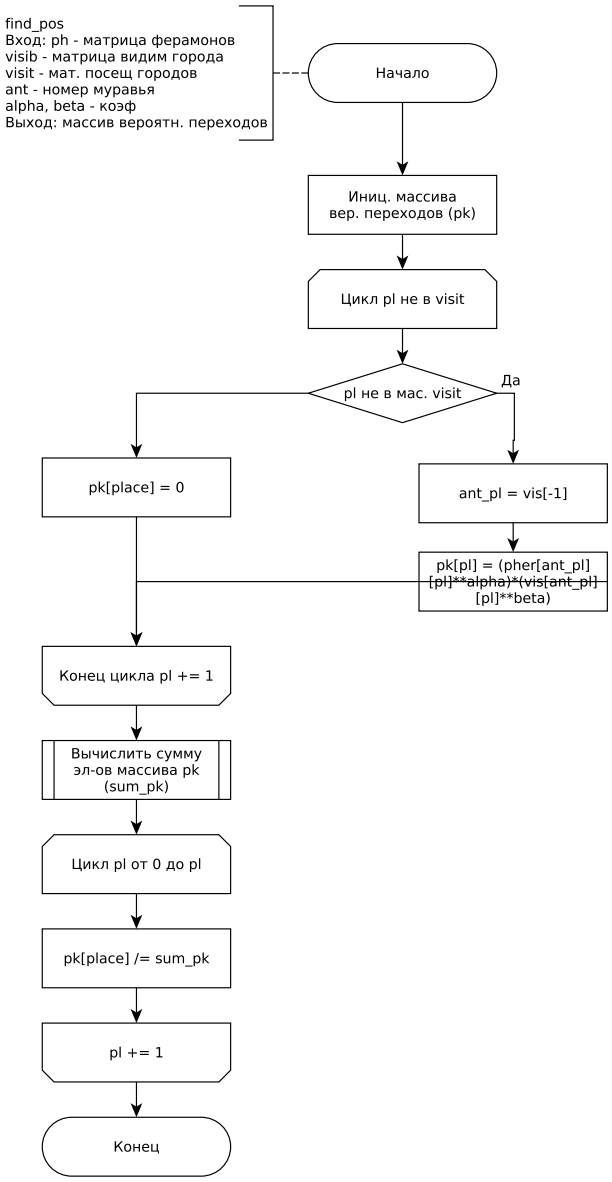

Рисунок 2.3 – Схема нахождения массива вероятностей переходов

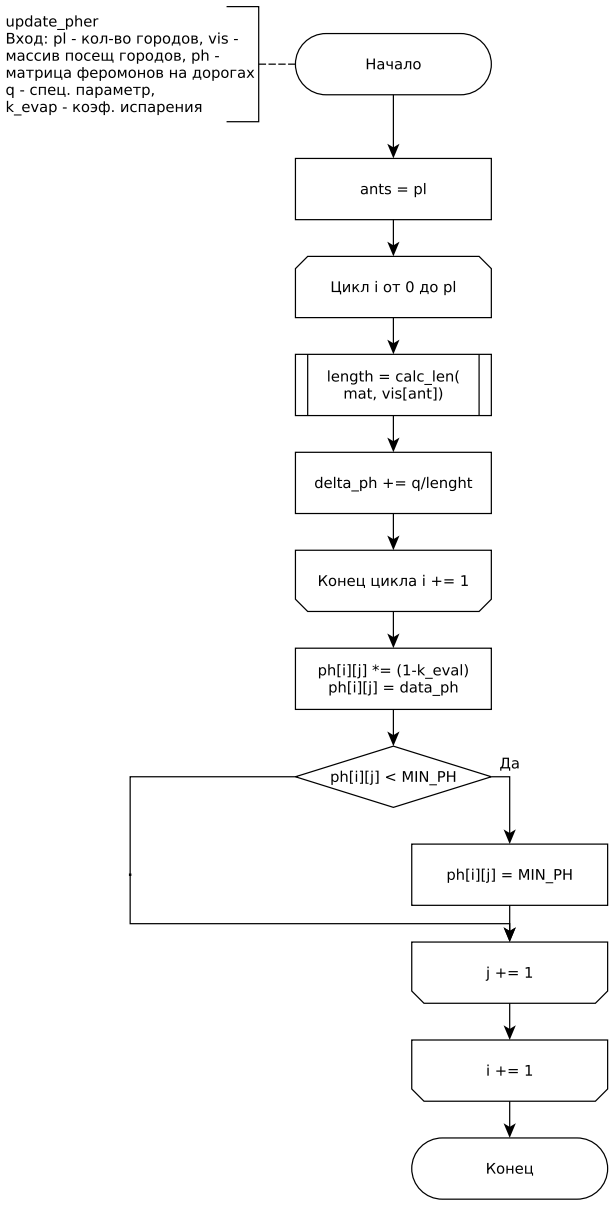

Рисунок 2.4 – Схема обновления матрицы феромонов

Вывод

В данном разделе были построены схемы алгоритмов требуемых методов решения задачи коммивояжёра.

