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 №7 по курсу "Анализ алгоритмов"

Тема Методы решения задачи коммивояжера

Студент Морозов Д.В.

Группа ИУ7-52Б

Оценка (баллы)

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

Москва — 2023 г.

Оглавление

Введение

Муравьиный алгоритм — алгоритм для нахождения приближённых решений задач оптимизации на графах, таких, как задача коммивояжера, транспортная задача и аналогичных задач поиска маршрутов на графах. Муравья нельзя назвать сообразительным. Отдельный муравей не в состоянии принять ни малейшего решения. Дело в том, что он устроен крайне примитивно: все его действия сводятся к элементарным реакциям на окружающую обстановку и своих собратьев. Муравей не способен анализировать, делать выводы и искать решения.

Эти факты, однако, никак не согласуются с успешностью муравьев как вида. Они существуют на планете более 100 миллионов лет, строят огромные жилища, обеспечивают их всем необходимым и даже ведут настоящие войны. В сравнении с полной беспомощностью отдельных особей, достижения муравьев кажутся немыслимыми.

Добиться таких успехов муравьи способны благодаря своей социальности. Они живут только в коллективах — колониях. Все муравьи колонии формируют так называемый роевой интеллект. Особи, составляющие колонию, не должны быть умными: они должны лишь взаимодействовать по определенным правилам, тогда колония целиком будет эффективна.

В колонии нет доминирующих особей, нет начальников и подчиненных, нет лидеров, которые раздают указания и координируют действия. Колония является полностью самоорганизующейся. Каждый из муравьев обладает информацией только о локальной обстановке, не один из них не имеет представления обо всей ситуации в целом — только о том, что узнал сам или от своих сородичей, явно или неявно. На неявных взаимодействиях муравьев, называемых стигмергией, основаны механизмы поиска кратчайшего пути от муравейника до источника пищи.

Каждый раз проходя от муравейника до пищи и обратно, муравьи оставляют за собой дорожку феромонов. Другие муравьи, почувствовав такие следы на земле, будут инстинктивно устремляться к нему. Поскольку эти муравьи тоже оставляют за собой дорожки феромонов, то чем больше муравьев проходит по определенному пути, тем более привлекательным он

становится для их сородичей. Чем короче путь до источника пищи, тем меньше времени требуется муравьям на него, следовательно, тем быстрее оставленные на нем следы становятся заметными.

В 1992 году в своей диссертации Марко Дориго (Marco Dorigo) предложил заимствовать описанный природный механизм для решения задач оптимизации [1]. Имитируя поведение колонии муравьев в природе, муравьиные алгоритмы используют многоагентные системы. Они эффективны при решении сложных комбинаторных задач — таких, например, как задача коммивояжера [2].

Целью лабораторной работы является описание и реализация решения задачи коммивояжера с помощью полного перебора и с помощью муравьиного алгоритма.

Задачи данной лабораторной:

1) описание алгоритма полного перебора и муравьиного алгоритма;

2) реализация алгоритмов;

3) параметризация муравьиного алгоритма на двух классах данных;

4) сравнение затрат реализаций по времени выполнения.

1 Аналитическая часть

1.1 Задача коммивояжера

Коммивояжер (фр. commis voyageur) — бродячий торговец. Задача коммивояжёра — одна из самых важных задач транспортной логистики, отрасли, занимающейся планированием транспортных перевозок [3]. В описываемой задаче рассматривается несколько городов и матрица попарных расстояний между ними (матрица смежности, если рассматривать граф).

Требуется найти такой порядок посещения городов, чтобы суммарное пройденное расстояние было минимальным, каждый город посещался ровно один раз и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном графе требуется найти гамильтонов цикл минимального веса [4].

1.2 Алгоритм полного перебора

Алгоритм полного перебора для решения задачи коммивояжера предполагает рассмотрение всех возможных путей в графе и выбор наименьшего из них. Смысл перебора состоит в том, что мы перебираем все варианты объезда городов и выбираем оптимальный. Однако, при таком подходе количество возможных маршрутов очень быстро возрастает с ростом n (сложность алгоритма равна $n!$).

