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 дисциплине "Анализ алгоритмов"

Тема Муравьиный алгоритм

Студент Андрич К.

Группа ИУ7И-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Задача коммивояжера применяется в различных сферах, таких как: маршрутизация транспортных потоков и выбор оптимальной траектории движения рабочего инструмента, а также в сферах, которые на первый взгляд не связаны с маршрутизацией: секвенирования нуклеотидных последовательностей биополимеров, эвристического определения схожести строк, построения практических алгоритмов исследования специально определённых бесконечных грамматических структур и построения эволюционных деревьев.

Муравьиный алгоритм (алгоритм оптимизации подражанием муравьиной колонии) — один из полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Алгоритм называется полиномиальным (имеющим полиномиальную временную сложность), если существует полином $p(x)$ такой, что на любом входном слове длины n алгоритм завершает вычисления после выполнения не более чем $p(n)$ операций .

Целью работы работы является проведение сравнительного анализа метода полного перебора и эвристического метода на базе муравьиного алгоритма.

Для достижения цели ставятся следующие задачи:

- изучить метод полного перебора и метод на базе муравьиного алгоритма для решения задачи коммивояжёра;
- реализовать метод полного перебора и метод на базе муравьиного алгоритма для решения задачи коммивояжёра;
- провести параметризацию метода на базе муравьиного алгоритма для выбранного класса задач, т.е. определить такие комбинации параметров или их диапазонов, при которых метод даёт наилучшие результаты на выбранном классе задач;
- сравнить алгоритм полного перебора и муравьиный по скорости работы.

1 | Аналитическая часть

В данной части будут рассмотрены основные теоретические аспекты, связанные с задачей коммивояжера.

1.1 Задача коммивояжера

Задача коммивояжера формулируется просто: на плоскости (в пространстве) расположены N городов, заданы расстояния между каждой парой городов. Требуется найти маршрут минимальной длины с посещением каждого города ровно один раз и с возвращением в исходную точку. Необходимо дополнительно учитывать расстояния от города до города, которые предполагаются известными. Эти «расстояния» можно заменить на количество затраченного времени, стоимость проезда или предполагать другие произвольные значения.

На графах задача формулируется следующим образом: требуется найти гамильтонов цикл наименьшей стоимости во взвешенном полном графе. Т.е. выйдя из стартовой вершины, посетить каждую вершину графа ровно один раз и вернуться в начальную по кратчайшему пути.

Гамильтонов путь – путь, содержащий каждую вершину графа ровно один раз. Гамильтонов путь, начальная и конечная вершины которого совпадают, называется гамильтоновым циклом.

1.2 Муравьиный алгоритм

Муравьинный алгоритм – один из алгоритмов нахождения решений задачи коммивояжера.

Алгоритм основывается на поведении муравьев в поиске кратчайшего пути от колонии до источника питания. Моделирование поведения муравья связано с распределением феромонов на пути (на ребре в данном случае), количество феромона пропорционально длине маршрута. Таким образом, чем короче найденный маршрут, тем больше будет на нем феромонов. Чтобы алгоритм не сводился к единственному варианту, вводится «испарение» феромонов с пути со временем.

Муравьинный алгоритм использует три основных понятия:

1. память муравья – список вершин, которые он уже посетил; также есть список $J_{i,k}$ – список вершин, которые нужно посетить i-тому муравью, находящемуся в вершине k;

2. видимость – обратная расстоянию величина, $\eta = \frac{1}{D_{i,j}}$, где $D_{i,j}$ – расстояние между вершинами i и j;

3. феромонный след – «опыт» муравьев, коэффициент вероятности того, что муравей захочет перейти из вершины i в j – $\tau_{i,j}$.

Вероятность перехода муравья k в вершину j из вершины i рассчитывается по формуле 1.1:

$$P_{i,j,k}(t) = \begin{cases} \frac{[\tau_{i,j}(t)]^{\alpha} * [\eta_{i,j}]^{\beta}}{\sum\limits_{l=J_{i,k}} [\tau_{i,j}(t)]^{\alpha} * [\eta_{i,j}]^{\beta}} & j \in J_{i,k} \\ 0 & j \notin J_{i,k} \end{cases} \tag{1.1}$$

α, β – весовые коэффициенты, которые задают важность феромона и привлекательность ребра.

Если $\alpha = 0$, то алгоритм становится «жадным», если $\beta = 0$, то муравьи оказываются «слепы» и опираются только на феромонный след.

Пройдя ребро (i, j), муравей откладывает на нем некоторое количество феромона, которое рассчитывается по формуле 1.2:

$$\Delta\tau_{i,j,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, & (i,j) \in T_k(t) \\ 0, & (i,j) \notin T_k(t) \end{cases} \tag{1.2}$$

p - коэффициент испарения феромона;
Q - значение порядка длины оптимального пути;
L_k - длина маршруту, пройденная муравьем k к моменту времени t;

Пусть $p \in [0, 1]$ - коэффициент испарения, тогда правило испарения имеет вид (формула 1.3):

$$\tau_{i,j}(t+1) = (1-p) * \tau_{i,j}(t) + \Delta\tau_{i,j}(t); \Delta\tau_{i,j}(t) = \sum_{k=1}^{m} \Delta\tau_{i,j,k}(t) \tag{1.3}$$

где m - количество муравьев в колонии.

1.3 Вывод

В данном разделе поставлена цель и описаны задачи и теоретическая часть муравьиного алгоритма и полного перебора.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы алгоритмов полного перебора и муравьиного алгоритма.

