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 Бугаков И. С.

Группа ИУ7-54Б

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

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной лабораторной работе будут рассмотрены методы решения задачи коммивояжера: полный перебор и метод на основе муравьиного алгоритма.

Целью данной лабораторной работы является сравнительный анализ методов решения задачи коммивояжера [4].

Задачи лабораторной работы:

1) сформулировать постановку задачи коммивояжера;
2) описать методы решения задачи коммивояжера;
3) реализация указанных методов;
4) провести параметризацию метода на основе муравьиного алгоритма;
5) провести замеры и анализ временной эффективности указанных методов.

1 Аналитическая часть

Задача коммивояжера — задача оптимизации, заключающаяся в поиске в графе пути (или цикла) минимальной стоимости, проходящего через каждую вершину только один раз. Таким образом задача коммивояжера сводится к задаче поиска гамильтонова пути (или цикла) минимальной стоимости. Задача коммивояжера является np-трудной задачей, соответственно, на данный момент не доказана возможность ее решения за полиномиальное время [6].

В работе рассматриваются два возможных метода решения задачи коммивояжера: полный перебор и метод на основе муравьиного алгоритма.

1.1 Метод на основе полного перебора

1.1.1 Описание алгоритма

Данный алгоритм решает задачу методом «грубой силы»: перебор всех возможных перестановок вершин и поиск минимального веса пути, образованного вершинами в перестановке. Данный алгоритм для любого графа получит правильный ответ, однако его асимптотика позволяет использовать его лишь для графов с небольшим числом вершин [5].

1.1.2 Асимптотика

Число перестановок из n различных элементов составляет $n!$, соответственно, асимптотика алгоритма составляет $O(n!)$.

1.2 Метод на основе муравьиного алгоритма

1.2.1 Описание алгоритма

Алгоритм основывается на поведении муравьев при поиске пути от колонии к источнику пищи.

Изначально в каждой вершине графа располагается по одному муравью. Вводится матрица феромонов: $T = (\tau_{i,j})_{i,j=\overline{1,n}}$, где n — число вершин в графе. Каждое $\tau_{i,j}$ обозначает количество феромона на ребре из вершины i в вершину j. Для исключения возможности обнуления вероятности перехода в еще не посещенную вершину вводится некоторое минимальное количество феромона q_{min}. Изначально также полагается $\tau_{i,j} = q_{min}, \forall i, j = \overline{1, n}$.

В каждый из d дней каждый муравей пытается построить маршрут. Если муравей ока-

зывается в тупике, т. е. не может из вершины, в которой он сейчас находится перейти в еще не посещенную вершину, или найденный им маршрут имеет большую стоимость, чем найденный им ранее, такой маршрут не учитывается в обновлении феромонов. Иначе в соответствии с найденными маршрутами обновляются феромоны. В системе с элитными муравьями, в конце каждого дня вносится дополнительная доза феромона на маршруты минимальной длины. По истечении d дней выбирается наилучший маршрут.

Вероятность или желание перехода муравья k из текущей вершины i в смежную с ней вершину j в день t определяется по формуле 1.1.

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

где $\eta_{i,j} = \frac{1}{d_{i,j}}$ — величина обратно пропорциональная весу ребра, $\tau_{i,j}(t)$ — количество феромона на ребре из вершины i в вершину j в день t, $\alpha \in [0,1]$ — величина, называемая коэффициентом «жадности», $\beta = 1 - \alpha$ — величина, называемая коэффициентом «стадности», $Y_k(t)$ - множество ребер входящих в маршрут муравья на данный момент.

Таким образом выбор следующей вершины осуществляется среди всех еще не посещенных, смежных с данной. Выбор осуществляется случайным образом, однако, «случайность» распределена в соответствии, с ранее посчитанными вероятностями.

В конце дня необходимо обновить феромон на каждом ребре. Обновление осуществляется в соответствии с формулой 1.2.

$$\tau_{i,j} = (1 - \rho) \cdot \tau_{i,j}(t) + \Delta\tau_{i,j}(t) \tag{1.2}$$

где ρ - коэффициент испарения, $\tau_{i,j}(t)$ — количество феромона на ребре из вершины i в вершину j в день t, $\Delta\tau_{i,j}(t)$ — изменение феромона на ребре из вершины i в вершину j в день t в связи с полученными муравьями путями. $\Delta\tau_{i,j}(t)$ определяется в соответствии с формулами 1.3, 1.4.

$$\Delta\tau_{i,j}(t) = \sum_{k=1}^{n} \Delta\tau_{i,j}(t,k) \tag{1.3}$$

$$\Delta\tau_{i,j}(t,k) = \begin{cases} 0, & \text{если муравей } k \text{ на итерации } t \text{ не проходил ребро из вершины } i \text{ в } j \\ \frac{Q}{L_k(t)}, & \text{иначе} \end{cases} \tag{1.4}$$

где Q — дневной запас феромона, $L_k(t)$ — длина пути построенного муравьем k в день t.