Такой подход гарантирует точное решение задачи, однако, уже при небольшом числе городов решение за приемлемое количество времени невозможно.

1.3 Муравьиный алгоритм

Муравьиные алгоритмы представляют собой перспективный метод решения задач коммивояжера, в основе которого лежит моделирование по-

ведения колонии муравьев [?].

Каждый муравей определяет для себя маршрут, который необходимо пройти на основе феромона, который он ощущает, во время прохождения, каждый муравей оставляет феромон на своем пути, чтобы остальные муравьи могли по нему ориентироваться. В результате при прохождении каждым муравьем различного маршрута наибольшее число феромона остается на оптимальном пути.

Самоорганизация колонии является результатом взаимодействия следующих компонентов:

— случайность — муравьи имеют случайную природу движения (как будто бросили монетку);

— многократность — колония допускает число муравьев, достигающее от нескольких десятков до миллионов особей;

— положительная обратная связь — во время движения муравей откладывает феромон, позволяющий другим особям определить для себя оптимальный маршрут;

— отрицательная обратная связь — по истечении определенного времени феромон испаряется.

Описание поведения муравьев при выборе пути.

1) Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов — список запретов. Обозначим через J_{ik} список городов, которые необходимо посетить муравью k, находящемуся в городе i.

2) Муравьи обладают «зрением» — желание посетить город j, если муравей находится в городе i. Будем считать, что видимость обратно пропорциональна расстоянию между городами.

3) Муравьи обладают «обонянием» — они могут улавливать след феромона, подтверждающий желание посетить город j из города i на

основании опыта других муравьёв. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{i,j}(t)$.

4) Пройдя ребро (i, j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ — длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано формулой (1.5).

Введем некую целевую функцию (1.1)

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

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

Посчитаем вероятности перехода в заданную точку по формуле (1.2):

$$P_{kij} = \begin{cases} \frac{\tau_{ij}^a \eta_{ij}^b}{\sum_{q=1}^{m} \tau_{iq}^a \eta_{iq}^b}, \text{вершина не была посещена ранее муравьем k,} \\ 0, \text{иначе} \end{cases} \tag{1.2}$$

где

— a — параметр влияния длины пути;

— b — параметр влияния феромона;

— τ_{ij} — расстояния от города i до j;

— η_{ij} — количество феромонов на ребре ij.

Когда все муравьи завершили движение происходит обновление феромона по формуле (1.3):

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

где

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \tau_{ij}^k. \tag{1.4}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} Q/L_k, \text{ребро посещено k-ым муравьем,} \\ 0, \text{иначе} \end{cases} \tag{1.5}$$

При этом

— L_k — длина пути k-ого муравья;

— Q — настраивает количество откладываемого феромона.

2 Конструкторская часть

2.1 Алгоритм полного перебора

На рисунке 2.1 приведена схема полного перебора.

Рисунок 2.1 – Схема алгоритма полного перебора

2.2 Муравьиный алгоритм

На рисунке 2.2 приведена схема муравьиного алгоритма.

