Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

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

Тема Методы решения задачи коммивояжёра

Студент Золотухин А.В.

Группа ИУ7-54Б

Оценка (баллы)

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2022 г.

Оглавление

Введение

Задача коммивояжёра — одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город.

Задача коммивояжёра [1] относится к числу транс вычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никакими теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения 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.

Цель работы: изучить муравьиный алгоритм на материале решения задачи коммивояжёра.

Задачи лабораторной работы:

— описать методы решения.

— описать реализацию, реализовать метод.

— выбрать класс данных, составить набор данных.

— провести параметризацию метода на основании муравьиного алгоритма для выбранного класса данных.

— провести сравнительный анализ двух методов.

— дать рекомендации о применимости метода решения задачи коммивояжера на основе муравьиного алгоритма.

1 Аналитическая часть

В данном разделе содержится описание задачи коммивояжера и методы её решения.

1.1 Задача коммивояжера

В общем случае задача коммивояжера может быть сформулирована следующим образом: найти самый выгодный (самый короткий, самый дешевый, и т.д.) маршрут, начинающийся в исходном городе и проходящий ровно один раз через каждый из указанных городов.

Проблему коммивояжера можно представить в виде модели на графе, то есть, используя вершины и ребра между ними. Таким образом, M вершин графа соответствуют M городам, а ребра (i, j) между вершинами i и j — пути сообщения между этими городами. Каждому ребру (i, j) можно сопоставить критерий выгодности маршрута $c_{ij} \geq 0$,который можно понимать как, например, расстояние между городами, время или стоимость поездки. В целях упрощения задачи и гарантии существования маршрута обычно считается, что модельный граф задачи является полностью связным, то есть, что между произвольной парой вершин существует ребро.

1.2 Метод полного перебора

Пусть дано M - число городов, D - матрица смежности, каждый элемент которой - вес пути из одного города в другой. Существует метод грубой силы решения поставленной задачи, а именно полный перебор всех возможных маршрутов в заданном графе с нахождением минимального по весу. Этот метод гарантированно даст идеальное решение (глобальный минимум по весу). Однако стоит учитывать, что сложность такого алгоритма составляет $M!$ и время выполнения программы, реализующий такой подход, будет расти экспоненциально в зависимости от размеров входной матрицы.

1.3 Муравьиный алгоритм

На практике чаще всего необходимо получить решение как можно быстрее, при этом требуемое решение не обязательно должно быть наилучшем, был разработан ряд методов, называемых эвристическими, которые решают поставленную задачу за гораздо меньшее время, чем метод полного перебора. В основе таких методов лежат принципы из окружающего мира, которые в дальнейшем могут быть формализованы.

Одним из таких методов является муравьиный метод [2]. Он применим к решению задачи коммивояжера и основан на идее муравейника.

Пусть у муравья есть 3 чувства:

— зрение (муравей может оценить длину ребра);

— обоняние (муравей может унюхать феромон - вещество, выделяемое муравьем, для коммуникации с другими муравьями);

— память (муравей запоминает свой маршрут).

Благодаря введению обоняния между муравьями возможен непрямой обмен информацией.

Введем вероятность $P_{k,ij}(t)$ выбора следующего города j на маршруте муравьем k, который в текущий момент времени t находится в городе i.

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

где

$\tau_{i,j}$ — феромон на ребре ij;

$\eta_{i,j}$ — привлекательность города j;

α — параметр влияния длины пути;

β — параметр влияния феромона.

Очевидно, что при $\beta = 0$ алгоритм превращается в классический жадный алгоритм, а при $\alpha = 0$ он быстро сойдется к некоторому субоптимальному решению. Выбор правильного соотношения параметров являет-

ся предметом исследований, и в общем случае производится на основании опыта.

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах феромон, обратно пропорциональный длине пройденного пути:

$$\Delta\tau_{k,ij} = \frac{Q}{L_k} \tag{1.2}$$

где

Q – количество феромона, переносимого муравьем;

L_k – стоимость k-го пути муравья (обычно длина).

После окончания условного дня наступает условная ночь, в течение которой феромон испаряется с ребер с коэффициентом ρ. Количество феромона на следующий день вычисляется по следующей формуле:

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

где

$\rho_{i,j}$ – доля феромона, который испарится;

$\tau_{i,j}(t)$ – количество феромона на дуге ij;

$\Delta\tau_{i,j}(t)$ – количество отложенного феромона.