Все обновления производятся, если найденный в текущий день путь короче, чем все пути найденные до этого. Иначе обновления производятся в соответствии с предыдущим кратчайшим путем.

Метод на основе муравьиного алгоритма имеет меньшую асимптотическую сложность, чем полный перебор за $O(n!)$, однако данный метод не всегда находит точный ответ [3].

Вывод

В данной части были рассмотрены два метода решения задачи коммивояжера: метод на основе полного перебора и метод на основе муравьиного алгоритма.

2 Конструкторская часть

2.1 Схемы алгоритмов

Ниже приведены схемы алгоритмов, упомянутых в аналитической части:

— на рисунке 2.1 — схема полного перебора для поиска гамильтонова пути;

— на рисунке 2.2 — схема муравьиного алгоритма.

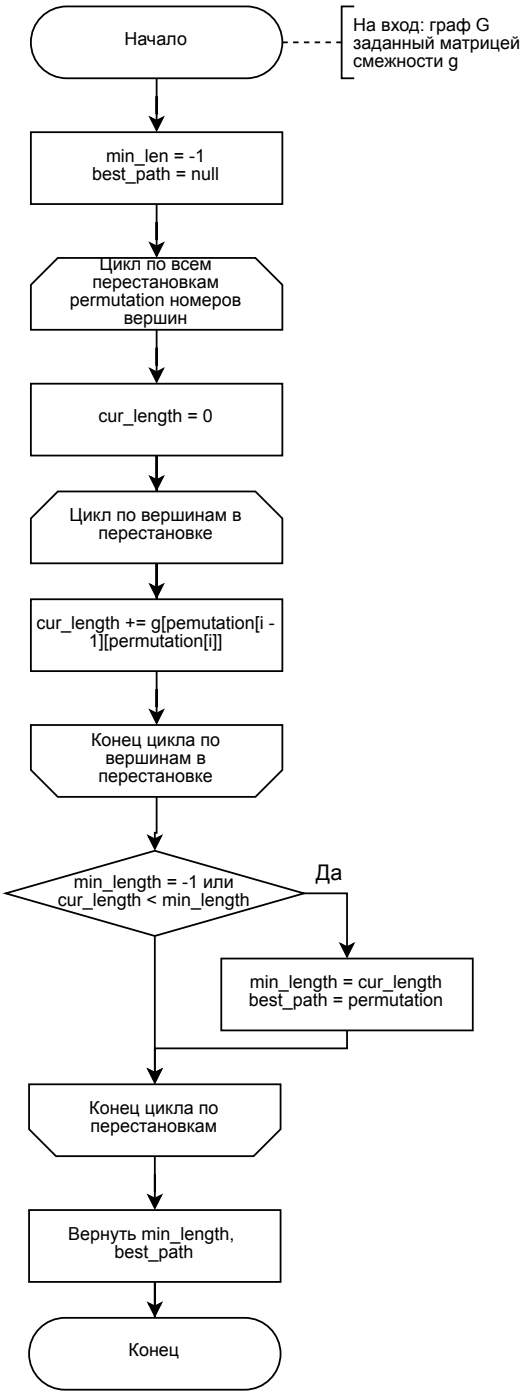

Рисунок 2.1 — Схема алгоритма полного перебора

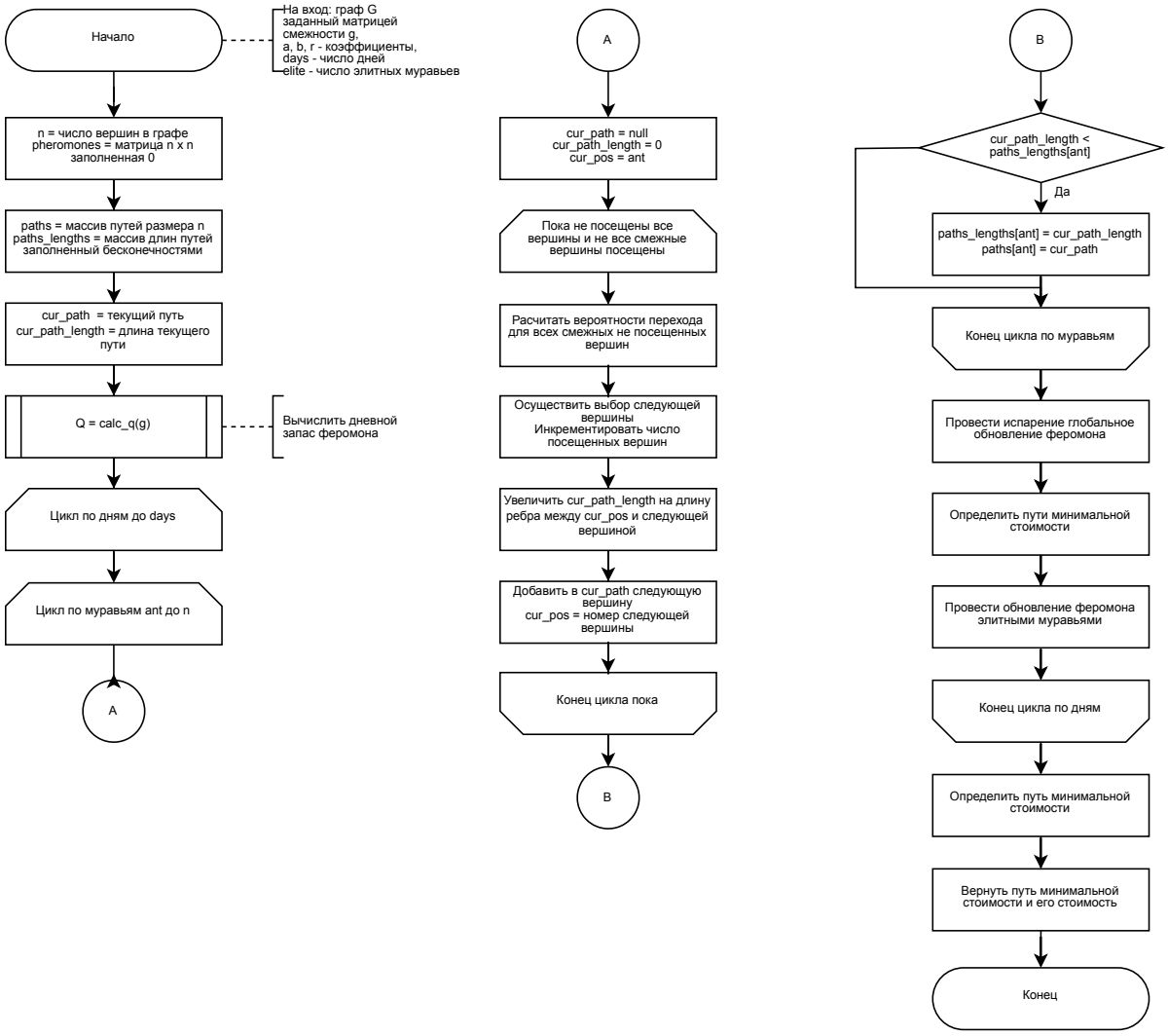

Рисунок 2.2 — Схема муравьиного алгоритма

Вывод

На основе аналитической части построены схемы алгоритмов решения задачи коммивояжера.

3 Технологическая часть

3.1 Выбор языка программирования

Для реализации указанных алгоритмов был выбран язык C++, т. к. данный язык предоставляет возможность работать с динамической памятью с помощью контейнерных классов.

Для визуализации данных использовался язык Python, т. к. данный язык обладает широким выбором библиотек для выполнения этой задачи. В лабораторной работе использовалась библиотека matplotlib [2].