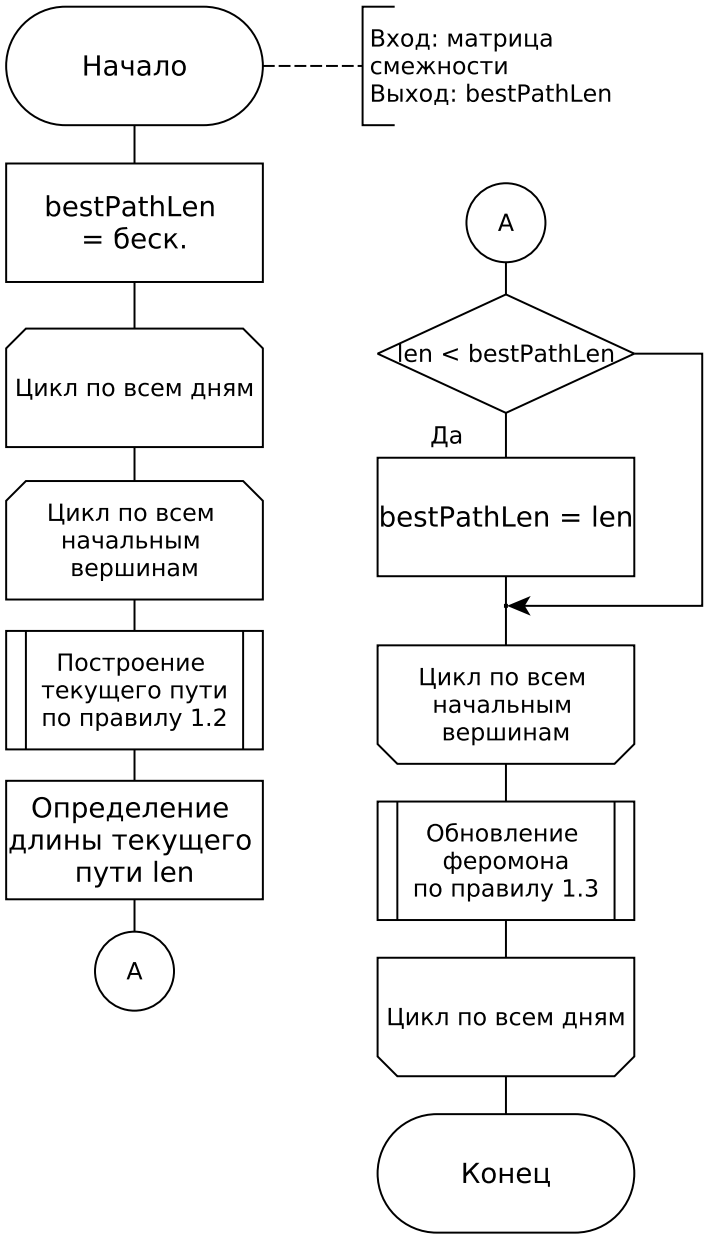

Рисунок 2.2 – Схема муравьиного алгоритма

2.3 Требования к программе

Программа должна выполнять следующие действия:

— генерировать граф из 8–10 вершин, вычислять наименьшую длину гамильтонова цикла методом полного перебора;

— определять наилучшие параметры муравьиного алгоритма для данного класа графов;

— замерять время реализации алгоритма полного перебора и муравьиного алгоритма.

3 Технологическая часть

3.1 Средства реализации

Для реализации данной лабораторной работы был выбран язык программирования C++ [5] и среда разработки CLion, которая позволяет замерять процессорное время с помощью пакета `<ctime>` [6].

3.2 Реализация алгоритмов

В листингах 3.1–3.3 приведены реализации класса Colony, алгоритма полного перебора и муравьиного алгоритма. В листингах 3.4–3.7 приведены реализации вспомогательных функций.

Листинг 3.1 – Алгоритм полного перебора

```
1  std::pair<std::vector<int>, double> exhaustiveSearch(const Graph
       &graph) {
2      std::vector<int> bestPath(graph.size());
3      std::vector<int> curPath(graph.size());
4      for (int i = 0; i < graph.size(); i++) {
5          curPath[i] = i;
6      }
7      double bestPathLen = pathLen(graph, curPath);
8
9      while(std::next_permutation(curPath.begin() + 1,
           curPath.end())) {
10         auto len = pathLen(graph, curPath);
11         if (len < bestPathLen) {
12             bestPathLen = len;
13             bestPath = curPath;
14         }
15     }
16
17     return std::make_pair(bestPath, bestPathLen);
18 }
```

Листинг 3.2 – Класс Colony

```
1  struct SimulationResult {
2      std::vector<size_t> path;
3      double pathLen;
4  };
5
6  class Colony {
7      private:
8      static constexpr const double START_PHEROMONE = 0.3;
9      std::default_random_engine generator;
10     std::uniform_real_distribution<double> distribution =
           std::uniform_real_distribution<double>(0, 1);
11
12     Graph graph;
13     Graph pheromoneGraph;
14     Parameters params;
15
16     public:
17     explicit Colony(Graph &graph);
18     SimulationResult simulation(Parameters parameters);
19
20     private:
21     std::vector<size_t> makePath(size_t start_vert);
22     double getPathLen(std::vector<size_t> path);
23     void updatePheromone(const std::vector<std::vector<size_t>>
           &paths);
24     void vaporizePheromone();
25     void addPheromone(const std::vector<std::vector<size_t>>
           &paths);
26     void addPheromone(const std::vector<size_t> &path);
27     double randomProbability();
28     std::vector<double> vertexesProbabilities(size_t curVert,
           const std::vector<size_t> &availableVertexes);
29     size_t choseVert(size_t vertex, const std::vector<size_t>
           &visitedVertexes);
30 };
```