Таким образом, псевдокод муравьиного алгоритма можно представить так:

1. Ввод матрицы расстояний D, количества городов M;
2. Инициализация параметров алгоритма — α β, Q, $tmax$, ρ;
3. Инициализация ребер — присвоение «привлекательности» η_{ij} и начальной концентрации феромона τ_{start};
4. Размещение муравьев по городам;
5. Инициализация начального кратчайшего маршрута L_p = null и определение длины кратчайшего маршрута L_{min} = inf;
6. Цикл по времени жизни колонии $t = 1..t_{max}$;
   (a) Цикл по всем муравьям $k = 1..M$;

i. Построить маршрут $T_k(t)$ по формуле (1.1) и рассчитать длину получившегося маршрута $L_k(t)$;

ii. Обновить феромон на маршруте по формуле (1.2);

iii. Если $L_k(t) < L_{min}$, то $L_{min} = L_k(t)$ и $L_p = T_k(t)$;

(b) Конец цикла по муравьям;

(c) Цикл по всем ребрам графа;

i. Обновить следы феромона на ребре по формуле (1.3);

(d) Конец цикла по ребрам;

7. Конец цикла по времени;

8. Вывести кратчайший маршрут L_p и его длину L_{min}.

1.4 Модификация с элитными муравьями

Одним из усовершенствований является введение в алгоритм так называемых «элитных муравьёв». В дополнение коррекции феромона согласно формуле (1.3) дополнительно добавляется количество феромона, пропорциональное длине лучшего пути для всех его дуг следующим образом:

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \Delta\tau_{i,j}(t) + n_e\Delta\tau^e_{i,j}(t), \tag{1.4}$$

где n_e — число элитных муравьёв.

$$\Delta\tau^e_{i,j}(t) = \frac{Q}{L_{best}} \tag{1.5}$$

где L_{best} — длина наилучшего в данный момент времени маршрута.

Проходя ребра, входящие в короткие пути, муравьи с большей вероятностью будут находить еще более короткие пути. Таким образом, эффективной стратегией является искусственное увеличение уровня феромонов на самых удачных маршрутах. Для этого на каждой итерации алгоритма каждый из элитных муравьев проходит путь, являющийся самым коротким из найденных на данный момент.

Вывод

Таким образом, существуют две группы методов для решения задачи коммивояжера - точные и эвристические. К точным относится метод полного перебора, к эвристическим - муравьиный метод. Применение муравьиного алгоритма обосновано в тех случаях, когда необходимо быстро найти решение или когда для решения задачи достаточно получения первого приближения. В случае необходимости максимально точного решения используется алгоритм полного перебора.

2 Конструкторская часть

В этом разделе содержатся схемы алгоритмов решения задачи коммивояжера.

2.1 Разработка алгоритмов

На рисунке 2.1 представлен алгоритм полного перебора.

Рисунок 2.1 – Схема алгоритма полного перебора.

На рисунке 2.2 представлен муравьиный алгоритм.

Рисунок 2.2 – Схема муравьиного алгоритма.

Вывод

В данном разделе были разработаны схема алгоритма полного перебора и схема муравьиного алгоритма.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к программе

Программа на вход получает матрицу смежности графа.

Выход программы: суммарная стоимость этого маршрута — целое число и минимальный по стоимости маршрут — последовательность целых чисел.

Алгоритм полного перебора должен возвращать кратчайший путь.

Программа должна производить параметризацию реализации метода на основе муравьиного алгоритма.

Программа должна замерять время выполнения реализаций метода полного перебора и метода на основе муравьиного алгоритма

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования C# [3], т.к. его средств достаточно для реализации поставленной задачи.

Время работы алгоритмов было замерено с помощью свойства TotalProcessorTime класса Process, которое возвращает объект TimeSpan, указывающий количество времени, потраченного процессом на загрузку ЦП [4].

3.3 Сведения о модулях программы

Программа состоит из следующих модулей.

1. Programm.cs — главный файл программы, в котором располагается код меню.

2. AntAlgorithm.cs — файл с классом, реализующим муравьиный алгоритм.

3. BruteForce.cs — файлы с классом, реализующим алгоритм полного перебора.

4. Map.cs, Path.cs — файлы классами маршрута и графа соответственно.

3.4 Реализация алгоритмов

В листингах 3.1, 3.2 представлены реализации муравьиного алгоритма и алгоритма полного перебора.