В качестве IDE был выбран Clion, т. к. обладает функциями подсветки синтаксиса и автодополнения для обоих указанных языков. Также в данной IDE реализована поддержка средства автоматизированной сборки проектов cmake.

3.2 Реализации алгоритмов

В листингах ниже представлены реализации указанных алгоритмов:

— в листинге 3.1 — реализация алгоритма полного перебора;

— в листинге 3.2 — функция вычисления дневной дозы феромона для данного графа;

— в листинге 3.3 — реализация муравьиного алгоритма;

— в листинге 3.4 — функция проверки графа на связность;

— в листинге 3.5 — функция генерации следующей в лексикографическом порядке перестановки.

Листинг 3.1 — Алгоритм полного перебора

```
pair<double, vector<int>> complete_bust(const vector<vector<double>>
    &g) {
  int n = g.size();
  vector<int> permutation(n);

  vector<int> path;
  double sum = 0, cur_ans = -1;

  for (int i = 0; i < n; ++i)
  permutation[i] = i;

  do {
    for (int i = 1; i < permutation.size(); ++i) {
      sum += g[permutation[i - 1]][permutation[i]];
    }
    if (cur_ans == -1 || cur_ans > sum) {
      path = permutation;
      cur_ans = sum;
    }
    sum = 0;
  } while (next_permutation_(permutation.begin(), permutation.end()))
     ;
  return {cur_ans, path};
}
```

Листинг 3.2 — Вычисление дневной дозы феромона

```
double calc_q(const vector<vector<double>> &g) {
  double q = 0;
  int count = 0;
  for (const auto &i: g)
  for (double j: i)
  if (j < INF) {
    q += j;
    ++count;
  }
  return q / count;
}
```

