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»

Лабораторная работа №6
по дисциплине «Анализ Алгоритмов»

Тема Методы решения задачи коммивояжера

Студент Пермякова Е. Д.

Группа ИУ7-52Б

Преподаватели Строганов Д. В., Волкова Л. Л

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является сравнительный анализ методов решения задачи коммивояжера.

Для достижения поставленной цели требуется решить следующие задачи:

1) описать методы решения задачи коммивояжера основанные на полном переборе и на муравьином алгоритме;

2) выделить преимущества и недостатки рассматриваемых методов;

3) описать схему алгоритмов решения задачи коммивояжера;

4) реализовать алгоритмы решения задачи коммивояжера;

5) выполнить параметризацию метода основанного на муравьином алгоритме;

6) провести сравнительный анализ методов решения задачи коммивояжера.

1 Аналитическая часть

В данном разделе рассматриваются методы решения задачи коммивояжера основанные на полном переборе и на муравьином алгоритме.

1.1 Задача коммивояжера

Задача коммивояжера: на основе заданного графа и информации о расстоянии между его вершинами составить такой путь/цепь (далее «маршрут»), чтобы все узлы входили в него по 1 разу и при этом сумма меток дуг/ребер, соединяющих каждую последовательну пару вершин маршрута, была минимальна. Также возможна другая постановка задачи с возвратом в начальные город – поиск гамельтонова пути.

1.2 Метод полного перебора

Метод полного перебора заключается в том, чтобы рассмотреть все возможные варианты маршрута в рассматриваемом графе. Главным недостатком алгоритма является его ассимптотическая сложность, которая равна (O(n!)), где n – это количество узлов в графе. Достоинством данного решения является гарантированное нахождение наилучшего маршрута.

1.3 Метод на основе муравьиного алгоритма

Муравьиный алгоритм [1] заключается в поиске решения, основанном на поведении муравьев. Главным достоинством данного алгоритма является более низкая временная сложность, относительно алгоритма основанного на полном переборе. Но для данного метода не гарантированно нахождение наилучшего результа.

За каждые сутки $t \in [1, t_{max}]$ каждый муравей $k \in [1, n]$, где n – это количество узлов в графе, формирует 1 маршрут, который претендует на решение задачи коммивояжера.

Муравей обладает следующими свойствами:

— **зрение** – муравей k стоя в вершине i может оценить привлекательность перехода η_{ij} из города i в город j

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

где D_{ij} — расстояние между вершинами i и j;

— **память** – муравей k запоминает посещение в день t города в кортеж посещенных городов $I_k(t)$;

— **обоняние** – муравей чует концентрацию феромона $\tau_{ij}(t)$ на дуге ij;

Общий вид алгоритма:
Для каждого $t \in [1, t_{max}]$

1) утром муравьи выходят из муравейника, каждый муравей встает в свою вершину (муравьи распределяются по одному в каждом узле графа);

2) за день t каждый муравей k формирует по одному маршруту;

3) к закату муравь возвращаются в муравейник и при необходимости обновляется наилучший из найденных маршрутов;

4) ночью обновляется феромон;

Стоя в день t в вершине i муравей k рассчитывает вероятность перехода в вершину j

$$P_{kij} = \begin{cases} \frac{\eta_{ij}^a \tau_{ij}^{1-a}}{\sum_{q=1}^{n} \eta_{iq}^a \tau_{iq}^{1-a}}, j \notin I_k(t) \\ 0, j \in I_k(t) \end{cases} \tag{1.2}$$

где a – коэффицент жадности решения.

Формула обновления феромона:

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

где $\in (0, 1)$ – коэффицент испарения феромона.

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

$$\Delta\tau_{k,ij}(t) = \begin{cases} \frac{Q}{L_k(t)}, \text{если муравей } k \text{ в день } t \text{ ходил по ребру } ij \\ 0, \text{иначе} \end{cases} \tag{1.5}$$

где $L_k(t)$ – длина маршрута муравья k в день t, Q – дневная квота феромона каждого муравья.

При этом, если $\tau_{ij}(t+1)$ стал меньше некоторой малой константы, то его значение откатывается дл этой малой константы.