3 Технологическая часть

В данном разделе будут приведены средства реализации, листинг кода и функциональные тесты.

3.1 Требования к программе

Программное обеспечение должно удовлетворять следующим требованиям:

— на вход подается имя файла;

— программа позволяет определять коэффициенты и количество дней;

— возможно измерение реального времени.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования Python[3].

В данном языке есть все требующиеся инструменты для данной лабораторной работы.

В качестве среды разработки была выбрана среда VS Code[4], запуск происходил через команду python main.py.

3.3 Средства замера времени

Замеры времени выполнения реализаций алгоритма будут проводиться при помощи функции process_time [5] библиотеки time. Данная команда возвращает значения процессорного времени типа int в наносекундах.

Замеры времени для каждой реализации алгоритма и для каждого комплекта входных данных проводились 100 раз.

Листинг 3.1 – Пример замера затраченного времени

```
def test_simple_mult(A, B):
# Start the stopwatch / counter
t1_start = process_time()
for i in range(N_TEST):
simple_mult(A, M, B, N, M)
# Stop the stopwatch / counter
t1_stop = process_time()
```

3.4 Реализации алгоритма

В листинге 3.2 представлен алгоритм полного перебора путей, а в листингах 3.3–3.7 — муравьиный алгоритм и дополнительные к нему функции.

Листинг 3.2 – Реализация алгоритма полного перебора путей

```
def fullCombinationAlg(matrix, size):
  places = np.arange(size)
  placesCombinations = list()

  for combination in it.permutations(places):
  combArr = list(combination)
  placesCombinations.append(combArr)
  minDist = float("inf")
  for i in range(len(placesCombinations)):
    placesCombinations[i].append(\
    placesCombinations[i][0])
    curDist = 0
    for j in range(size):
      startCity = placesCombinations[i][j]
      endCity = placesCombinations[i][j + 1]
      curDist += matrix[startCity][endCity]
    if (curDist < minDist):
      minDist = curDist
  bestWay = placesCombinations[i]
  return minDist, bestWay
```

Листинг 3.3 – Реализация муравьиного алгоритма

```
def antAlgorithm(matrix, places, alpha, beta, k_evaporation,
   days):
  q = calcQ(matrix, places)
```

```
3       bestWay = []
4       minDist = float("inf")
5       pheromones = calcPheromones(places)
6       visibility = calcVisibility(matrix, places)
7       ants = places
8       for day in range(days):
9       route = np.arange(places)
10      visited = calcVisitedPlaces(route, ants)
11      for ant in range(ants):
12      while (len(visited[ant]) != ants):
13      pk = findWays(pheromones, visibility, visited, places, ant,
            alpha, beta)
14      chosenPlace = chooseNextPlaceByPosibility(pk)
15      visited[ant].append(chosenPlace - 1)
16
17      visited[ant].append(visited[ant][0])
18
19      curLength = calcLength(matrix, visited[ant])
20
21      if (curLength < minDist):
22      minDist = curLength
23      bestWay = visited[ant]
24
25      pheromones = updatePheromones(matrix, places, visited,
            pheromones, q, k_evaporation)
26
27      return minDist, bestWay
```

Листинг 3.4 – Реализация алгоритма нахождения массива вероятностей переходов в непосещенные города

```
1   def findWays(pheromones, visibility, visited, places, ant, alpha,
        beta):
2       pk = [0] * places
3
4       for place in range(places):
5       if place not in visited[ant]:
6       ant_place = visited[ant][-1]
7       pk[place] = pow(pheromones[ant_place][place], alpha) * \
8       pow(visibility[ant_place][place], beta)
9       else:
10      pk[place] = 0
11
12      sum_pk = sum(pk)
13
14      for place in range(places):
15      pk[place] /= sum_pk
16
17      return pk
```