Листинг 3.3 — Муравьиный алгоритм

```
pair<double, vector<int>>
ant_algorithm(const vector<vector<double>> &g, int days_cnt, double a
   , double b, double r, int elite_cnt) {
  int n = g.size();
  double q = calc_q(g), min_q = 1;
  vector<int> ans(n);
  vector<vector<int>> paths(n, vector<int>(0));
  vector<double> paths_lengths(n, INF);
  vector<vector<double>> pheromones(n, vector<double>(n, min_q));
  vector<bool> visited;
  int visited_cnt, cur_pos;
  vector<double> wishes(n);
  vector<int> cur_path;
  double total_wish, probability, best_path_length, ans_path_length =
      INF, cur_length;
  for (int t = 0; t < days_cnt; ++t) {
    for (int ant = 0; ant < n; ++ant) {
      cur_path.clear();
      cur_length = 0;
      visited.clear();
      visited.resize(n, false);
      cur_pos = ant;
      cur_path.push_back(ant);
      visited[cur_pos] = true;
      visited_cnt = 1;
      while (visited_cnt < n) {
        total_wish = 0;
        for (int i = 0; i < n; ++i)
        total_wish += ((visited[i] || g[cur_pos][i] >= INF - EPS) ? 0
            : pow(1 / g[cur_pos][i], a) * pow(pheromones[cur_pos][i],
            b));
        if (abs(total_wish) < EPS) {
          cur_length=INF;
          break;
        }
        for (int i = 0; i < n; ++i)
        wishes[i] = ((visited[i] || g[cur_pos][i] >= INF - EPS) ? 0 :
            pow(1 / g[cur_pos][i], a) * pow(pheromones[cur_pos][i], b
           ) / total_wish);
        probability = (double) rnd() / rnd.max();
```

```
      for (int i = 0; i < n; ++i) {
        if (probability >= wishes[i])
        probability -= wishes[i];
        else {
          cur_length += g[cur_pos][i];
          cur_pos = i;
          cur_path.push_back(cur_pos);
          visited_cnt++;
          visited[cur_pos] = true;
        }
      }
    }
    if (cur_length < paths_lengths[ant]) {
      paths_lengths[ant] = cur_length;
      paths[ant] = cur_path;
    }
  }
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      pheromones[i][j] *= (1 - r);
      if (pheromones[i][j] < min_q)
        pheromones[i][j] = min_q;
    }
  }
  for (int i = 0; i < n; ++i)
  for (int j = 1; j < paths[i].size(); ++j) {
    pheromones[paths[i][j - 1]][paths[i][j]] += q / paths_lengths[i
       ];
    pheromones[paths[i][j]][paths[i][j - 1]] += q / paths_lengths[i
       ];
  }
  best_path_length = paths_lengths[0];
  for (int i = 0; i < n; ++i) {
    best_path_length = min(best_path_length, paths_lengths[i]);
  }
  for (int i = 0; i < n; ++i) {
    if (abs(paths_lengths[i] - best_path_length) < EPS) {
      for (int j = 1; j < paths[i].size(); ++j) {
        pheromones[paths[i][j - 1]][paths[i][j]] += elite_cnt * q /
            paths_lengths[i];
        pheromones[paths[i][j]][paths[i][j - 1]] += elite_cnt * q /
```

```
                paths_lengths[i];
          }
        }
      }
    }
    ans_path_length = paths_lengths[0];
    ans = paths[0];
    for (int i = 0; i < n; ++i) {
      paths_lengths[i]);
      if (paths_lengths[i] < ans_path_length) {
        ans_path_length = paths_lengths[i];
        ans = paths[i];
      }
    }
    return {ans_path_length, ans};
}
```

Листинг 3.4 — Функция проверки графа на связность и dfs

```
bool check_connectivity(const vector<vector<double>> &g) {
  vector<bool> used(g.size());
  dfs(g, used, 0);
  for (int i = 0; i < g.size(); ++i)
  if (!used[i])
  return false;
  return true;
}

void dfs(const vector<vector<double>> &g, vector<bool> &used, int v)
   {
  used[v] = true;
  for (int u = 0; u < g.size(); ++u)
  if (!used[u] && g[v][u] < INF)
  dfs(g, used, u);
}
```

Листинг 3.5 — Функция генерации следующей в лексикографическом порядке перестановки

```
bool next_permutation_(vector<int>& nums) {
  int n = nums.size();
  if (n <= 1) return false;

  int i = n - 2;
  while (i >= 0 && nums[i] >= nums[i + 1]) {
    --i;
  }

  if (i < 0) {
    std::reverse(nums.begin(), nums.end());
    return false;
  }

  int j = n - 1;
  while (nums[j] <= nums[i]) {
    --j;
  }

  std::swap(nums[i], nums[j]);

  std::reverse(nums.begin() + i + 1, nums.end());

  return true;
}
```

3.3 Тестирование

Тестовые случаи для тестирования функции проверки графа на связность представлены в таблице 3.1.