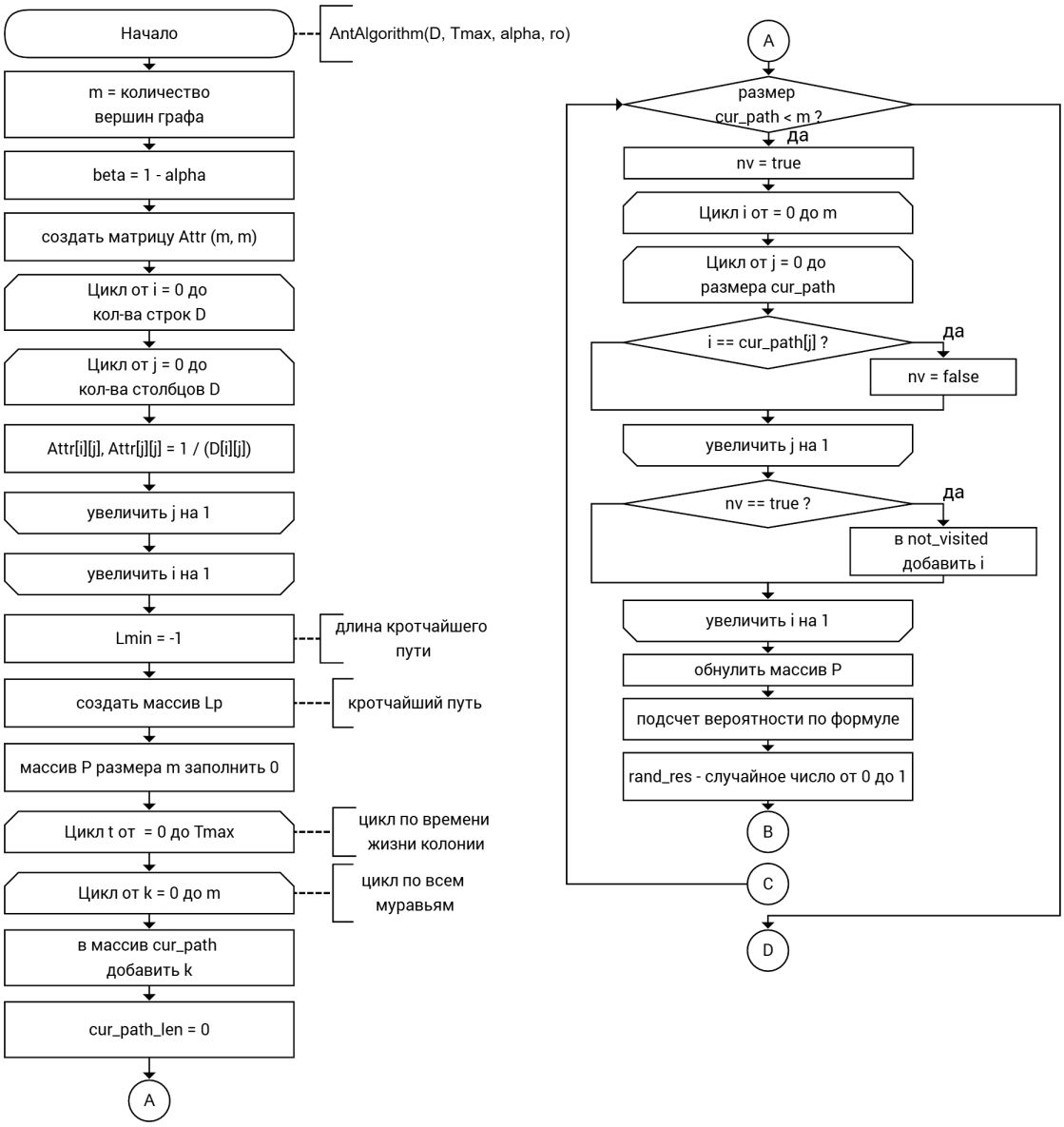

Рисунок 2.1 - Муравьиный алгоритм (часть 1)

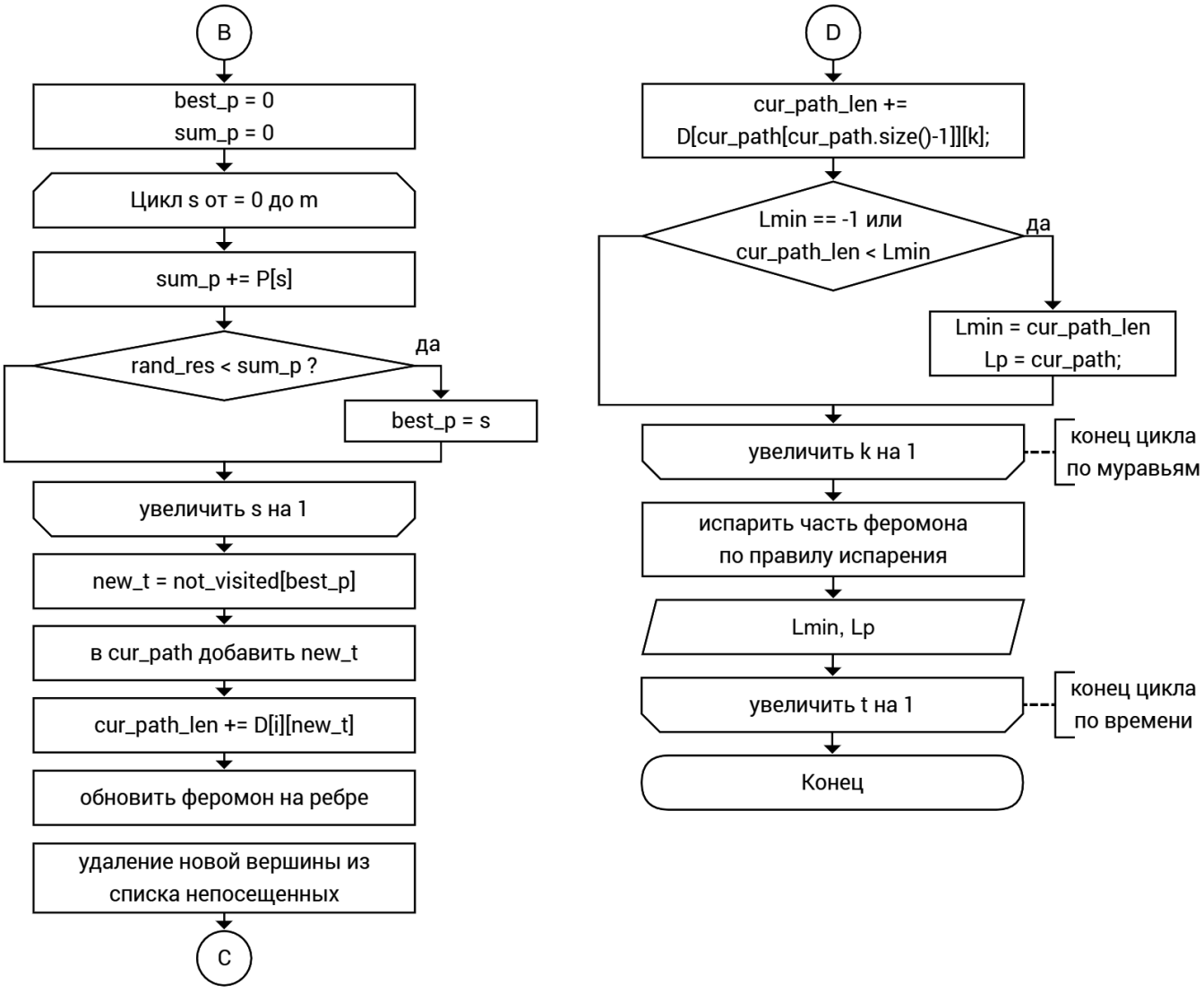

Рисунок 2.2 - Муравьиный алгоритм (часть 2)