1.4 Модмификация с элитными муравьями

Элитный муравей усиливает феромон дополнительной квотой дуги наилучшего маршрута длиной L_{best}

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{N} \Delta\tau_{k,ij}(t) + \Delta\tau(t), \tag{1.6}$$

$$\Delta\tau(t) = \begin{cases} \frac{Q}{L_{best}}, \text{если дуга входит в наилучший маршрут} \\ 0, \text{иначе} \end{cases} \tag{1.7}$$

Вывод

В данном разделе была рассмотрена задача коммивояжера и методы ее решения на основе полного перебора и муравьиного алгоритма.

2 Конструкторская часть

В этом разделе представлены схемы муравьиного алгоритма и алгоритма основанного на полном переборе.

2.1 Описание алгоритмов

В данной работе рассматривается поставка задачи коммивояжера с использованием неориентированного графа, описывающего расстояния между городами России от Калининграда до Владивостока, и производится поиск гамильтонова цикла.

На рисунках 2.1-2.3 представлены схемы алгоритма основанного на полном переборе и муравьиного алгоритма.

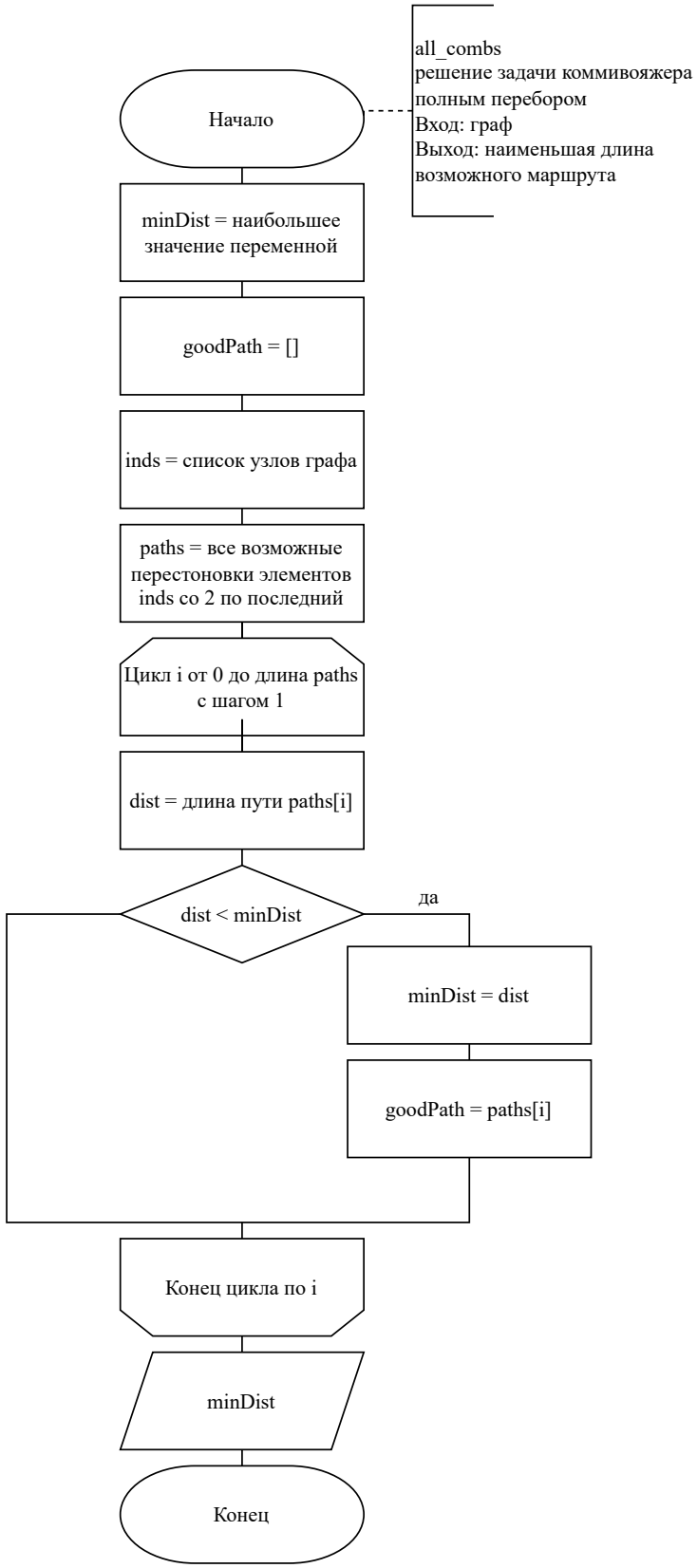

Рисунок 2.1 – Алгоритм основанный на полном переборе

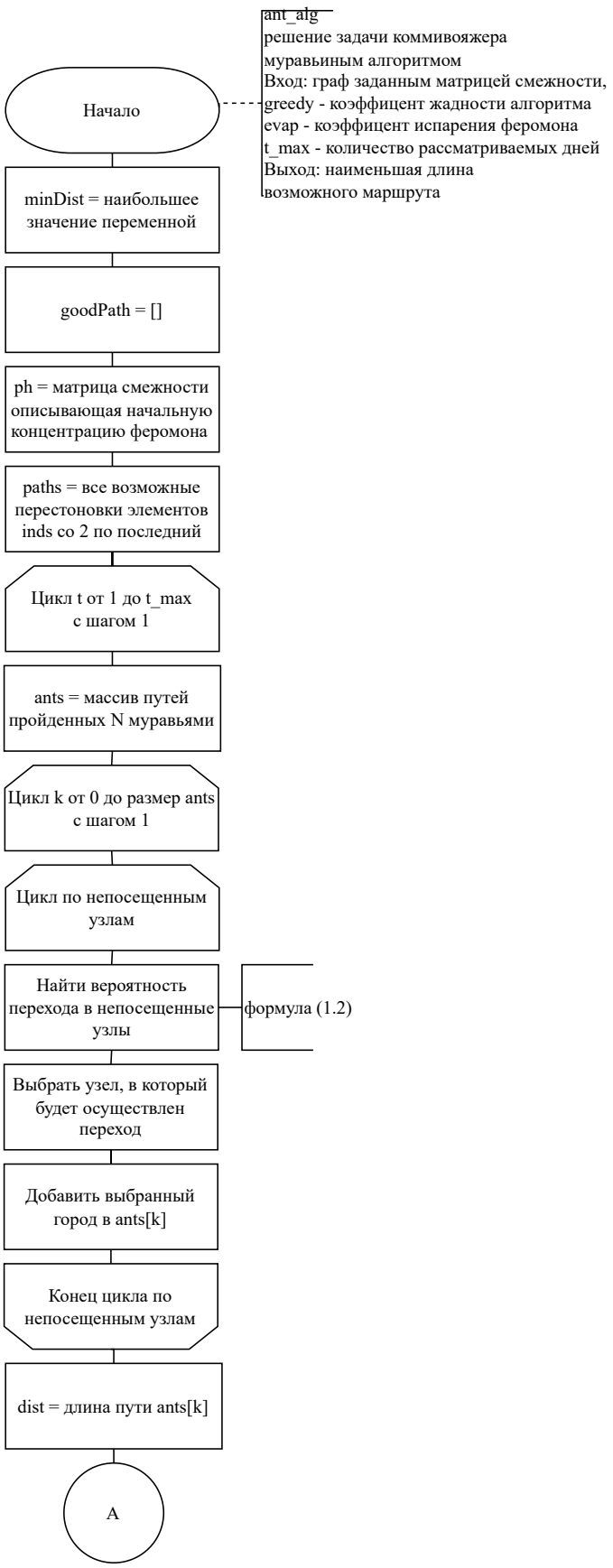

Рисунок 2.2 – Муравьиный алгоритм (начало)