Таблица 3.1 — Тестовые случаи для функции проверки связности

Класс эквивалентности	Входные данные	Ожидаемые выходные данные	Полученные выходные данные
Несвязный граф	$\begin{pmatrix} 0 & 10 & INF & INF & INF \\ 10 & 0 & INF & INF & INF \\ INF & INF & 0 & 15 & 12 \\ INF & INF & 15 & 0 & 13 \\ INF & INF & 12 & 13 & 0 \end{pmatrix}$	false	false
Связный граф	$\begin{pmatrix} 0 & 12 & 13 \\ 12 & 0 & 15 \\ 13 & 15 & 0 \end{pmatrix}$	true	true

Тестовые данные для функции вычисления начального количества феромона представлены в таблице 3.2.

Таблица 3.2 — Тестовые случаи для вычисления начального количества феромона

Класс эквивалентности	Входные данные	Ожидаемые выходные данные	Полученные выходные данные
Несвязный граф	$\begin{pmatrix} 0 & 25 & INF & INF & INF \\ 25 & 0 & INF & INF & INF \\ INF & INF & 0 & 15 & 12 \\ INF & INF & 15 & 0 & 13 \\ INF & INF & 12 & 13 & 0 \end{pmatrix}$	10	10
Связный граф	$\begin{pmatrix} 0 & 12 & 13 \\ 12 & 0 & 20 \\ 13 & 20 & 0 \end{pmatrix}$	10	10

Тестовые данные для проверки метода полного перебора представлены в таблице 3.3.

Таблица 3.3 — Тестовые случаи для метода полного перебора

Класс эквивалентности	Входные данные	Ожидаемые выходные данные	Полученные выходные данные
Полный граф	$\begin{pmatrix} 0 & 8 & 10 & 9 \\ 8 & 0 & 5 & 3 \\ 10 & 5 & 0 & 2 \\ 9 & 3 & 2 & 0 \end{pmatrix}$	Длина: 13 Путь: 1, 2, 4, 3	Длина: 13 Путь: 1, 2, 4, 3
Связный не полный граф	$\begin{pmatrix} 0 & 6 & INF & 5 & INF \\ 6 & 0 & 7 & 8 & INF \\ INF & 7 & 0 & INF & 1 \\ 5 & 8 & INF & 0 & 13 \\ INF & INF & 1 & 13 & 0 \end{pmatrix}$	Длина: 19 Путь: 4, 1, 2, 3, 5	Длина: 19 Путь: 4, 1, 2, 3, 5

Тестовые данные для метода на основе муравьиного алгоритма представлены в таблице 3.4.

Таблица 3.4 — Тестовые случаи для муравьиного алгоритма

Класс эквивалентности	Входные данные	Ожидаемые выходные данные	Полученные выходные данные
Полный граф	$\begin{pmatrix} 0 & 8 & 10 & 9 \\ 8 & 0 & 5 & 3 \\ 10 & 5 & 0 & 2 \\ 9 & 3 & 2 & 0 \end{pmatrix}$	Длина: 13 Путь: 1, 2, 4, 3	Длина: 13 Путь: 1, 2, 4, 3
Связный не полный граф	$\begin{pmatrix} 0 & 6 & INF & 5 & INF \\ 6 & 0 & 7 & 8 & INF \\ INF & 7 & 0 & INF & 1 \\ 5 & 8 & INF & 0 & 13 \\ INF & INF & 1 & 13 & 0 \end{pmatrix}$	Длина: 19 Путь: 4, 1, 2, 3, 5	Длина: 19 Путь: 4, 1, 2, 3, 5

Для проведения тестирования использовалась библиотека Google Tests [1].

Вывод

Были реализованы методы решения задачи коммивояжера. Проведено тестирование метода на основе полного перебора, а также вспомогательных функций. Все тесты были успешно пройдены.

4 Исследовательская часть

4.1 Характеристики устройства

Исследования проводились на машине со следующими характеристиками:

— процессор Intel(R) Core(TM) i5-10210U, тактовая частота 1.60 ГГц;

— оперативная память: 16 ГБ;

— операционная система: Ubuntu 22.04.4 LTS.

4.2 Параметризация

Для муравьиного алгоритма была произведена параметризация для определения наиболее оптимальных, в смысле минимизации ошибки ответа алгоритма по сравнению с точным значением, значений параметров:

— α — параметр, определяющий «жадность» решения;

— $\beta = 1 - \alpha$ — параметр, определяющий «стадность» решения;

— ρ — коэффициент испарения феромона;

— e — число элитных муравьев.

Параметризация проводилась с использованием полных неориентированных графов из 10 вершин, привязанных к карте местности Африки, для поиска незамкнутого маршрута. Пример графа приведен на рисунке 4.1.