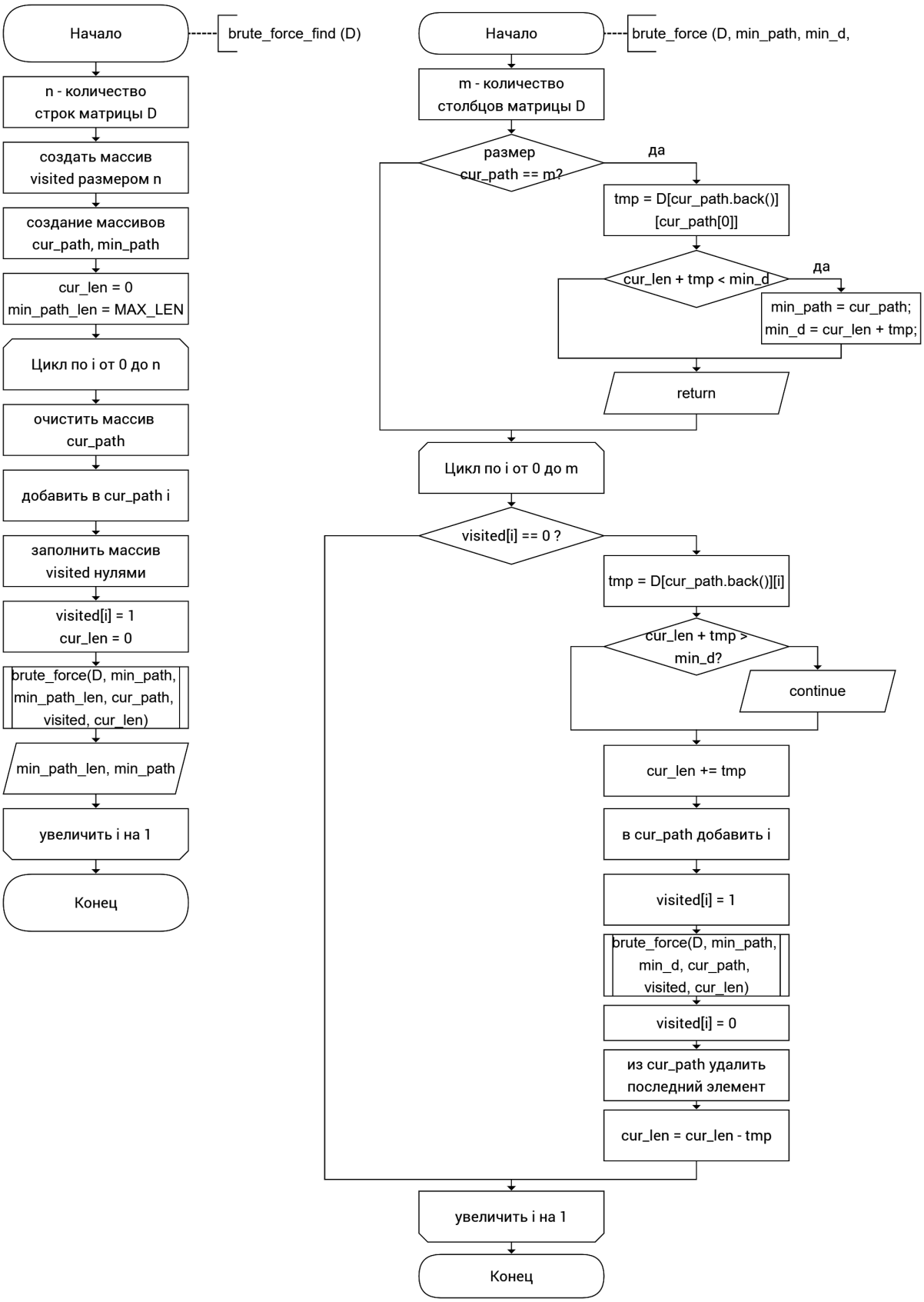

Рисунок 2.3 - Алгоритм полного перебора

2.2 Требования к программному обеспечению

На входе программа принимает матрицу смежности графа. На выходе ПО должно обеспечить вывод замеров времени работы алгоритма полного перебора и муравьиного алгоритма, выводить на экран длину самого короткого пути (стоимость обхода) и сам путь.

2.3 Оценка трудоемкости муравьиного алгоритма

Ниже приведены теоретические оценки трудоемкости муравьиного алгоритма.

1) Трудоемкость цикла поиска непосещенных вершин (формула 2.1):

$$f_1 = 1 + 2 + N \cdot (2 + 1 + 2 + 1 + S \cdot (3 + 1 + 1 + 1) + 1 + 1) = 3 + 6 \cdot N \cdot S + 8 \cdot N \quad (2.1)$$

2) Трудоемкость цикла подсчета вероятности (формула 2.2):

$$f_2 = 2 + N \cdot (2 + 1 + 1) + 2 + S \cdot (2 + 3 + 1 + 1 + 2 + S \cdot (2 + 4 + 2 + 1) + 1 + 7 + 2 + 1 + 1)$$

$$f_2 = 4 + 3 \cdot N + 21 \cdot S + 9 \cdot S^2 \quad (2.2)$$

3) Трудоемкость цикла вычисления нового города, в которую попадет муравей (формула 2.3):

$$f_3 = 4 + 2 + 2 + N \cdot (2 + 2 + 3 + 2) = 8 + 9 \cdot N \quad (2.3)$$

4) Трудоемкость цикла, в которром происходит испарение феромона (формула 2.4):

$$f_4 = 2 + N \cdot (2 + 2 + N \cdot (2 + 6 + 4 + 2 + 1 + 3)) = 2 + 4 \cdot N + 18 \cdot N^2 \quad (2.4)$$

5) Трудоемкость основоного цикла по времени жизни колонии (формула 2.5):

$$f_5 = 2 + T_{max} \cdot (3 + 2 + M \cdot (2 + 1 + 1 + 2 + 1 + N \cdot [1 + 1 + f_1 + f_2 + f_3 + 2 + 1 + 3 + 4 + 1] + 3 + 6 + 3 + 2) + f_4)$$

$$f_5 = 2 + T_{max} \cdot (5 + M \cdot (7 + N \cdot [2 + 3 + 6 \cdot N \cdot S + 8 \cdot N + 4 + 3 \cdot N + 21 \cdot S + 9 \cdot S^2 + 8 + 9 \cdot N + 11] + 14) + 2 + 4 \cdot N + 18 \cdot N^2)$$

$$f_5 = 2 + T_{max} \cdot (7 + M \cdot (21 + N \cdot (28 + 21 \cdot S + 9 \cdot S^2) + N^2 \cdot (20 + 6 \cdot S)) + 4 \cdot N + 18 \cdot N^2) \quad (2.5)$$

Таким образом, сложность муравьиного алгоритма равна $O(T_{max} \cdot M \cdot N^2)$.