Листинг 3.1 – Реализация муравьиного алгоритма

```
1  static class AntAlgorithm
2  {
3      public static Path GetRoute(Map map, int maxTime, double
           alpha, double beta, double Q, double pho)
4      {
5          Random r = new Random();
6
7          Path shortest = new Path(null, map, int.MaxValue);
8
9          int count = map.N;
10         double[,] pher = InitPheromone(0.1, count);
11
12         for (int time = 0; time < maxTime; time++)
13         {
14             List<Ant> ants = InitAnts(map);
15             double[,] deltaPher = InitPheromone(0, count);
16             for (int i = 0; i < count - 1; i++)
17                 foreach (Ant ant in ants)
18                 {
19                     int curTown = ant.LastVisited();
20
21                     double sum = 0;
22                     for (int town = 0; town < count; town++)
23                         if (!ant.IsVisited(town))
24                         {
25                             double tau = pher[curTown, town];
```

```
26                          double eta = 1.0 / map[curTown, town];
27                          sum += Math.Pow(tau, alpha) *
                                Math.Pow(eta, beta);
28                      }
29
30                  double check = r.NextDouble();
31                  int newTown = 0;
32                  for (; check > 0; newTown++)
33                      if (!ant.IsVisited(newTown))
34                      {
35                          double tau = pher[curTown, newTown];
36                          double eta = 1.0 / map[curTown,
                                newTown];
37                          double chance = Math.Pow(tau, alpha)
                                * Math.Pow(eta, beta) / sum;
38                          check -= chance;
39                      }
40                  newTown--;
41                  ant.VisitTown(newTown);
42                  deltaPher[curTown, newTown] += Q /
                        map[curTown, newTown];
43                  deltaPher[newTown, curTown] += Q /
                        map[newTown, curTown];
44              }
45              foreach (Ant ant in ants)
46              {
47                  if (ant.GetDistance() < shortest.N)
48                      shortest = ant.GetPath();
49              }
50              Ant elite = new Ant(map, shortest.Way[0]);
51              for (int i = 1; i < count; i++)
52              {
53                  int newTown = shortest.Way[i];
54                  int curTown = elite.LastVisited();
55                  elite.VisitTown(newTown);
56                  deltaPher[curTown, newTown] += Q / map[curTown,
                        newTown];
57                  deltaPher[newTown, curTown] += Q / map[newTown,
                        curTown];
58              }
59              for (int k = 0; k < count; k++)
```

```
60                  for (int t = 0; t < count; t++)
61                  {
62                      pher[k, t] = (1 - pho) * pher[k, t] +
                            deltaPher[k, t];
63                      pher[k, t] = pher[k, t] < 0.1 ? 0.1 : pher[k,
                            t];
64                  }
65          }
66          return shortest;
67      }
68
69      private static List<Ant> InitAnts(Map map)
70      {
71          List<Ant> ants = new List<Ant>();
72          for (int i = 0; i < map.N; i++)
73              ants.Add(new Ant(map, i));
74          return ants;
75      }
76
77      private static double[,] InitPheromone(double num, int size)
78      {
79          double[,] phen = new double[size, size];
80          for (int i = 0; i < size; i++)
81              for (int j = 0; j < size; j++)
82                  phen[i, j] = num;
83          return phen;
84      }
85
86  }
```

Листинг 3.2 – Реализация алгоритма полного перебора

```
1       static class BruteForce
2       {
3           public static Path GetRoute(Map map)
4           {
5               Path shortest = new Path(null, map, int.MaxValue);
6               List<int> a = new List<int>();
7               for (int i = 0; i < map.N; i++)
8                   a.Add(i);
9               foreach (List<int> cur in GetAllRoutes(a, new
                    List<int>()))
```

```
10              {
11                  Path check = new Path(cur, map, -1);
12                  check.GetDistance();
13                  if (shortest.N > check.N)
14                      shortest = check;
15              }
16              return shortest;
17          }
18
19          private static IEnumerable<List<int>>
               GetAllRoutes(List<int> arg, List<int> awithout)
20          {
21              if (arg.Count == 1)
22              {
23                  var result = new List<List<int>> { new
                       List<int>() };
24                  result[0].Add(arg[0]);
25                  return result;
26              }
27              else
28              {
29                  var result = new List<List<int>>();
30
31                  foreach (var first in arg)
32                  {
33                      var others0 = new List<int>(arg.Except(new
                           int[1] { first }));
34                      awithout.Add(first);
35                      var others = new
                           List<int>(others0.Except(awithout));
36
37                      var combinations = GetAllRoutes(others,
                           awithout);
38                      awithout.Remove(first);
39
40                      foreach (var tail in combinations)
41                      {
42                          tail.Insert(0, first);
43                          result.Add(tail);
44                      }
45                  }
```

```
46                  return result;
47              }
48          }
49      }
```