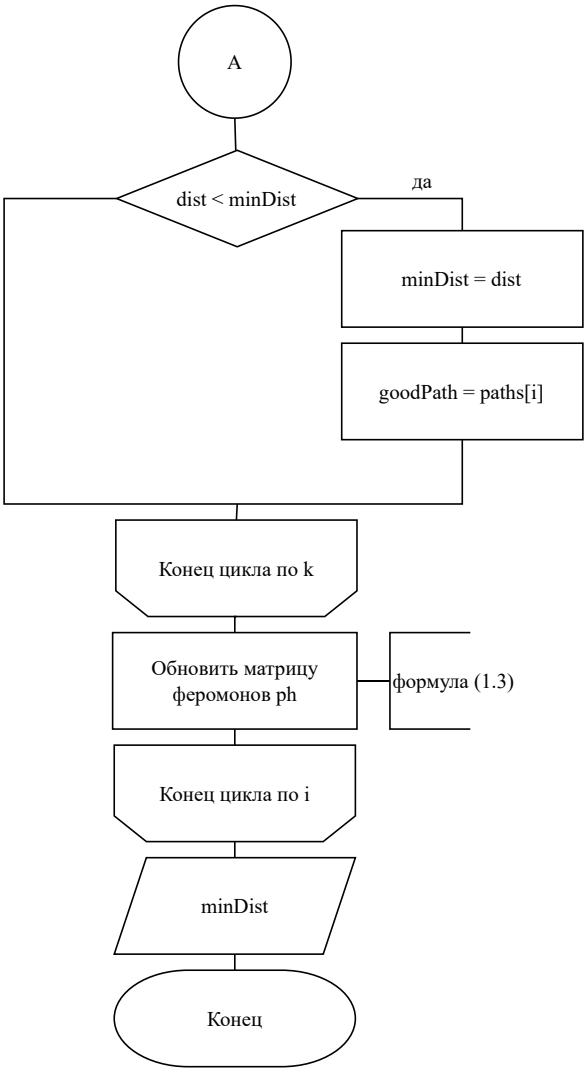

Рисунок 2.3 – Муравьиный алгоритм (конец)

Вывод

В данном разделе были представлены схемы муравьиного алгоритма и алгоритма основанного на полном переборе.

3 Технологическая часть

В данном разделе представлены средства реализации, листинги муравьиного алгоритма, алгоритма основанного на полном переборе и функциональные тесты.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования $Python$ [2], так как он удовлетворяет требованиям лабраторной работы: может замерить процессорное время (с помошью функции $process\_time\_ns(...)$ из библиотеки $time$ [3])

В данной работе для реализации был выбран язык программирования $Python$ [2], так как он удовлетворяет требованиям лабраторной работы: поддерживает динамические структуры данных, такие как массивы, имеет библиотеки $GeoPy$ [4] для вычисления расстояния городами и $Matplotlib$ [5] для построения графиков и может замерить процессорное время (с помошью функции $process\_time\_ns(...)$ из библиотеки $time$ [3])

3.2 Реализация алгоритмов

В листингах 3.1-3.2 представлены реализации алгоритма основанного на полном переборе и муравьиного алгоритма.

Листинг 3.1 – Реализация алгоритма основанного на полном переборе

```
def all_combs(graph : CitiesMap):
  lenpath = len(graph)
  inds = [i for i in range(1, lenpath)]

  minDist = float('inf')
  goodPath = []
  for path in permutations(inds):
    path = [0] + list(path)
    dist = 0
    for i in range(lenpath - 1):
      dist += graph[path[i]][path[i + 1]]
    dist += graph[path[0]][path[-1]]

    if dist < minDist:
```

```
      minDist = dist
      goodPath = path[:]

  return (minDist, goodPath)
```