С учетом того, что в моей реализации алгоритма в каждом городе находится 1 муравей, сложность алгоритма равна $O(T_{max} \cdot N^3)$.

2.4 Вывод

На основе теоретических данных, полученные в аналитическом разделе были построены схемы иследуеммых алгоритмов и показана оценка трудоемкости муравьиного алгоритма.

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран $C++$ [3], так как я имею опыт написания программ на данном языке. В качестве среды разработки выбран $VisualStudio$.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.4 представлена реализация алгоритма полного перебора и муравьиного алгоритма.

Листинг 3.1: Реализация алгоритма полного перебора (нерекурсивная часть)

```
pair<int, vector<int> > brute_force_find(Matrix D)
{
  int n = D.rows();
  vector<bool> visited(n, 0);
  vector<int> cur_path;
  vector<int> min_path;
  int cur_len = 0;
  int min_path_len = MAX_LEN;
  for (int i = 0; i < n; i++)
  {
    cur_path.clear();
    cur_path.push_back(i);
    fill(visited.begin(), visited.end(), 0);
    visited[i] = 1;
    cur_len = 0;
    brute_force(D, min_path, min_path_len, cur_path, visited, cur_len);
  }
  return pair<int, vector<int>>(min_path_len, min_path);
}
```

Листинг 3.2: Реализация алгоритма полного перебора (рекурсивная часть)

```
void brute_force(Matrix &D, vector<int> &min_path, int &min_d, vector<int> &
    cur_path, vector<bool> &visited, int &cur_len) {
  size_t M = D.cols();
  if (cur_path.size() == M) {
    ss++;
    int tmp = D[cur_path.back()][cur_path[0]];
    if (cur_len + tmp < min_d)
    {
      min_path = cur_path;
      min_d = cur_len + tmp;
    }
    return;
  }
  for (size_t i = 0; i < M; i++) {
    if (!visited[i])
    {
      int tmp = D[cur_path.back()][i];
      if (cur_len + tmp > min_d)
      continue;
      cur_len += tmp;
      cur_path.push_back(i);
      visited[i] = 1;
      brute_force(D, min_path, min_d, cur_path, visited, cur_len);
      visited[i] = 0;
      cur_path.pop_back();
      cur_len -= tmp;
    }
  }
}
```

Листинг 3.3: Реализация муравьиного алгоритма част 1

```
pair<int, vector<int>> AntAlgorithm(Matrix D, const int Tmax, const double
    alpha, const double ro){
  const int teta_beg = 10;
  const int teta_min = 5;
  const size_t M = D.cols();
  const double Q = D.avg() * M;
  const double betta = 1 - alpha;

  Matrix Attr(M, M);
  for (size_t i = 0; i < D.rows(); i++){
    for (size_t j = 0; j < D.cols(); j++) {
      double tmp = 1/D[i][j];
      Attr[i][j] = tmp;
      Attr[j][i] = tmp;
    }
  }
  Matrix Teta(M, M, teta_beg);
  Matrix Delta_Teta(M, M);

  int Lmin = -1;
  vector<int> Lp;
  vector<double> P(M, 0.0);

  double random_res;
  for (int t = 0; t < Tmax; t++) {
    Delta_Teta.zero();
    for (size_t k = 0; k < M; k++) {
      vector<int> cur_path;
      cur_path.push_back(k);
      int cur_path_length = 0;
      int i = k;
      while (cur_path.size() < M) {
        vector<int> not_visited;
        bool nv = true;
        for (size_t i = 0; i < M; i++) {
          nv = true;
          for (size_t j = 0; j < cur_path.size(); j++) {
            if ((int)i == cur_path[j]) nv = false;
          }
          if (nv)
          not_visited.push_back(i);
        }
        for (size_t j = 0; j < M ; j++){
          P[j] = 0.0;
        }
```

```
        for (size_t j = 0; j < not_visited.size(); j++) {
          if (D[i][not_visited[j]]) {
            double sum = 0;
            for (auto n: not_visited) {
              sum += pow(Teta[i][n], alpha) * pow(Attr[i][n], betta);
            }
            P[j] = pow(Teta[i][not_visited[j]], alpha) *    pow(Attr[i][
               not_visited[j]], betta)/sum;
          }
          else {
            P[j] = 0;
          }
        }
        random_res  = (rand() % 100) / 100.0;
        int best_p = 0;
        double sum_p = 0;
        for (size_t s = 0; s < M; s++) {
          sum_p += P[s];
          if(random_res < sum_p)
          {
            best_p = s;
            break;
          }
        }
        int new_town = not_visited[best_p];
        cur_path.push_back(new_town);
        cur_path_length += D[i][new_town];

        Delta_Teta[i][new_town] += Q/cur_path_length;

        i = new_town;
        not_visited.erase(not_visited.begin()+best_p);
      }
      cur_path_length += D[cur_path[cur_path.size()-1]][k];
      if (Lmin == -1 || cur_path_length < Lmin){
        Lmin = cur_path_length;
        Lp = cur_path;
      }
    }
    for (size_t k = 0; k < M; k++) {
      for (size_t h = 0; h < M; h++) {
        Teta[k][h] = Teta[k][h] * (1.0 - ro) + Delta_Teta[k][h];
        if (Teta[k][h] < teta_min){
            Teta[k][h] = teta_min;
        }
      }
    }
  }
return pair<int, vector<int>>(Lmin, Lp); }
```

3.3 Тестовые данные

В таблице 3.1 представлена заготовка данных для тестирования по методу черного ящика алгоритма полного перебора и муравьиного алгоритма.