3.5 Тестирование

В таблице 3.1 приведены тесты для методов, реализующих алгоритм полного перебора и муравьиный алгоритм. Тесты пройдены успешно. При тестировании муравьиного алгоритма тесты прошли успешно, так как матрицы смежности маленькие, и поэтому муравьиный алгоритм возвращает такие же результаты, как и алгоритм полного перебора.

Таблица 3.1 – Тестирование методов

Матрица смежности	Ожидаемый результат	Действительный результат
$\begin{pmatrix} 0 & 1 & 10 & 7 \\ 1 & 0 & 1 & 2 \\ 10 & 1 & 0 & 1 \\ 7 & 2 & 1 & 0 \end{pmatrix}$	3 – 0 1 2 3	3 – 0 1 2 3
$\begin{pmatrix} 0 & 3 & 5 & 7 \\ 7 & 0 & 1 & 2 \\ 5 & 1 & 0 & 1 \\ 7 & 2 & 1 & 0 \end{pmatrix}$	8 – 0 2 1 3	8 – 0 2 1 3
$\begin{pmatrix} 0 & 3 & 5 \\ 3 & 0 & 1 \\ 5 & 1 & 0 \end{pmatrix}$	4 – 0 1 2	4 – 0 1 2

Вывод

Были реализованы муравьиный алгоритм и алгоритм полного перебора.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование.

1. Операционная система: Windows 10 Корпоративная, Версия 21H1, Сборка ОС 19043.2006.
2. Оперативная память: 8 ГБ.
3. Процессор: AMD Ryzen 5 4600H с видеокартой Radeon Graphics 3.00 ГГц [5].

Исследование проводилось на ноутбуке, включенном в сеть электропитания. Во время исследования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой.

4.2 Демонстрация работы программы

На рисунке 4.1 представлены результаты работы реализаций алгоритма полного перебора и муравьиного алгоритма.

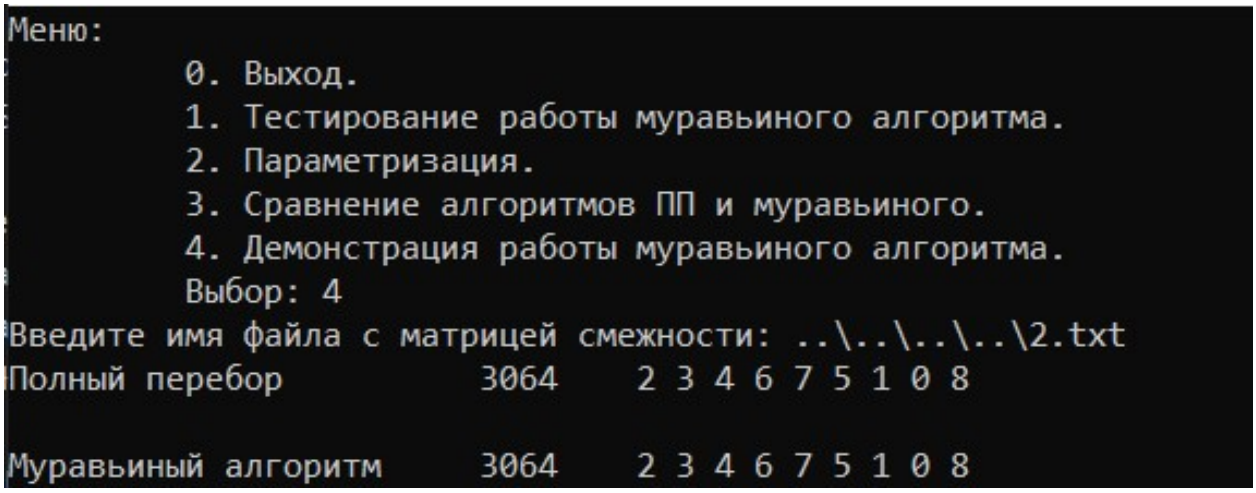

Рисунок 4.1 – Демонстрация работы программы в консоли

4.3 Параметризация метода

Для проведения параметризации были использованы матрицы смежности (4.1), (4.2), (4.3). В качестве класса данных была выбрана карта перемещения по портам через океаны. Матрицы смежности для выбранного класса данных:

$$M_1 = \begin{pmatrix} 0 & 681 & 335 & 725 & 1215 & 657 & 210 & 1325 & 372 & 227 \\ 681 & 0 & 675 & 152 & 1537 & 1080 & 886 & 1494 & 694 & 454 \\ 335 & 675 & 0 & 827 & 1547 & 405 & 364 & 1660 & 704 & 518 \\ 725 & 152 & 827 & 0 & 1470 & 1232 & 930 & 1373 & 657 & 498 \\ 1215 & 1537 & 1547 & 1470 & 0 & 1825 & 1378 & 580 & 843 & 1136 \\ 657 & 1080 & 405 & 1232 & 1825 & 0 & 447 & 1949 & 982 & 879 \\ 210 & 886 & 364 & 930 & 1378 & 447 & 0 & 1502 & 535 & 432 \\ 1325 & 1494 & 1660 & 1373 & 580 & 1949 & 1502 & 0 & 967 & 1204 \\ 372 & 694 & 704 & 657 & 843 & 982 & 535 & 967 & 0 & 293 \\ 227 & 454 & 518 & 498 & 1136 & 879 & 432 & 1204 & 293 & 0 \end{pmatrix} \quad (4.1)$$

$$M_2 = \begin{pmatrix} 0 & 140 & 1194 & 1362 & 688 & 167 & 481 & 220 & 325 \\ 140 & 0 & 1054 & 1277 & 640 & 185 & 341 & 154 & 382 \\ 1194 & 1054 & 0 & 784 & 731 & 1164 & 749 & 997 & 1436 \\ 1362 & 1277 & 784 & 0 & 674 & 1306 & 1048 & 1169 & 1659 \\ 688 & 640 & 731 & 674 & 0 & 632 & 505 & 495 & 1002 \\ 167 & 185 & 1164 & 1306 & 632 & 0 & 451 & 167 & 492 \\ 481 & 341 & 749 & 1048 & 505 & 451 & 0 & 284 & 723 \\ 220 & 154 & 997 & 1169 & 495 & 167 & 284 & 0 & 507 \\ 325 & 382 & 1436 & 1659 & 1002 & 492 & 723 & 507 & 0 \end{pmatrix} \quad (4.2)$$

$$M_3 = \begin{pmatrix} 0 & 1726 & 541 & 139 & 744 & 829 & 361 & 447 & 1278 \\ 1726 & 0 & 2149 & 1865 & 2173 & 1969 & 2087 & 1904 & 2864 \\ 541 & 2149 & 0 & 402 & 389 & 710 & 182 & 741 & 785 \\ 139 & 1865 & 402 & 0 & 629 & 838 & 222 & 400 & 1139 \\ 744 & 2173 & 389 & 629 & 0 & 330 & 543 & 1029 & 951 \\ 829 & 1969 & 710 & 838 & 330 & 0 & 849 & 1238 & 1272 \\ 361 & 2087 & 182 & 222 & 543 & 849 & 0 & 559 & 917 \\ 447 & 1904 & 741 & 400 & 1029 & 1238 & 559 & 0 & 1370 \\ 1278 & 2864 & 785 & 1139 & 951 & 1272 & 917 & 1370 & 0 \end{pmatrix} \quad (4.3)$$

Первый столбец представляет собой значения параметра t_{max}, второй — α, третий — ρ, в четвертом столбце представлено максимальное отклонение длины маршрута, полученного методом на основе муравьиного алгоритма, от оптимальной длины маршрута, вычисленной методом полного перебора, в пятом — отклонение медианы.

Для каждой комбинации параметров, муравьиный алгоритм запускался по 30 раз (для каждой матрицы 10 раз). Для каждого из 30 полученных значений вычислялось отклонение от наилучшего маршрута. Для таблицы выбирались максимальное значение отклонения и медианное.

Как видно из таблицы А.1, наилучшим значением настроечного параметра α является значение, равное 0.5, а коэффициента испарения ρ — 0.1. Уже при $t_{max} = 500$ все медианные отклонения равны нулю.

4.4 Время выполнения реализаций алгоритмов

Замеры времени для каждого количества количества вершин в графе проводились 20 раз. В качестве результата взято среднее время работы алгоритма на данном количества вершин при следующих параметрах: $\alpha = 0.5$, $\rho = 0.5$, $t_{max} = 500$.