Листинг 3.2 – Реализация муравьиного алгоритма

```
class Ant:
  def __init__(self, istart, cmap: CitiesMap, pheromones, greedy)
     :
    self.path = self.generatePath(istart, cmap, pheromones,
       greedy)
    self.lenPath = self.calcLenPath(cmap, self.path)

  def getPath(self):
    return self.path[:]

  def getLenPath(self):
    return self.lenPath

  def generatePath(self, istart, cmap: CitiesMap, pheromones,
     greedy, printing=False):
    path = [istart]
    ncities = len(cmap)
    while len(path) != ncities:
      probPaths = [0] * ncities
      for i in range(ncities):
        if i in path:
          probPaths[i] = 0
        else:
          ist = path[-1]
          probPaths[i] = cmap.attractiveness(ist, i) ** greedy *
             \
            pheromones[ist][i] ** (1 - greedy)
      sumProbPaths = sum(probPaths)
      for i in range(ncities):
        probPaths[i] /= sumProbPaths

      r = random.random()
      last = 0
      lines = []
      for i in range(len(probPaths)):
```

```
        lines.append((last, last + probPaths[i]))
        last = last + probPaths[i]

      i = 0
      while not( lines[i][0] <= r < lines[i][1] ):
        i += 1
      ind = i


      path.append(ind)
    return path

  def calcLenPath(self, cmap, path):
    lenPath = 0
    for i in range(len(cmap) - 1):
      lenPath += cmap[path[i]][path[i + 1]]
    lenPath += cmap[path[-1]][path[0]]
    return lenPath

class Pheromones:
  min_ph = 0.01
  def __init__(self, size):
    self.phmap = []
    for i in range(size):
      self.phmap.append([1 for _ in range(size)])
      self.phmap[i][i] = 0

  def __getitem__(self, item):
    return self.phmap[item]

  def update(self, ants, lenBestPath, cmap: CitiesMap,
     evaporation, printing=False):
    deltaMap = [[0 for i in range(len(cmap))] for _ in range(len(
       cmap))]

    for a in ants:
      path = a.getPath()
      delta = cmap.day_pheromone / a.getLenPath()
      for i in range(len(path) - 1):
        deltaMap[path[i]][path[i + 1]] += self.phmap[path[i]][
           path[i + 1]] * (1 - evaporation) + delta
```

```
        deltaMap[path[i + 1]][path[i]] = deltaMap[path[i]][path[i
            + 1]]


      deltaMap[path[0]][path[-1]] += self.phmap[path[i]][path[i +
          1]] * (1 - evaporation) + delta
      deltaMap[path[-1]][path[0]] = deltaMap[path[0]][path[-1]]

    for i in range(len(cmap)):
      for j in range(i):
        self.phmap[i][j] = max(self.min_ph, deltaMap[i][j])
        self.phmap[j][i] = self.phmap[i][j]

def ant_alg(cmap: CitiesMap, greedy, evaporation, t_max):
  bestPath = []
  lenBestPath = float('inf')
  pheromones = Pheromones(len(cmap))

  for t in range(t_max):
    ants = [Ant(i, cmap, pheromones, greedy) for i in range(len(
       cmap))]
    for a in ants:
      if a.getLenPath() < lenBestPath:
        bestPath = a.getPath()
        lenBestPath = a.getLenPath()
    pheromones.update(ants, lenBestPath, cmap, evaporation)

  return (lenBestPath, bestPath)
```

3.3 Классы эквивалентности тестирования

Для тестирования были выделены следующие классы тестирования:

1) Граф пустой;

2) Граф с 1 вершиной;

3) Граф с 2 вершинами;

4) Граф с 3 вершинами;

3.4 Функциональные тесты

В таблице **??** приведены функциональные тесты для алгоритмов умножения матриц.

Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Граф представленный матрицей смежности	Ожидаемая длина маршрута	Результат программы
$\begin{pmatrix} \end{pmatrix}$	0	0
$\begin{pmatrix} 10 \end{pmatrix}$	0	0
$\begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$	2	2
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix}$	6	6

Вывод

В данном разеде были представлены средства реализации, листинг кода и функциональные тесты.

4 Исследовательская часть

Цель исследования – сравнительный анализ методов решения задачи коммивояжера

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— Операционная система – Майкрософт Windows 11 Домашняя для одного языка; Версия - 10.0.22631;

— Установленная оперативная память (RAM) - 16,0 ГБ;

— Процессор - AMD Ryzen 7 5800H with Radeon Graphics, 3201 МГц, ядер: 8, логических процессоров: 16;

При проведении замеров времени ноутбук был включен в сеть электропитания, а не работал от аккамулятора, и были запущены только встроенное приложение окружения и система замеров времени.