Таблица 3.1 – Подготовленные тестовые данные

Матрица	Алгоритм полного перебора	Муравьиный алгоритм
0 9 8 2 9 9 0 1 7 4 8 1 0 7 3 2 7 7 0 8 9 4 3 8 0	Минимальная длина пути = 22 Минимальный путь: 0 3 1 2 4	Минимальная длина пути = 22 Минимальный путь: 0 3 1 2 4
0 7 7 7 0 3 7 3 0	Min path length = 17 Минимальный путь: 0 1 2	Min path length = 17 Минимальный путь: 0 1 2

3.4 Вывод

Реализованы алгоритм полного перебора и муравьиный алгоритм, подготовлены тесты для оценки качества их работы.

В результате тестирования рассматриваемые алгоритмы вывели на экран ожидаемые результаты тестов.

4 | Исследовательская часть

В данной части производится экспериментальное сравнение времени работы алгоритма полного перебора и муравьиного алгоритма. Также рассматривается параметризация муравьиного алгоритма.

4.1 Пример работы

На рисунках 4.1 и 4.2 представлены примеры работы программы на матрицах разного размера: выводится матрица смежности, длина самого короткого пути и сам путь для алгоритма полного перебора и для муравьиного алгоритма.

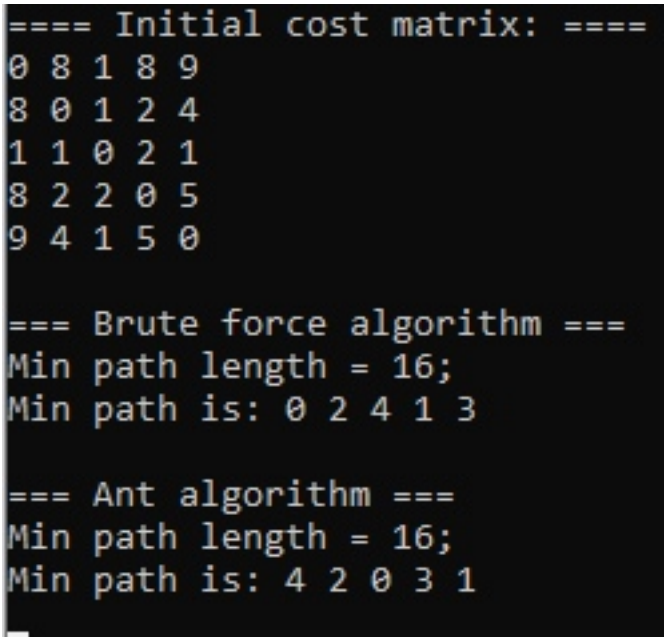

Рисунок 4.1: Пример работы программы
на матрице размера 5x5

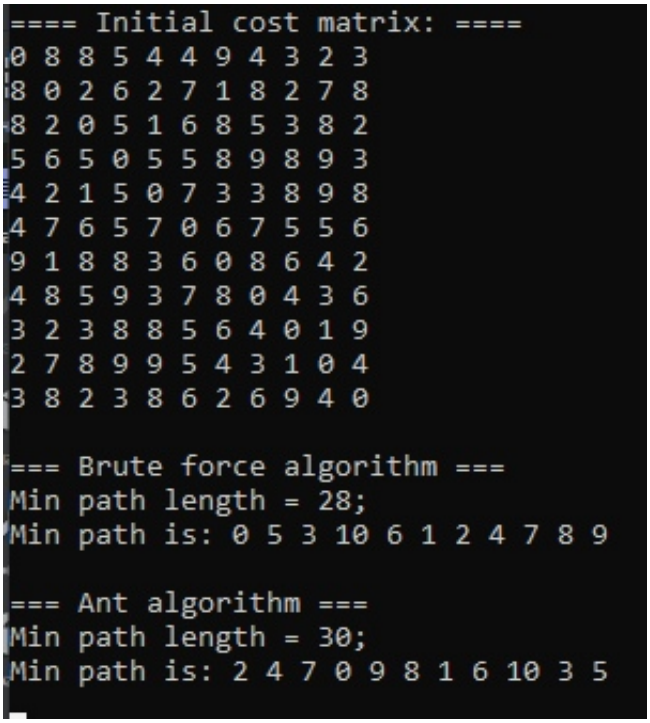

Рисунок 4.2: Пример работы программы
на матрице размера 11x11

4.2 Параметризация муравьиного алгоритма

Для исследования случайным образом была сгенерирована симметричная матрица расстояний размером 12x12. В каждом эксперименте фиксировались значения α, β, ρ и $tmax$. В течение экспериментов коэффициенты α, β, ρ менялись от 0 до 1 включительно с шагом 0.25, tmax от 10 до 310 с шагом 20.

Результатом эксперимента считается разница ($diff$) между длинной маршрута, рассчитанного при помощи алгоритма полного перебора и муравьиного алгоритма с определенным набором параметров. Каждый эксперимент проводился 2 раза и в результат записывалось значение длины лучшего найденного пути.

В таблице 4.1 представлены результаты эксперимента для параметров α, ρ и $tmax$, строки таблицы отсортированы по возрастанию столбца $diff$. Для наглядности первые 14 строк, давшие лучшие результаты, выделены голубым цветом.

Таблица 4.1 – Результаты параметризации муравьиного алгоритма

ρ	α	$tmax$	$diff$
0.000000	0.250000	90.000000	0.000000
0.000000	0.500000	250.000000	0.000000
0.000000	0.750000	290.000000	0.000000
0.000000	1.000000	330.000000	0.000000
0.250000	0.000000	210.000000	0.000000
0.250000	0.250000	210.000000	0.000000
0.250000	0.500000	170.000000	0.000000
0.250000	0.500000	330.000000	0.000000
0.250000	0.750000	330.000000	0.000000
0.500000	0.000000	250.000000	0.000000
0.500000	0.500000	330.000000	0.000000
0.500000	0.750000	170.000000	0.000000
0.750000	0.750000	50.000000	0.000000
1.000000	0.250000	50.000000	0.000000
0.000000	0.750000	210.000000	2.000000
1.000000	0.000000	210.000000	2.000000
1.000000	0.000000	290.000000	2.000000
0.000000	0.750000	250.000000	2.000000
1.000000	0.250000	90.000000	2.000000
1.000000	0.250000	130.000000	2.000000
1.000000	0.500000	130.000000	2.000000
1.000000	0.500000	250.000000	2.000000
1.000000	0.500000	290.000000	2.000000
0.000000	0.000000	250.000000	2.000000
0.000000	0.750000	330.000000	2.000000
0.000000	1.000000	50.000000	2.000000
0.000000	1.000000	210.000000	2.000000
0.000000	1.000000	250.000000	2.000000
0.000000	1.000000	290.000000	2.000000
0.000000	0.000000	290.000000	2.000000
0.250000	0.000000	170.000000	2.000000
0.250000	0.000000	290.000000	2.000000
0.000000	0.250000	210.000000	2.000000
0.250000	0.250000	290.000000	2.000000
0.250000	0.250000	330.000000	2.000000
0.000000	0.250000	250.000000	2.000000
0.250000	0.500000	210.000000	2.000000
0.250000	0.500000	250.000000	2.000000
0.000000	0.000000	170.000000	2.000000