Результаты замеров времени приведены в таблице 4.1. На рисунке 4.2, приведена зависимость времени работы реализаций алгоритмов.

Таблица 4.1 – Время выполнения реализаций алгоритма полного перебора и муравьиного алгоритма

Количество вершин	Муравьиный алгоритм, мс	Полный перебор, мс
5	12.500	0.000
6	20.313	1.563
7	33.594	9.375
8	50.781	90.625
9	73.438	1007.813
10	96.094	10689.063

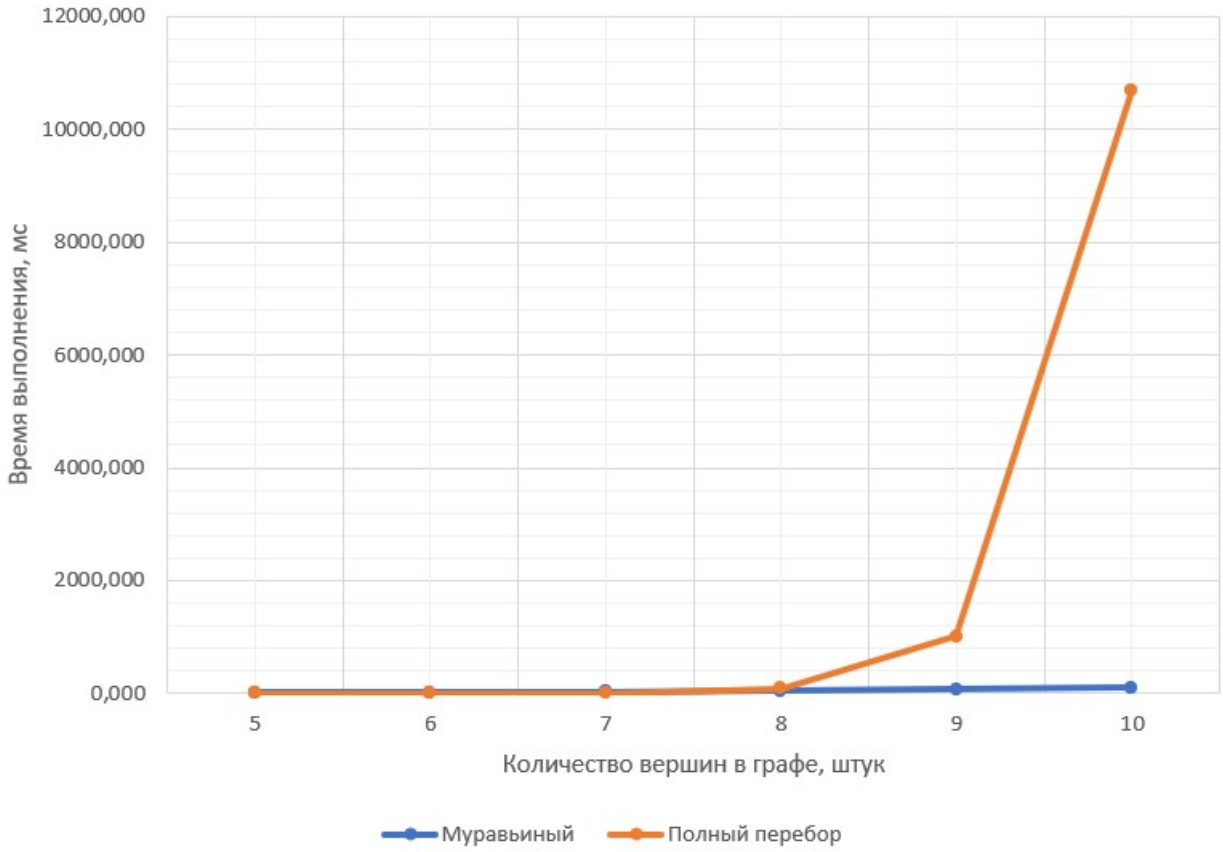

Рисунок 4.2 – Зависимость времени работы реализаций алгоритма полного перебора и муравьиного алгоритма

Как можно наблюдать на графике, время работы алгоритма полного перебора растет экспоненциально в зависимости от размера матрицы смежности, в отличие от муравьиного алгоритма, время которого изменяется практически линейно.