4.2 Время выполнения алгоритмов

Для замера процессорного времени использовалась функция $process\_time\_ns(...)$ из библиотеки $time$ [3]).

Замеры времени проводились по принципу: для одних входных данных проводилось 5 замеров и если относительная стандартная ошибка среднего (rse) была >= 5%, то для этих данных замеры продолжались, в таблицу заносилось среднее значение.

Был получен график зависимости времени работы агоритмов, полного перебора и муравьиного, от количества узлов в графе 4.1.

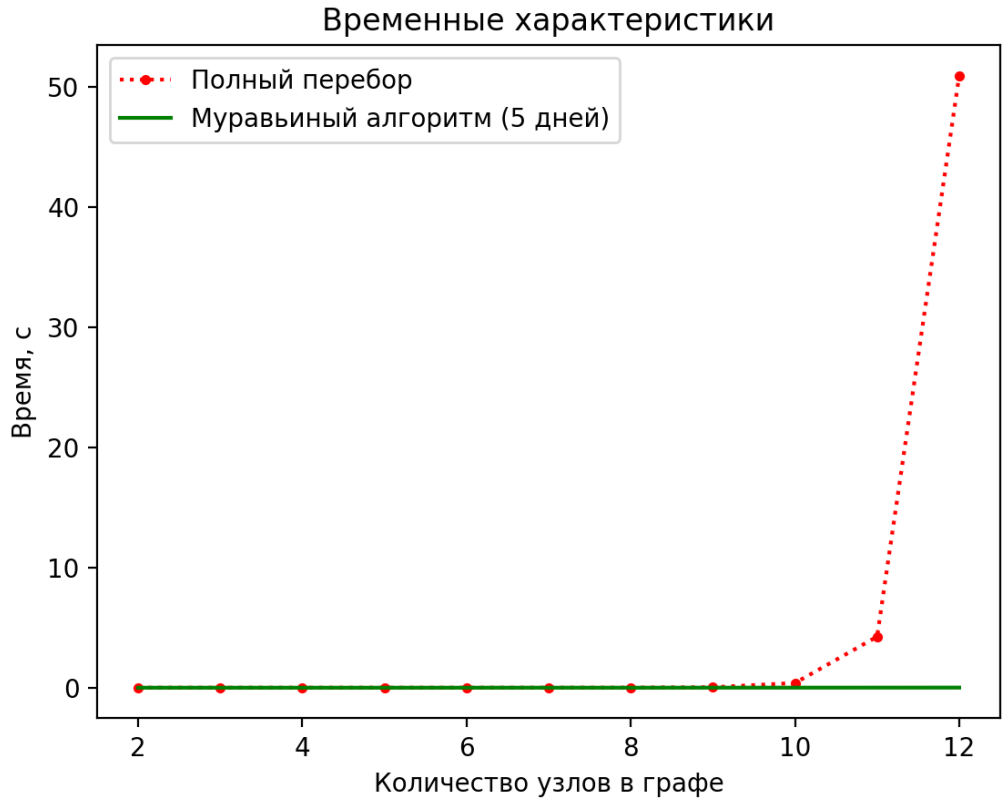

Рисунок 4.1 – Зависимости времени работы агоритмов, полного перебора и муравьиного, от количества узлов в графе

По результатам замеров времени можно сделать вывод о том, что при размерах матрицы больших 10 муравьиный алгоритм работает быстрее, чем алгоритм использующий метод полного перебора.

4.3 Результаты параметризации

Целью параметризации – определение параметры, при которых задача выбранного класса данных решается с наилучшим результатом.