ρ	α	$tmax$	$diff$
0.250000	0.000000	250.000000	2.000000
0.250000	0.500000	290.000000	2.000000
0.000000	0.250000	290.000000	2.000000
0.250000	0.750000	90.000000	2.000000
0.250000	0.750000	130.000000	2.000000
0.250000	0.750000	170.000000	2.000000
0.250000	0.750000	210.000000	2.000000
0.250000	0.750000	250.000000	2.000000
0.250000	0.750000	290.000000	2.000000
0.000000	0.250000	330.000000	2.000000
0.250000	1.000000	90.000000	2.000000
0.250000	1.000000	210.000000	2.000000
0.250000	1.000000	250.000000	2.000000
0.250000	1.000000	290.000000	2.000000
0.250000	1.000000	330.000000	2.000000
0.500000	0.000000	90.000000	2.000000
0.000000	0.500000	170.000000	2.000000
0.500000	0.000000	290.000000	2.000000
0.500000	0.250000	330.000000	2.000000
0.500000	0.500000	250.000000	2.000000
0.500000	0.500000	290.000000	2.000000
0.000000	0.000000	210.000000	2.000000
0.000000	0.500000	290.000000	2.000000
0.500000	1.000000	170.000000	2.000000
0.500000	1.000000	210.000000	2.000000
0.500000	1.000000	290.000000	2.000000
0.750000	0.000000	170.000000	2.000000
0.750000	0.250000	130.000000	2.000000
0.750000	0.250000	250.000000	2.000000
0.750000	0.500000	170.000000	2.000000
0.750000	0.500000	250.000000	2.000000
1.000000	0.250000	290.000000	3.000000
1.000000	0.250000	330.000000	3.000000
0.250000	0.250000	130.000000	3.000000
1.000000	0.500000	210.000000	3.000000
0.250000	0.250000	170.000000	3.000000
0.000000	0.250000	130.000000	3.000000
1.000000	0.750000	330.000000	3.000000
0.250000	0.250000	250.000000	3.000000
0.500000	0.000000	210.000000	3.000000
0.000000	0.500000	130.000000	3.000000
1.000000	0.750000	130.000000	3.000000

ρ	α	$tmax$	$diff$
1.000000	0.750000	290.000000	3.000000
0.000000	0.250000	170.000000	3.000000
0.500000	0.250000	90.000000	3.000000
0.500000	0.250000	250.000000	3.000000
0.250000	0.500000	50.000000	3.000000
0.500000	0.500000	50.000000	3.000000
0.500000	0.500000	90.000000	3.000000
0.500000	0.500000	130.000000	3.000000
0.250000	0.500000	130.000000	3.000000
0.000000	0.500000	210.000000	3.000000
0.000000	1.000000	90.000000	3.000000
0.000000	1.000000	170.000000	3.000000
0.500000	0.750000	250.000000	3.000000
0.500000	0.750000	290.000000	3.000000
0.000000	0.000000	90.000000	3.000000
0.000000	0.000000	330.000000	3.000000
0.500000	1.000000	250.000000	3.000000
0.250000	0.750000	50.000000	3.000000
0.500000	1.000000	330.000000	3.000000
0.000000	0.500000	330.000000	3.000000
0.750000	0.000000	210.000000	3.000000
0.750000	0.000000	290.000000	3.000000
0.750000	0.000000	330.000000	3.000000
0.000000	0.750000	50.000000	3.000000
0.750000	0.250000	170.000000	3.000000
0.750000	0.250000	210.000000	3.000000
0.250000	0.000000	90.000000	3.000000
0.750000	0.250000	290.000000	3.000000
0.750000	0.250000	330.000000	3.000000
0.750000	0.500000	130.000000	3.000000
0.000000	0.750000	90.000000	3.000000
0.000000	0.750000	130.000000	3.000000
0.750000	0.500000	290.000000	3.000000
0.750000	0.500000	330.000000	3.000000
0.000000	0.750000	170.000000	3.000000
0.750000	0.750000	330.000000	3.000000
1.000000	0.000000	90.000000	3.000000
0.000000	0.000000	50.000000	3.000000
1.000000	0.000000	250.000000	3.000000
0.250000	0.000000	330.000000	3.000000
1.000000	0.000000	330.000000	3.000000
0.250000	1.000000	130.000000	3.000000

ρ	α	$tmax$	$diff$
0.250000	1.000000	170.000000	3.000000
0.250000	0.250000	90.000000	3.000000
1.000000	0.250000	210.000000	3.000000
0.000000	0.500000	90.000000	4.000000
0.500000	0.250000	130.000000	4.000000
0.500000	0.250000	170.000000	4.000000
1.000000	0.250000	170.000000	4.000000
0.500000	0.250000	210.000000	4.000000
1.000000	0.250000	250.000000	4.000000
0.750000	0.000000	10.000000	4.000000
0.750000	0.000000	130.000000	4.000000
0.250000	0.750000	10.000000	4.000000
0.500000	0.250000	290.000000	4.000000
0.750000	0.000000	250.000000	4.000000
0.000000	0.000000	10.000000	4.000000
0.250000	0.250000	50.000000	4.000000
1.000000	0.750000	170.000000	4.000000
1.000000	0.750000	250.000000	4.000000
0.750000	0.250000	50.000000	4.000000
0.250000	0.500000	90.000000	4.000000
0.000000	1.000000	130.000000	4.000000
0.500000	0.500000	170.000000	4.000000
0.500000	0.500000	210.000000	4.000000
0.250000	0.000000	130.000000	4.000000
0.000000	0.250000	50.000000	4.000000
0.750000	0.500000	50.000000	4.000000
0.500000	0.000000	130.000000	4.000000
0.500000	0.750000	50.000000	4.000000
0.750000	0.500000	210.000000	4.000000
0.500000	0.750000	130.000000	4.000000
0.000000	0.000000	130.000000	4.000000
0.500000	0.750000	210.000000	4.000000
0.000000	0.500000	50.000000	4.000000
0.750000	0.750000	130.000000	4.000000
0.750000	0.750000	210.000000	4.000000
0.250000	1.000000	50.000000	4.000000
0.750000	1.000000	50.000000	4.000000
0.750000	1.000000	250.000000	4.000000
0.750000	1.000000	330.000000	4.000000
1.000000	0.000000	50.000000	4.000000
0.500000	0.750000	330.000000	4.000000
1.000000	0.000000	130.000000	4.000000
1.000000	0.000000	170.000000	4.000000
0.500000	1.000000	50.000000	4.000000
0.500000	1.000000	90.000000	4.000000