4.5 Оценка трудоёмкости

Алгоритм полного перебора имеет факториальную сложность, т.к. количество всех маршрутов равно числу размещений из n городов.

$$A_n = n! \tag{4.4}$$

Следовательно сложность метода полного перебора $O(n!)$

Сложность муравьиного алгоритма равна $O(t_{max} \cdot n^2 \cdot m)$, где t_{max} — время жизни колонии, m — количество муравьев в колонии, n — количество точек в графе.

— $f_{\text{утро}} = 6 + 9n + 5n^2$

— $f_{\text{день}} = n - 39m + 24mn + 15mn^2$

— $f_{\text{закат}} = 5 + 19n + 7m + 4mn$

— $f_{\text{ночь}} = 1 + 3n + 20n^2$

Итоговая трудоемкость метода на основе муравьиного алгоритма рассчитывается по формуле (4.5)

$$f_{\text{муравьиного алгоритма}} = t_{max} \cdot (f_{\text{утро}} + f_{\text{день}} + f_{\text{закат}} + f_{\text{ночь}}) = t_{max} \cdot (12 + 32n - 32m + 25n^2 + 29mn + 15mn^2) \tag{4.5}$$

4.6 Вывод

В результате проведенных экспериментов были выявлены оптимальные параметры для метода на основе муравьиного алгоритма при выбранном классе эквивалентности: $\alpha = 0.5$, $\rho = 0.1$, $t_{max} = 500$. Однако стоит учитывать, что чем больше значение t_{max}, тем больше вероятность того, что будет найден оптимальный маршрут, но при этом будет возрастать время выполнения программы.

Также был проведен сравнительный анализ муравьиного алгоритма и алгоритма полного перебора. Метод полного перебора следует использовать для матриц небольшого размера (до 8) и в случае, если необходимо получить точное решение. В остальных случаях муравьиный алгоритм является более эффективным по времени, если достаточно получить хорошее решение по выбранной метрике (например, отклонение длины маршрута, полученного методом на основе муравьиного алгоритма, от оптимальной длины маршрута).

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

— описаны методы решения.

— описана реализация, реализован метод.

— выбран класс данных, составлен набор данных.

— проведена параметризация метода на основании муравьиного алгоритма для выбранного класса данных.

— проведен сравнительный анализ двух методов.

— даны рекомендации о применимости метода решения задачи коммивояжера на основе муравьиного алгоритма.

Поставленная цель достигнута: изучен муравьиный алгоритм на материале решения задачи коммивояжёра.

В результате проведенных экспериментов были выявлены оптимальные параметры для метода на основе муравьиного алгоритма при выбранном классе эквивалентности: $\alpha = 0.5$, $\rho = 0.1$, $t_{max} = 500$.

Метод полного перебора следует использовать для матриц небольшого размера (до 8) и в случае, если необходимо получить точное решение. В остальных случаях муравьиный алгоритм является более эффективным по времени, если достаточно получить хорошее решение по выбранной метрике (например, отклонение длины маршрута, полученного методом на основе муравьиного алгоритма, от оптимальной длины маршрута).

Список использованных источников

1. Мудров В.И. Задача о коммивояжёре. — М.: Наука, 1969. — 62 с.

2. Дориго М. Муравьиная система. — М.: Наука, 1996. — 29 с.

3. Краткий обзор языка C# [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/` (дата обращения: 25.09.2022).

4. Process.TotalProcessorTime Свойство [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.process.totalprocessortime?view=net-6.0` (дата обращения: 02.12.2022).

5. AMD Ryzen™ 5 4600H [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-5-4600h` (дата обращения: 25.09.2022).

Приложение А

Таблица с результатами параметризации

Таблица А.1 – Результаты параметризации

Параметр t_{max}	Параметр α	Параметр ρ	Максимальное отклонение	Медианное отклонение
100	0.1	0.1	341	92
100	0.1	0.25	438	136
100	0.1	0.5	318	92
100	0.1	0.75	429	112
100	0.1	0.9	401	112
100	0.25	0.1	401	0
100	0.25	0.25	322	39
100	0.25	0.5	325	83
100	0.25	0.75	201	57
100	0.25	0.9	236	83
100	0.5	0.1	184	0
100	0.5	0.25	143	0
100	0.5	0.5	152	0
100	0.5	0.75	309	0
100	0.5	0.9	271	39
100	0.75	0.1	60	0
100	0.75	0.25	60	0
100	0.75	0.5	149	0
100	0.75	0.75	194	0
100	0.75	0.9	265	0
100	0.9	0.1	92	0
100	0.9	0.25	149	0
100	0.9	0.5	92	0
100	0.9	0.75	194	0
100	0.9	0.9	271	14