Параметризация муравьиного алгоритма проводилась по параметрам

— коэффицент жадности алгоритма, $a \in 0.1, 0.25, 0.5, 0.75, 0.9$;

— коэффицент испарения феромона, $p \in 0.1, 0.25, 0.5, 0.75, 0.9$;

— количество дней, рассматриваемых в алгоритме, $t_max \in 5, 10, 50, 100, 200$;

Был рассмотрен один класс данных, который состоял из 3 графов 4.1-4.3 представленных матрицами смежности размерностью в 10 элементов. Для

вычисления расстояния между городами использовалась библиотека $GeoPy$ [4].

$$G_1 = \begin{pmatrix} 0 & 2712 & 1382 & 1561 & 1006 & 741 & 5846 & 4112 & 2092 & 1109 \\ 2712 & 0 & 1521 & 4005 & 2947 & 3272 & 3671 & 2835 & 1797 & 1896 \\ 1382 & 1521 & 0 & 2491 & 1426 & 1797 & 5078 & 3774 & 1939 & 1201 \\ 1561 & 4005 & 2491 & 0 & 1091 & 822 & 7378 & 5671 & 3653 & 2664 \\ 1006 & 2947 & 1426 & 1091 & 0 & 654 & 6441 & 4911 & 2901 & 1903 \\ 741 & 3272 & 1797 & 822 & 654 & 0 & 6562 & 4853 & 2832 & 1842 \\ 5846 & 3671 & 5078 & 7378 & 6441 & 6562 & 0 & 2109 & 3825 & 4741 \\ 4112 & 2835 & 3774 & 5671 & 4911 & 4853 & 2109 & 0 & 2025 & 3029 \\ 2092 & 1797 & 1939 & 3653 & 2901 & 2832 & 3825 & 2025 & 0 & 1005 \\ 1109 & 1896 & 1201 & 2664 & 1903 & 1842 & 4741 & 3029 & 1005 & 0 \end{pmatrix} \quad (4.1)$$

Граф 1 (4.1) описывает расстояние в километрах между городами: Барнаул, Екатеринбург, Калининград, Москва, Санкт-Петербург, Владивосток, Якутск, Норильск, Воркута.

$$G_2 = \begin{pmatrix} 0 & 2005 & 6837 & 5312 & 2869 & 5759 & 6999 & 1181 & 1699 & 1993 \\ 2005 & 0 & 5166 & 3804 & 1197 & 3894 & 5468 & 1017 & 1922 & 773 \\ 6837 & 5166 & 0 & 1582 & 4029 & 1544 & 645 & 6168 & 7041 & 4865 \\ 5312 & 3804 & 1582 & 0 & 2614 & 1478 & 1694 & 4759 & 5725 & 3385 \\ 2869 & 1197 & 4029 & 2614 & 0 & 2893 & 4290 & 2151 & 3119 & 881 \\ 5759 & 3894 & 1544 & 1478 & 2893 & 0 & 2109 & 4911 & 5671 & 3774 \\ 6999 & 5468 & 645 & 1694 & 4290 & 2109 & 0 & 6441 & 7378 & 5078 \\ 1181 & 1017 & 6168 & 4759 & 2151 & 4911 & 6441 & 0 & 1091 & 1426 \\ 1699 & 1922 & 7041 & 5725 & 3119 & 5671 & 7378 & 1091 & 0 & 2491 \\ 1993 & 773 & 4865 & 3385 & 881 & 3774 & 5078 & 1426 & 2491 & 0 \end{pmatrix} \quad (4.2)$$

Граф 2 (4.2) описывает расстояние в километрах между городами: Барнаул, Краснодар, Сыктывкар, Хабаровск, Чита, Сургут, Якутск, Владивосток, Москва, Калининград, Екатеринбург.

$$G_3 = \begin{pmatrix} 0 & 851 & 2049 & 2737 & 3537 & 4526 & 4262 & 4710 & 2290 & 5225 \\ 851 & 0 & 1233 & 1957 & 2690 & 3677 & 3411 & 3899 & 3109 & 4376 \\ 2049 & 1233 & 0 & 1477 & 1531 & 2524 & 2320 & 2667 & 4334 & 3313 \\ 2737 & 1957 & 1477 & 0 & 1518 & 2240 & 1857 & 2899 & 4741 & 2664 \\ 3537 & 2690 & 1531 & 1518 & 0 & 994 & 824 & 1392 & 5788 & 1810 \\ 4526 & 3677 & 2524 & 2240 & 994 & 0 & 414 & 972 & 6761 & 922 \\ 4262 & 3411 & 2320 & 1857 & 824 & 414 & 0 & 1332 & 6461 & 993 \\ 4710 & 3899 & 2667 & 2899 & 1392 & 972 & 1332 & 0 & 6999 & 1699 \\ 2290 & 3109 & 4334 & 4741 & 5788 & 6761 & 6461 & 6999 & 0 & 7378 \\ 5225 & 4376 & 3313 & 2664 & 1810 & 922 & 993 & 1699 & 7378 & 0 \end{pmatrix} \quad (4.3)$$