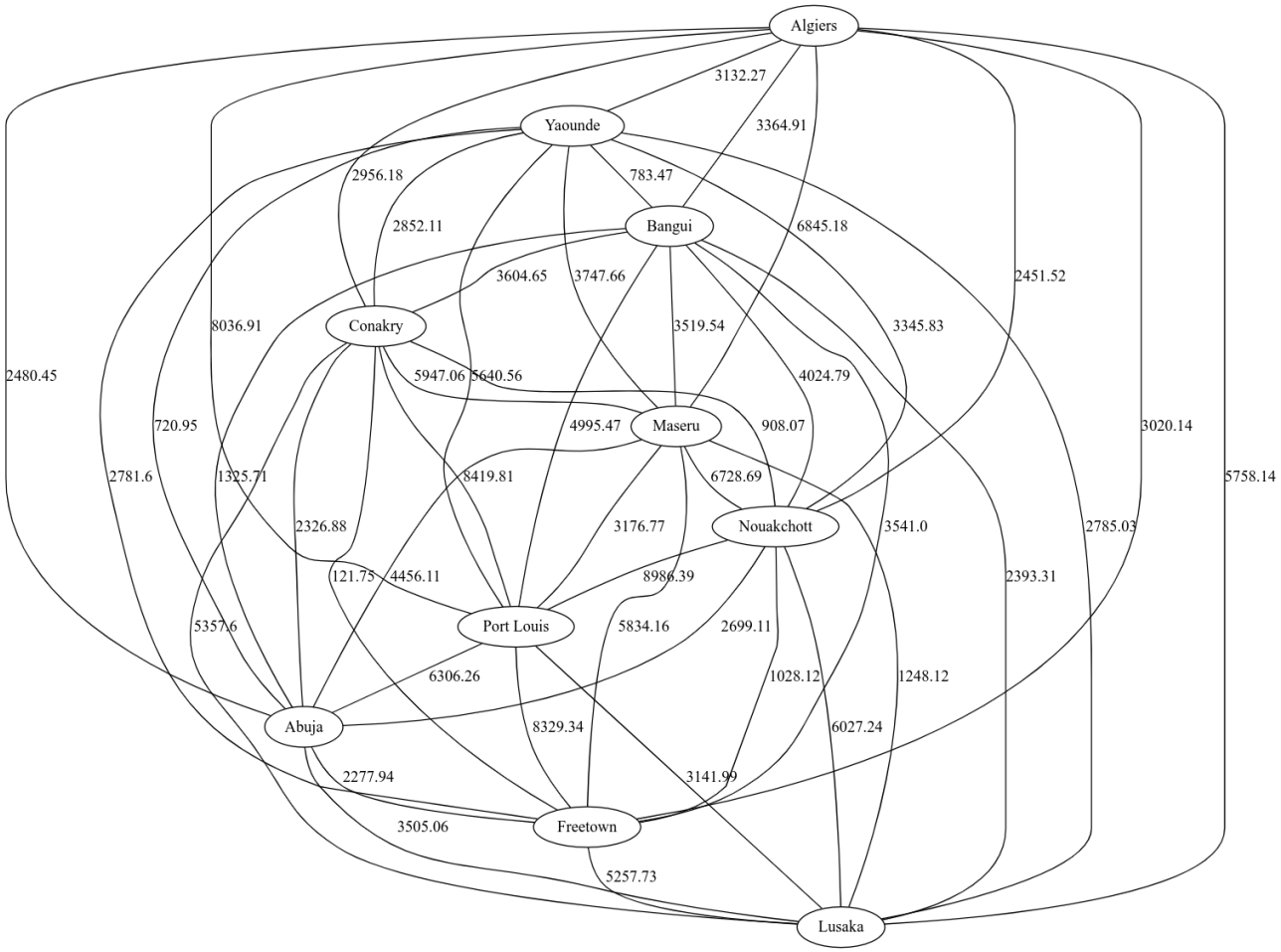

Рисунок 4.1 — Пример графа для параметризации

Матрицы смежности для графов, на которых производилась параметризация:

— граф 1:

$$\begin{pmatrix}
0 & 8986.4 & 2451.5 & 3345.8 & 2699.1 & 4024.8 & 1028.1 & 6027.2 & 908.1 & 6728.7 \\
8986.4 & 0 & 8036.9 & 5640.6 & 6306.3 & 4995.5 & 8329.3 & 3142.0 & 8419.8 & 3176.8 \\
2451.5 & 8036.9 & 0 & 3132.3 & 2480.5 & 3364.9 & 3020.1 & 5758.1 & 2956.2 & 6845.2 \\
3345.8 & 5640.6 & 3132.3 & 0 & 720.9 & 783.5 & 2781.6 & 2785.0 & 2852.1 & 3747.7 \\
2699.1 & 6306.3 & 2480.5 & 720.9 & 0 & 1325.7 & 2277.9 & 3505.1 & 2326.9 & 4456.1 \\
4024.8 & 4995.5 & 3364.9 & 783.5 & 1325.7 & 0 & 3541.0 & 2393.3 & 3604.7 & 3519.5 \\
1028.1 & 8329.3 & 3020.1 & 2781.6 & 2277.9 & 3541.0 & 0 & 5257.7 & 121.7 & 5834.2 \\
6027.2 & 3142.0 & 5758.1 & 2785.0 & 3505.1 & 2393.3 & 5257.7 & 0 & 5357.6 & 1248.1 \\
908.1 & 8419.8 & 2956.2 & 2852.1 & 2326.9 & 3604.7 & 121.7 & 5357.6 & 0 & 5947.1 \\
6728.7 & 3176.8 & 6845.2 & 3747.7 & 4456.1 & 3519.5 & 5834.2 & 1248.1 & 5947.1 & 0
\end{pmatrix}$$