Листинг 3.3 – Муравьиный алгоритм

```
SimulationResult Colony::simulation(Parameters parameters) {
    params = parameters;
    pheromoneGraph = Graph(graph.size(), START_PHEROMONE);
    std::vector<size_t> bestPath;
    double bestPathLen = -1;

    for (size_t i = 0; i < params.simulationDays; ++i) {
        std::vector<std::vector<size_t>> dayPaths(graph.size());
        for (size_t v = 0; v < graph.size(); ++v) {
            auto path = makePath(v);
            auto pathLen = getPathLen(path);
            dayPaths.at(v) = path;

            if (pathLen < bestPathLen || std::abs(bestPathLen +
               1) < std::numeric_limits<double>::epsilon()) {
                bestPath = path;
                bestPathLen = pathLen;
            }
        }
        updatePheromone(dayPaths);
        addPheromone(bestPath); // elite ants
    }

    return { bestPath, bestPathLen };
}
```

Листинг 3.4 – Построение маршрута

```
std::vector<size_t> Colony::makePath(size_t startVert) {
    std::vector<size_t> visitedVertexes = {startVert};

    auto curVert = startVert;
    while (visitedVertexes.size() != graph.size()) {
        curVert = choseVert(curVert, visitedVertexes);
        visitedVertexes.push_back(curVert);
    }

    return visitedVertexes;
}
```

Листинг 3.5 – выбор следующей вершины

```
size_t Colony::choseVert(size_t vertex, const std::vector<size_t>
    &visitedVertexes) {
    auto probability = randomProbability();
    auto availableVertexes = graph.getAvailableVertices(vertex,
        visitedVertexes);

    auto probabilities = vertexesProbabilities(vertex,
        availableVertexes);
    auto denominator = std::accumulate(probabilities.begin(),
        probabilities.end(), 0.0);

    double curProb = 0.0;
    for (auto &v: availableVertexes) {
        curProb += probabilities.at(v) / denominator;
        if (curProb >= probability) {
            return v;
        }
    }

    return availableVertexes[availableVertexes.size() - 1];
}
```

Листинг 3.6 – получение вероятностей перехода в доступные вершины

```
std::vector<double> Colony::vertexesProbabilities(size_t curVert,
    const std::vector<size_t> &availableVertexes) {
    std::vector<double> probabilities(graph.size(), 0);

    for (auto &v: availableVertexes) {
        probabilities.at(v) = static_cast<double &&>(
        std::pow(pheromoneGraph.get(curVert, v), params.a)
        * std::pow(1. / graph.get(curVert, v), params.b)
        );
    }

    return probabilities;
}
```

Листинг 3.7 – обновление феромона

```
1  void Colony::updatePheromone(const
       std::vector<std::vector<size_t>> &paths) {
2      for (size_t i = 0; i < graph.size(); ++i) {
3          for (size_t j = 0; j < graph.size(); ++j) {
4              if (i == j) {
5                  continue;
6              }
7
8              pheromoneGraph.set(i, j, pheromoneGraph.get(i, j) *
                   (1 - params.p));
9              if (pheromoneGraph.get(i, j) <= 1e-5) {
10                 pheromoneGraph.set(i, j, 0.3);
11             }
12         }
13     }
14
15     for (auto &path: paths) {
16         for (size_t i = 0; i < path.size() - 1; ++i) {
17             size_t from = path.at(i);
18             size_t to = path.at(i + 1);
19             auto delta = params.q / graph.get(from, to);
20             auto new_val = pheromoneGraph.get(from, to) + delta;
21             pheromoneGraph.set(from, to, new_val);
22             pheromoneGraph.set(to, from, new_val);
23         }
24     }
25 }
```

3.3 Функциональное тестирование

В таблице 3.1 приведены тесты для функций, реализующих алгоритм полного перебора и муравьиный алгоритм. Все тесты пройдены успешно.