Продолжение таблицы А.1

Параметр t_{max}	Параметр α	Параметр ρ	Максимальное отклонение	Медианное отклонение
200	0.1	0.1	370	0
200	0.1	0.25	312	53
200	0.1	0.5	276	39
200	0.1	0.75	318	71
200	0.1	0.9	277	53
200	0.25	0.1	277	14
200	0.25	0.25	190	0
200	0.25	0.5	201	0
200	0.25	0.75	201	0
200	0.25	0.9	271	39
200	0.5	0.1	152	0
200	0.5	0.25	60	0
200	0.5	0.5	60	0
200	0.5	0.75	124	0
200	0.5	0.9	152	0
200	0.75	0.1	0	0
200	0.75	0.25	14	0
200	0.75	0.5	14	0
200	0.75	0.75	149	0
200	0.75	0.9	152	0
200	0.9	0.1	39	0
200	0.9	0.25	60	0
200	0.9	0.5	60	0
200	0.9	0.75	92	0
200	0.9	0.9	194	0
500	0.1	0.1	152	0
500	0.1	0.25	188	0
500	0.1	0.5	141	0
500	0.1	0.75	224	0
500	0.1	0.9	194	0

Продолжение таблицы А.1

Параметр t_{max}	Параметр α	Параметр ρ	Максимальное отклонение	Медианное отклонение
500	0.25	0.1	146	0
500	0.25	0.25	152	0
500	0.25	0.5	143	0
500	0.25	0.75	194	0
500	0.25	0.9	146	0
500	0.5	0.1	0	0
500	0.5	0.25	0	0
500	0.5	0.5	0	0
500	0.5	0.75	14	0
500	0.5	0.9	141	0
500	0.75	0.1	0	0
500	0.75	0.25	0	0
500	0.75	0.5	14	0
500	0.75	0.75	60	0
500	0.75	0.9	146	0
500	0.9	0.1	0	0
500	0.9	0.25	0	0
500	0.9	0.5	60	0
500	0.9	0.75	60	0
500	0.9	0.9	60	0
1000	0.1	0.1	143	0
1000	0.1	0.25	149	0
1000	0.1	0.5	143	0
1000	0.1	0.75	124	0
1000	0.1	0.9	124	0
1000	0.25	0.1	124	0
1000	0.25	0.25	124	0
1000	0.25	0.5	60	0
1000	0.25	0.75	124	0
1000	0.25	0.9	141	0

Продолжение таблицы А.1

Параметр t_{max}	Параметр α	Параметр ρ	Максимальное отклонение	Медианное отклонение
1000	0.5	0.1	0	0
1000	0.5	0.25	0	0
1000	0.5	0.5	0	0
1000	0.5	0.75	0	0
1000	0.5	0.9	0	0
1000	0.75	0.1	0	0
1000	0.75	0.25	0	0
1000	0.75	0.5	0	0
1000	0.75	0.75	0	0
1000	0.75	0.9	60	0
1000	0.9	0.1	0	0
1000	0.9	0.25	0	0
1000	0.9	0.5	0	0
1000	0.9	0.75	60	0
1000	0.9	0.9	60	0
2000	0.1	0.1	124	0
2000	0.1	0.25	124	0
2000	0.1	0.5	60	0
2000	0.1	0.75	124	0
2000	0.1	0.9	146	0
2000	0.25	0.1	0	0
2000	0.25	0.25	60	0
2000	0.25	0.5	0	0
2000	0.25	0.75	60	0
2000	0.25	0.9	60	0
2000	0.5	0.1	0	0
2000	0.5	0.25	0	0
2000	0.5	0.5	0	0
2000	0.5	0.75	0	0
2000	0.5	0.9	0	0

Продолжение таблицы А.1

Параметр t_{max}	Параметр α	Параметр ρ	Максимальное отклонение	Медианное отклонение
2000	0.75	0.1	0	0
2000	0.75	0.25	0	0
2000	0.75	0.5	0	0
2000	0.75	0.75	0	0
2000	0.75	0.9	60	0
2000	0.9	0.1	0	0
2000	0.9	0.25	0	0
2000	0.9	0.5	0	0
2000	0.9	0.75	0	0
2000	0.9	0.9	0	0