— граф 2:

$$\begin{pmatrix}
0 & 8986.4 & 1626.0 & 1655.4 & 2242.1 & 5979.2 & 465.4 & 6405.7 & 7972.6 & 6448.0 \\
8986.4 & 0 & 8740.2 & 7337.7 & 6769.7 & 3728.3 & 8867.0 & 2807.2 & 1062.8 & 2604.1 \\
1626.0 & 8740.2 & 0 & 1983.9 & 2708.1 & 5250.7 & 2056.5 & 6556.7 & 7843.7 & 6455.1 \\
1655.4 & 7337.7 & 1983.9 & 0 & 731.3 & 4412.8 & 1637.0 & 4826.3 & 6335.1 & 4827.4 \\
2242.1 & 6769.7 & 2708.1 & 731.3 & 0 & 4146.9 & 2099.2 & 4166.7 & 5741.7 & 4208.9 \\
5979.2 & 3728.3 & 5250.7 & 4412.8 & 4146.9 & 0 & 6044.9 & 3006.1 & 3140.4 & 2561.5 \\
465.4 & 8867.0 & 2056.5 & 1637.0 & 2099.2 & 6044.9 & 0 & 6211.2 & 7832.7 & 6291.9 \\
6405.7 & 2807.2 & 6556.7 & 4826.3 & 4166.7 & 3006.1 & 6211.2 & 0 & 1749.5 & 481.7 \\
7972.6 & 1062.8 & 7843.7 & 6335.1 & 5741.7 & 3140.4 & 7832.7 & 1749.5 & 0 & 1548.1 \\
6448.0 & 2604.1 & 6455.1 & 4827.4 & 4208.9 & 2561.5 & 6291.9 & 481.7 & 1548.1 & 0
\end{pmatrix}$$

— граф 3:

$$\begin{pmatrix}
0 & 8986.4 & 5477.6 & 8203.4 & 4188.2 & 4951.6 & 5979.2 & 2154.7 & 6405.7 & 628.8 \\
8986.4 & 0 & 3543.3 & 1618.5 & 4863.2 & 5812.3 & 3728.3 & 6883.9 & 2807.2 & 8701.8 \\
5477.6 & 3543.3 & 0 & 2834.2 & 1658.3 & 3175.9 & 1544.1 & 3461.7 & 1680.4 & 5262.4 \\
8203.4 & 1618.5 & 2834.2 & 0 & 4449.5 & 4339.2 & 2386.5 & 6281.7 & 2995.5 & 8056.9 \\
4188.2 & 4863.2 & 1658.3 & 4449.5 & 0 & 3839.9 & 2984.8 & 2040.4 & 2223.3 & 3846.4 \\
4951.6 & 5812.3 & 3175.9 & 4339.2 & 3839.9 & 0 & 2118.5 & 4073.2 & 4850.3 & 5159.4 \\
5979.2 & 3728.3 & 1544.1 & 2386.5 & 2984.8 & 2118.5 & 0 & 4310.1 & 3006.1 & 5941.2 \\
2154.7 & 6883.9 & 3461.7 & 6281.7 & 2040.4 & 4073.2 & 4310.1 & 0 & 4251.8 & 1819.0 \\
6405.7 & 2807.2 & 1680.4 & 2995.5 & 2223.3 & 4850.3 & 3006.1 & 4251.8 & 0 & 6030.9 \\
628.8 & 8701.8 & 5262.4 & 8056.9 & 3846.4 & 5159.4 & 5941.2 & 1819.0 & 6030.9 & 0
\end{pmatrix}$$

Результаты параметризации представлены в приложении А.

4.2.1 Замеры времени

Замеры производились с использованием полных графов с числом вершин от 2 до 11. Параметры муравьиного алгоритма: $\alpha = 0.1$, $\beta = 0.9$, число дней $= 100$, $\rho = 0.1$, число элитных муравьев $= 4$. Полученные зависимости представлены на рисунке 4.2.