ρ	α	$tmax$	$diff$
0.500000	1.000000	130.000000	4.000000
0.500000	0.000000	330.000000	4.000000
0.750000	0.000000	90.000000	5.000000
1.000000	0.500000	170.000000	5.000000
0.750000	0.250000	10.000000	5.000000
0.750000	0.500000	10.000000	5.000000
0.750000	0.750000	90.000000	5.000000
1.000000	0.500000	330.000000	5.000000
0.500000	0.750000	90.000000	5.000000
0.750000	0.750000	170.000000	5.000000
1.000000	0.750000	210.000000	5.000000
0.750000	0.500000	90.000000	5.000000
0.750000	0.750000	250.000000	5.000000
0.750000	0.750000	290.000000	5.000000
1.000000	1.000000	50.000000	5.000000
1.000000	1.000000	210.000000	5.000000
1.000000	1.000000	250.000000	5.000000
1.000000	1.000000	290.000000	5.000000
1.000000	1.000000	330.000000	5.000000
0.500000	0.000000	170.000000	5.000000
0.250000	0.000000	50.000000	5.000000
0.750000	1.000000	130.000000	5.000000
0.750000	1.000000	210.000000	5.000000
0.500000	0.000000	50.000000	5.000000
0.750000	1.000000	290.000000	5.000000
0.750000	0.000000	50.000000	5.000000
0.750000	1.000000	170.000000	6.000000
1.000000	0.000000	10.000000	6.000000
0.250000	0.500000	10.000000	6.000000
1.000000	0.500000	50.000000	6.000000
1.000000	0.750000	50.000000	6.000000
1.000000	0.750000	90.000000	6.000000
1.000000	0.500000	90.000000	6.000000
0.750000	1.000000	90.000000	6.000000
0.500000	0.500000	10.000000	6.000000
0.000000	0.250000	10.000000	7.000000
0.250000	1.000000	10.000000	7.000000
0.750000	0.250000	90.000000	7.000000
1.000000	1.000000	130.000000	7.000000
1.000000	1.000000	170.000000	7.000000
0.500000	0.250000	50.000000	7.000000
0.500000	0.000000	10.000000	7.000000
0.000000	0.500000	10.000000	8.000000
1.000000	0.500000	10.000000	8.000000
1.000000	1.000000	10.000000	8.000000

ρ	α	$tmax$	$diff$
1.000000	0.250000	10.000000	8.000000
1.000000	1.000000	90.000000	8.000000
0.250000	0.250000	10.000000	8.000000
0.000000	0.750000	10.000000	9.000000
0.250000	0.000000	10.000000	9.000000
0.750000	1.000000	10.000000	10.000000
0.500000	1.000000	10.000000	10.000000
0.500000	0.250000	10.000000	10.000000
0.500000	0.750000	10.000000	11.000000
0.750000	0.750000	10.000000	11.000000
1.000000	0.750000	10.000000	11.000000
0.000000	1.000000	10.000000	12.000000

Из результатов таблицы 4.1 можно выделить оптимальные коэффициенты α, ρ, значение $tmax$; результат анализа отражен в таблице 4.2.

Таблица 4.2 – Оптимальные коэффициенты муравьиного алгоритма

ρ	α	$tmax$
0.25	0.5	170
0.25	0.5	330
0.5	0.5	330
0.5	0.75	170

4.3 Сравнение времени работы

На графиках (рисунки 4.3 и 4.4) представлено сравнение времени работы алгоритма полного перебора и муравьиного алгоритма с вычисленными параметрами $\rho = 0.25$, $\alpha = 0.5$, $tmax = 170$.

На первом графике (рисунок 4.3) исследование происходило на матрицах размера 2x2, 3x3, 4x4, ..., 11x11.

На втором графике (рисунок 4.4) исследование происходило на матрицах размера 2x2, 3x3, 4x4, ..., 8x8.

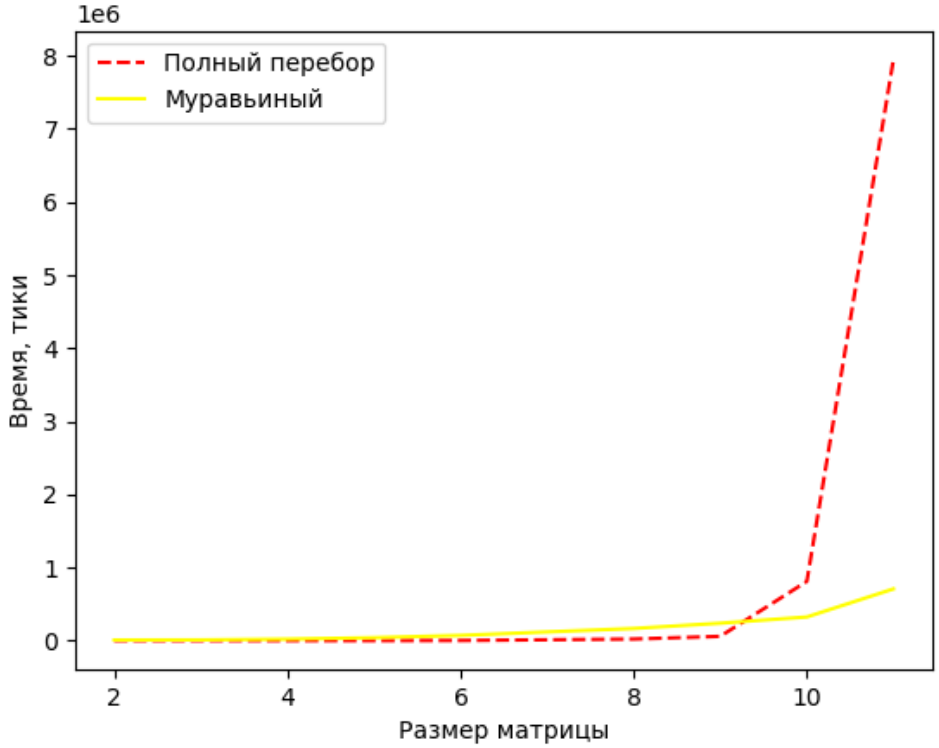

Рисунок 4.3 – Сравнение времени работы алгоритма полного перебора и муравьиного алгоритма на матрицах 2x2, ..., 11x11.