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

```
1       def calcPheromones(size):
2       min_phero = 1
3       pheromones = [[min_phero for i in range(size)] for j in
            range(size)]
4       return pheromones
```

Листинг 3.6 – Реализация алгоритма выбора следующего города

```
1   def chooseNextPlaceByPosibility(pk):
2     posibility = random()
3     choice = 0
4     chosenPlace = 0
5
6     while ((choice < posibility) and (chosenPlace < len(pk))):
7     choice += pk[chosenPlace]
8     chosenPlace += 1
9
10    return chosenPlace
```

Листинг 3.7 – Реализация алгоритма обновления матрицы феромонов

```
1   def updatePheromones(matrix, places, visited, pheromones, q,
      k_evaporation):
2     ants = places
3
4     for i in range(places):
5     for j in range(places):
6     delta = 0
7     for ant in range(ants):
8     length = calcLength(matrix, visited[ant])
9     delta += q / length
10
11    pheromones[i][j] *= (1 - k_evaporation)
12    pheromones[i][j] += delta
13    if (pheromones[i][j] < MIN_PHEROMONE):
14    pheromones[i][j] = MIN_PHEROMONE
15
16    return pheromones
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Применена методология черного ящика. Тесты для всех сортировок пройдены *успешно*.

Таблица 3.1 – Функциональные тесты

Матрица смежности	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]

Вывод

Были выбраны язык программирования и среда разработки, приведены сведения о модулях программы, листинги алгоритма, проведено функциональное тестирование.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Intel Core i7 9750H 2.6 ГГц;

— Оперативная память: 16 ГБ;

— Операционная система: Kubuntu 22.04.3 LTS x86_64 Kernel: 6.2.0-36-generic

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 показан пример работы с программой.

```
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 файлы:  6  штук
1. mat9_highdif.csv
2. mat9_lowdif.csv
3. ports.csv
4. portsname
5. ports.xlsx
6. test1.csv

Выберите файл: 3


Введите коэффициент alpha: 0.1
Введите коэффициент evaporation: 0.3
Введите кол-во дней: 10