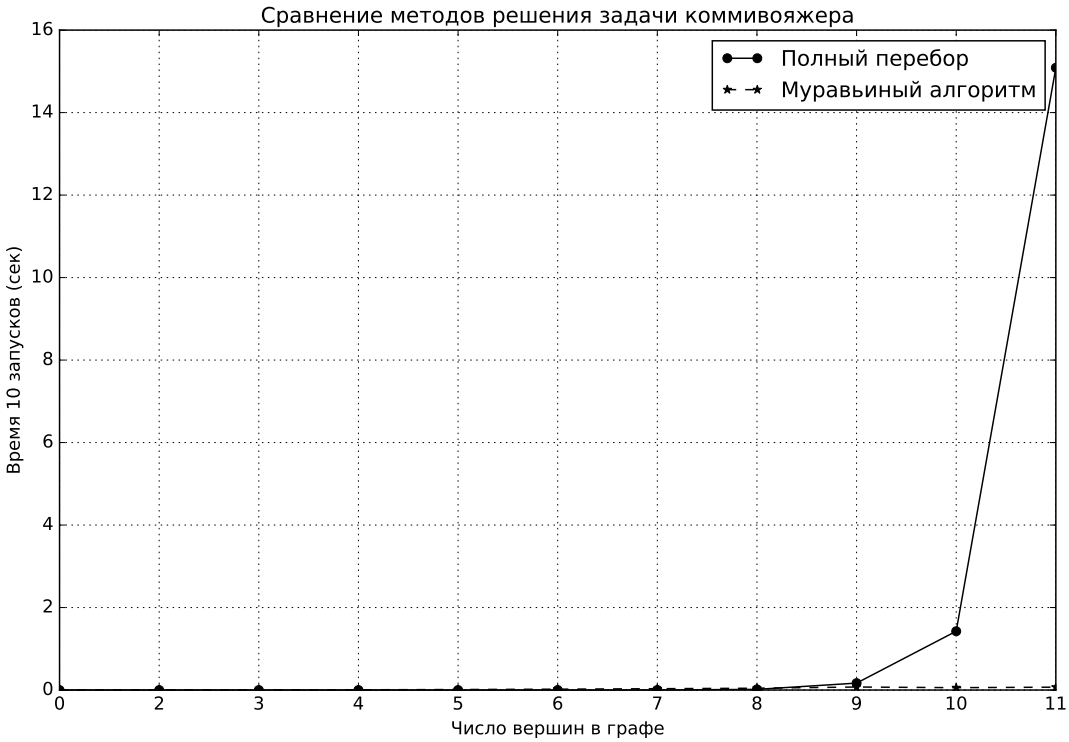

Рисунок 4.2 — Графики зависимости времени решения задачи коммивояжера от числа вершин в графе

4.3 Вывод

В результате параметризации были определены следующие оптимальные параметры муравьиного алгоритма:

— $\alpha \in [0,\ 0.4]$;

— $\beta \in [0.6,\ 0.1]$;

— $\rho \in [0.1,\ 0.2]$;

— число дней > 100;

— число элитных муравьев $\in [5, 15]$.

По результатам замеров времени выполнения было выявлено, что муравьиный алгоритм начинает превосходить в скорости выполнения полный перебор, для графов с числом вершин не менее 9.

ЗАКЛЮЧЕНИЕ

Целью данной работы являлся анализ методов решения задачи коммивояжера: полный перебор и метод на основе муравьиного алгоритма. В ходе лабораторной работы были выполнены следующие задачи:

1) сформулирована постановка задачи коммивояжера;
2) описаны методы решения задачи коммивояжера;
3) реализованы указанные методы;
4) проведена параметризация метода на основе муравьиного алгоритма и выявлены оптимальные параметры;
5) проведены замеры и анализ временной эффективности указанных методов.

По итогам анализа было выявлено, что оптимальными для муравьиного алгоритма являются следующие параметры:

— $\alpha \in [0,\ 0.4]$;
— $\beta \in [0.6,\ 0.1]$;
— $\rho \in [0.1,\ 0.2]$;
— число дней > 100;
— число элитных муравьев $\in [5, 15]$.

В результате анализа временных затрат было определено, что метод на основе муравьиного алгоритма превосходит полный перебор в скорости выполнения для графов с числом вершин не менее 9.

Поставленная цель исследования методов решения задачи коммивояжера была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Googletest user's guide [Электронный ресурс]. Режим доступа: `https://google.github.io/googletest/`. Дата обращения 8.10.2024.
2. matplotlib [Электронный ресурс]. Режим доступа: `https://matplotlib.org/`. Дата обращения 24.10.2024.
3. Stützle T. Dorigo M. *Ant Colony Optimization*. MIT Press, 2004. 319 с.
4. Lars Kjelldahl. Real-time large scale fluids for games. 2008.
5. Левитин А. В. *Алгоритмы. Введение в разработку и анализ.* Вильямс, Москва, 2006. 302 с.
6. Мудров В. И. *Задача о коммивояжере*. Знание, Москва, 1969. 62 с.

ПРИЛОЖЕНИЕ А