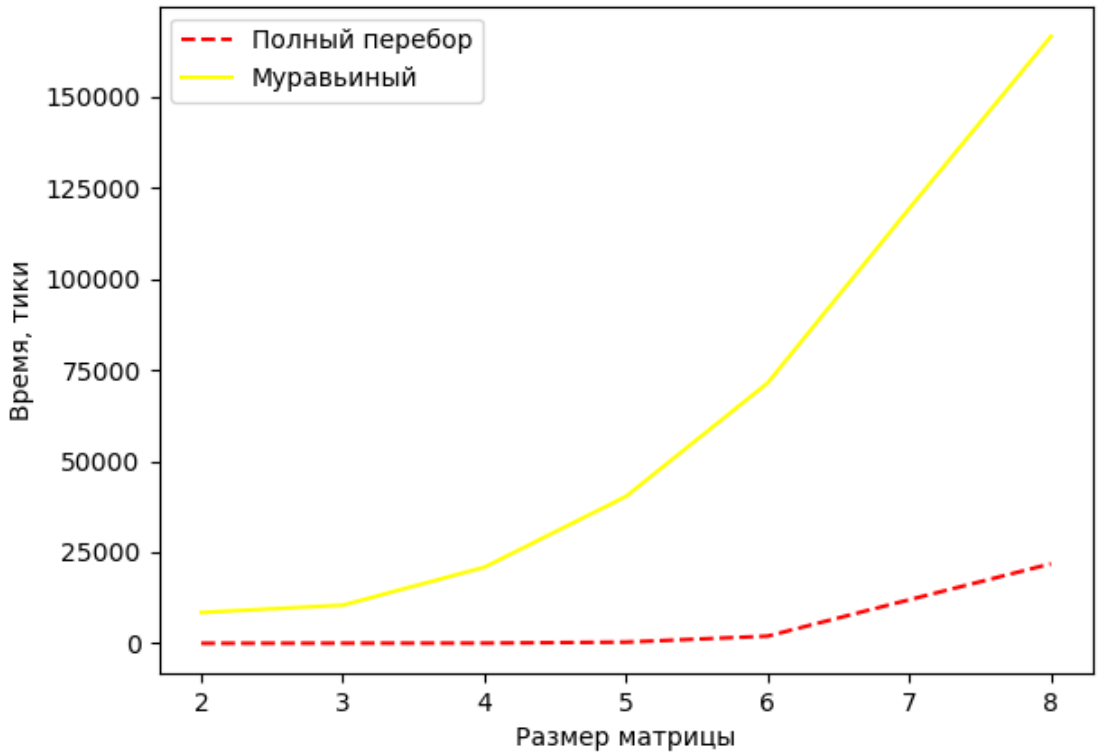

Рисунок 4.4 – Сравнение времени работы алгоритма полного перебора и муравьиного алгоритма на матрицах 2x2, ..., 8x8.

4.4 Вывод

В результате проведенных экспериментов были найдены оптимальные параметры для муравьиного алгоритма. Муравьиный алгоритм дает лучшие результаты при
$0.25 \leq \rho \leq 0.5$,
$0.5 \leq \alpha \leq 0.75$,
$170 \leq tmax \leq 330$.

При этом чем больше значение $tmax$, тем больше вероятность того, что будет найден самый короткий маршрут.

Также проведен сравнительный анализ времения работы муравьиного алгоритма и алгоритма полного перебора. По скорости муравьиный алгоритм сильно выигрывает у алгоритма полного перебора, на матрицах, размер которых превышает 10x10.

Алгоритм полного перебора стоит использовать, если:

1) матрицы имеют размер, меньший чем 10x10 элементов;

2) в задачах, где нужно гарантированно найти самый короткий путь.

Заключение

Задача коммивояжера – одна из важных задач комбинаторики и пользуется популярностью благодаря тому, что к ней сводится большое количество практических задач. Она была выбрана первой для решения с использованием подхода, заимствующего механизмы поведения муравьиной колонии. Позже этот подход нашел применение в решении других комбинаторных проблем, в числе которых задачи о назначениях, задача раскраски графа, задачи маршрутизации, задачи из областей data mining и распознавания образов и другие.

Проведенные опыты показали, что критерий выбора алгоритма для расчета маршрута будет зависеть от количества городов N:

1. при $N < 10$ поиск маршрута целесообразно осуществлять с помощью алгоритма полного перебора, так как он дает точное решение задачи коммивояжера, не проигрывая при этом по времени муравьиному алгоритму;

2. при $N > 10$ поиск маршрута целесообразно осуществлять с помощью муравьиного алгоритма, так как он работает быстрее алгоритма полного перебора, но при этом муравьиный алгоритм зависит от настроечных параметров, которые подбираются только исходя из экспериментов.

При выполнении лабораторной работы цель была достигнута: проведен сравнительный анализ метода полного перебора и эвристического метода на базе муравьиного алгоритма.

Также были выполнены следующие задачи:

1. реализованы методыы полного перебора и метод на базе муравьиного алгоритма для решения задачи коммивояжёра;

2. проведена параметризация метода на базе муравьиного алгоритма для выбранного класса задач, т.е. определены комбинации параметров, при которых метод даёт наилучшие результаты на выбранном классе задач;

3. произведено сравнение алгоритма полного перебора и муравьиного по скорости работы.

.

Литература

[1] Новиков Ф.А. Дискретная математика для программистов . – СПб.: Изд-во Питер, 2012. – 208 с.

[2] Майника Э. Алгоритмы оптимизации на сетях и графах. – СПб.: М.: Мир, 1981. – 323 с.

[3] Vector в C++: основные приемы эффективного использования [Электронный ресурс]. - Режим доступа: https://codelessons.ru/cplusplus/vektory-v-c-dlya-nachinayushhix.html (дата обращения: 03.12.2020)

[4] Полиномиальные алгоритмы и задачи [Электронный ресурс]. - Режим доступа: https://studfile.net/preview/5562531/page:41/ (дата обращения: 22.12.2020)