Минимальная сумма пути =  18403
Путь:  [0, 6, 3, 2, 4, 1, 5, 7, 0]
```

Рисунок 4.1 – Демонстрация работы программы.

4.3 Временные характеристики

Для замера процессорного времени используется функция *process_ time()* из библиотеки *time* на *Python*. Функция возвращает процессорное время типа float в секундах.

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

Замеры проводились для разного размера матриц, чтобы определить, когда наиболее эффективно использовать муравьиный алгоритм.

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

Таблица 4.1 – Результаты замеров времени

Размер	Полный перебор	Муравьиный
2	0.000130	0.019932
3	0.000138	0.031615
4	0.000104	0.044361
5	0.000420	0.089291
6	0.002390	0.152131
7	0.019703	0.254059
8	0.162850	0.398472
9	1.637611	0.594024
10	18.207853	0.857666

Также на рисунке 4.2 приведены графические результаты замеров.

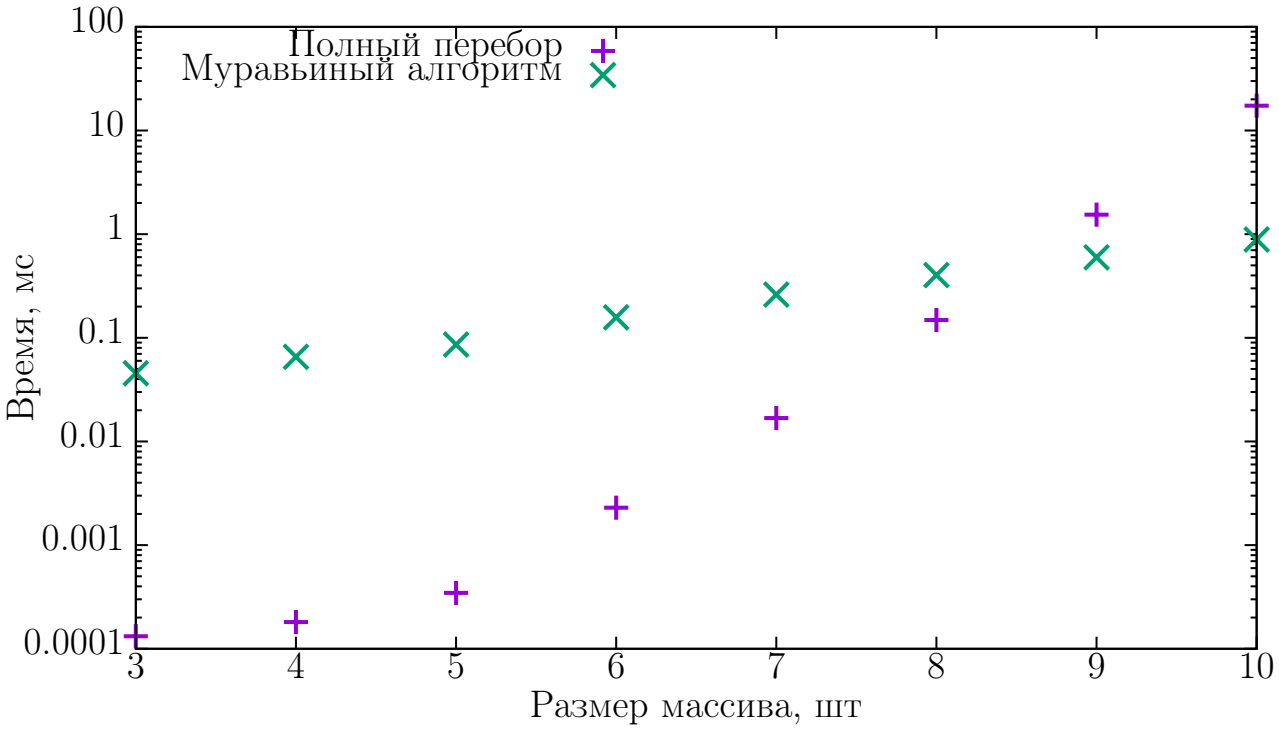

Рисунок 4.2 – Сравнение по времени алгоритмов полного перебора путей и муравьиного на разных размерах матриц

4.4 Постановка исследования

Автоматическая параметризация была проведена на двух классах данных — 4.4.1 и 4.4.2. 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 основаным на муравьином алгоритме методом значения и эталонного значения на данных значениях параметров, показатель качества решения.

Цель исследования — определить комбинацию параметров, которые позволяют решаить 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 9 элементов (небольшой разброс значений — от 1 до 2), которая представлена далее.

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

Для данного класса данных приведена таблица 4.2 с выборкой параметров, которые наилучшим образом решают поставленную задачу, полные результаты параметризации приведены в приложении А. 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.

$$K_1 = \begin{pmatrix} 0 & 9271 & 8511 & 2010 & 1983 & 7296 & 7289 & 3024 & 1011 \\ 9271 & 0 & 7731 & 4865 & 5494 & 6812 & 4755 & 7780 & 7641 \\ 8511 & 7731 & 0 & 1515 & 9297 & 7506 & 5781 & 5804 & 7334 \\ 2010 & 4865 & 1515 & 0 & 3662 & 9597 & 2876 & 8188 & 9227 \\ 1983 & 5494 & 9297 & 3662 & 0 & 8700 & 4754 & 7445 & 3834 \\ 7296 & 6812 & 7506 & 9597 & 8700 & 0 & 4216 & 5553 & 8215 \\ 7289 & 4755 & 5781 & 2876 & 4754 & 4216 & 0 & 4001 & 4715 \\ 3024 & 7780 & 5804 & 8188 & 7445 & 5553 & 4001 & 0 & 9522 \\ 1011 & 7641 & 7334 & 9227 & 3834 & 8215 & 4715 & 9522 & 0 \end{pmatrix} \quad (4.2)$$

Для данного класса данных приведена таблица с выборкой параметров, которые наилучшим образом решают поставленную задачу Days — количество дней, Result — результат работы, Mistake — ошибочность полученного результата).

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 смежности расстояний между следующими портами: Новороссийск, Усть-Луга, Восточный, Приморск, Санкт-Петербург, Мурманск, Порт-Кавказ, Ванино[6].

$$K_3 = \begin{pmatrix} 0 & 4525 & 5112 & 160 & 4581 & 5074 & 513 & 9686 \\ 4525 & 0 & 417 & 4682 & 67 & 2092 & 4479 & 12840 \\ 5112 & 417 & 0 & 613 & 349 & 2492 & 5095 & 13250 \\ 160 & 4682 & 613 & 0 & 4729 & 5221 & 84 & 9826 \\ 4581 & 67 & 349 & 4729 & 0 & 2083 & 4745 & 12900 \\ 5074 & 2092 & 2492 & 5221 & 2083 & 0 & 4978 & 4999 \\ 513 & 4479 & 5095 & 84 & 4745 & 4978 & 0 & 9734 \\ 9686 & 12840 & 13250 & 9826 & 12900 & 4999 & 9734 & 0 \end{pmatrix} \quad (4.3)$$

Таблица 4.4 – Параметры для класса данных 3

α	ρ	Days	Result	Mistake
0.1	0.3	100	18403	0
0.1	0.3	300	18403	0
0.1	0.3	500	18403	0