Граф 3 (4.3) описывает расстояние в километрах между городами: Иркутск, Красноярск, Омск, Воркута, Казань, Брянск, Тверь, Краснодар, Владивосток, Калининград.

Таблица значений параметризации представлена в приложении А.

По результатам параметризации было определено несколько наилучших наборов параметров 4.1-4.3:

Таблица 4.1 – Результаты параметризации муравьиного алгоритма для графа 1 (фрагмент)

a	p	t_max	max_diff	avg_diff	mid_diff
0.90	0.50	200	383.00	114.90	0.00
0.90	0.25	200	383.00	38.30	0.00
0.75	0.50	200	1133.00	419.90	191.50
0.75	0.75	200	750.00	381.40	383.00
0.90	0.10	200	383.00	38.30	0.00

Таблица 4.2 – Результаты параметризации муравьиного алгоритма для графа 2 (фрагмент)

a	p	t_max	max_diff	avg_diff	mid_diff
0.90	0.50	200	0.00	0.00	0.00
0.90	0.25	200	383.00	114.90	0.00

Продолжение таблицы 4.2

a	p	t_max	max_diff	avg_diff	mid_diff
0.75	0.50	200	897.00	472.90	383.00
0.75	0.75	200	964.00	337.30	191.50
0.90	0.10	200	636.00	101.90	0.00

Таблица 4.3 – Результаты параметризации муравьиного алгоритма для графа 3 (фрагмент)

a	p	t_max	max_diff	avg_diff	mid_diff
0.90	0.50	200	383.00	114.90	0.00
0.90	0.25	200	383.00	38.30	0.00
0.75	0.50	200	383.00	76.60	0.00
0.75	0.75	200	636.00	255.10	383.00
0.90	0.10	200	750.00	113.30	0.00

4.4 Вывод

В данном разделе было проведено исследование зависимости времени работы муравьиного алгоритма и алгоритма полного перебора, из которого был сделан вывод о том что при размерах матрицы больших 10 муравьиный алгоритм работает быстрее, чем алгоритм использующий метод полного перебора. Также была проведена параметризация, по результатам которой было определено несколько наилучших наборов параметров для описанного класса данных.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была выполнена поставленная цель, которая заключалась в сравнительном анализе методов решения задачи коммивояжера.

В процессе выполнения данной работы были выполнены все задачи:

1) описать методы решения задачи коммивояжера основанные на полном переборе и на муравьином алгоритме;

2) выделить преимущества и недостатки рассматриваемых методов;

3) описать схему алгоритмов решения задачи коммивояжера;

4) реализовать алгоритмы решения задачи коммивояжера;

5) выполнить параметризацию метода основанного на муравьином алгоритме;

6) провести сравнительный анализ методов решения задачи коммивояжера.

Основываясь на проведенном исследовании зависимости времени работы алгоритмов и параметризации муравьиного алгоритма был сделан вывод о том, что при размерах матрицы больших 10 муравьиный алгоритм работает быстрее, также было определено несколько наилучших наборов параметров для описанного класса данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. С.Д. Штовба. Муравьиные алгоритмы. Exponenta Pro. Математика в приложениях.

2. Welcome to Python [Электронный ресурс]. (дата обращения: 22.12.2024).

3. time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions`. (дата обращения: 22.12.2024).

4. GeoPy 2.4.1 documentation [Электронный ресурс]. (дата обращения: 22.12.2024).

5. Matplotlib 3.9.2 documentation [Электронный ресурс]. (дата обращения: 22.12.2024).

Приложение А