0.1	0.7	100	18403	0
0.1	0.7	300	18403	0
0.1	0.7	500	18403	0
0.2	0.1	100	18403	0
0.2	0.1	300	18403	0
0.2	0.1	500	18403	0
0.2	0.2	100	18403	0
0.2	0.2	300	18403	0
0.2	0.2	500	18403	0
0.2	0.5	100	18403	0
0.2	0.5	300	18403	0
0.2	0.5	500	18403	0
0.2	0.8	100	18403	0
0.2	0.8	300	18403	0
0.2	0.8	500	18403	0
0.3	0.1	100	18403	0
0.3	0.1	300	18403	0

0.3	0.1	500	18403	0
0.3	0.2	5	18403	0
0.3	0.2	50	18403	0
0.3	0.2	100	18403	0
0.3	0.2	300	18403	0
0.3	0.2	500	18403	0
0.4	0.5	50	18403	0
0.4	0.5	300	18403	0
0.4	0.5	500	18403	0
0.5	0.2	100	18403	0
0.5	0.2	300	18403	0
0.5	0.2	500	18403	0
0.6	0.2	100	18403	0
0.6	0.2	300	18403	0
0.6	0.2	500	18403	0
0.6	0.3	300	18403	0
0.6	0.3	500	18403	0
0.6	0.4	100	18403	0
0.6	0.4	500	18403	0
0.6	0.5	100	18403	0
0.6	0.5	300	18403	0
0.6	0.5	500	18403	0

Вывод

В результате исследования было получно, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так, при размере матрицы, равном 2, муравьиный алгоритм медленее алгоритма полного перебора в 153 раза, а при размере матрицы, равном 9, муравьиный алгоритм быстрее алгоритма полного перебора в раз, а при размере в 10 – уже в 21 раз. Следовательно, при рамзерах матриц больше 8 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.

Также во время исследования было замечено — чем меньше α, тем меньше погрешностей возникает. При этом число дней жизни колонии значительно влияет на качество решения: чем значение параметра $Days$ больше, тем меньше отклонение решения от эталонного.

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы поставленная ранее цель была достигнута: реализованы методы решения задачи коммивояжера на основе муравьиного алгоритма

Исходя из полученных результатов, использование муравьиного алгоритма наиболее эффективно по времени при больших размерах матриц. Так при размере матрицы, равном 2, муравьиный алгоритм медленее алгоритма полного перебора в 153 раза, а при размере матрицы, равном 9, муравьиный алгоритм быстрее алгоритма полного перебора в раз, а при размере в 10 – уже в 21 раз. Следовательно, при размерах матриц больше 8 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует оптимального решения, в отличие от метода полного перебора.

В ходе выполнения лабораторной работы были решены следующие задачи:

— описана задача коммивояжера;

— описаны методы решения задачи коммивояжера;

— реализованы выбранные алгоритмы;

— проведено функциональное тестирование разработанных алгоритмов;

— исследовано время работы реализации алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исследование решения задачи коммивояжера [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/issledovanie-resheniya-zadachi-kommivoyazhera/viewer (дата обращения: 15.12.2023).
2. Анализ трудоёмкости различных алгоритмических подходов для решения задачи коммивояжёра [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/analiz-trudoemkosti-razlichnyh-algoritmicheskih-podhodov-dlya-resheniya-zadachi-kommivoyazhera (дата обращения: 15.12.2023).
3. Welcome to Python [Электронный ресурс]. — Режим доступа: https://www.python.org (дата обращения: 11.10.2023).
4. Vscode Документация[Электронный ресурс]. — Режим доступа: https://code.visualstudio.com/docs (дата обращения: 24.09.2023).
5. Standard library header <ctime> [Электронный ресурс]. — — Режим доступа: https://en.cppreference.com/w/cpp/header/ctime.
6. Cargo Calculator [Электронный ресурс]. — Режим доступа: https://www.searates.com/ru/services/distances-time/ (дата обращения: 15.12.2023).