Таблица 3.1 – Тестирование функций

Матрица смежности	Ожидаемый рез.	Действительный рез.
$\begin{pmatrix} 0 & 1 & 10 & 7 \\ 1 & 0 & 1 & 2 \\ 10 & 1 & 0 & 1 \\ 7 & 2 & 1 & 0 \end{pmatrix}$	10, [0, 1, 2, 3, 0]	10, [0, 1, 2, 3, 0]
$\begin{pmatrix} 0 & 3 & 5 & 7 \\ 7 & 0 & 1 & 2 \\ 5 & 1 & 0 & 1 \\ 7 & 2 & 1 & 0 \end{pmatrix}$	11, [2, 3, 1, 0, 2]	11, [2, 3, 1, 0, 2]
$\begin{pmatrix} 0 & 3 & 5 \\ 3 & 0 & 1 \\ 5 & 1 & 0 \end{pmatrix}$	9, [0, 1, 2, 0]	9, [0, 1, 2, 0]

4 Исследовательская часть

4.1 Параметризация метода

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров. Рассмотрим два класса данных и подберем к ним параметры, при которых метод даст точный результат при минимальном количестве итераций (итерациями считаются кол-во дней). Будем рассматривать матрицы размерности 10×10 и проводить 10 запусков для каждого набора параметров.

В качестве первого класса данных рассмотрим графы, матрицы смежности которых содержат значения в диапазоне $[1, 5]$.

В качестве второго класса данных рассмотрим графы, матрицы смежности которых содержат значения в диапазоне $[1, 1000]$.

Результат параметризации представлен в таблицах 4.1–4.6 со значениями a, p, $tmax$, $delta$, где

— a, p — настраиваемые параметры;

— $tmax$ — число дней симуляции;

— $delta$ — разность эталонного решения задачи и решения муравьиного алгоритма с данными параметрами на первом классе данных.

Наилучшие параметры для класса данных 1: $a = 0.9, p = 0.8, tmax = 200$.

Наилучшие параметры для класса данных 2: $a = 0.1, p = 0.2, tmax = 50$.

Таблица 4.1 – Результат параметризации для класса данных 1 (ч. 1)

a	p	tmax	delta
0.1	0	50	1.6
0.1	0	100	0.4
0.1	0	200	0.4
0.1	0	400	0.4
0.1	0.2	50	1
0.1	0.2	100	0.4
0.1	0.2	200	0.6
0.1	0.2	400	0.6
0.1	0.4	50	1.6
0.1	0.4	100	0.6
0.1	0.4	200	0.8
0.1	0.4	400	0.2
0.1	0.6	50	1.4
0.1	0.6	100	1.2
0.1	0.6	200	0.8
0.1	0.6	400	0.4
0.1	0.8	50	1.2
0.1	0.8	100	1.6
0.1	0.8	200	0.6
0.1	0.8	400	0.4
0.1	1	50	1.6
0.1	1	100	0.8
0.1	1	200	0.6
0.1	1	400	0.4
0.25	0	50	1
0.25	0	100	0.8
0.25	0	200	0.4
0.25	0	400	0
0.25	0.2	50	1
0.25	0.2	100	0.8
0.25	0.2	200	0.8
0.25	0.2	400	0
0.25	0.4	50	1.8
0.25	0.4	100	0.8
0.25	0.4	200	0.8
0.25	0.4	400	0.2
0.25	0.6	50	1.6
0.25	0.6	100	1
0.25	0.6	200	0.2

Таблица 4.2 – Результат параметризации для класса данных 1 (ч. 2)

a	p	tmax	delta
0.25	0.8	50	1.2
0.25	0.8	100	1
0.25	0.8	200	0.6
0.25	0.8	400	0.2
0.25	1	50	1.2
0.25	1	100	0.4
0.25	1	200	0.4
0.25	1	400	0.2
0.5	0	50	1.8
0.5	0	100	0.6
0.5	0	200	0
0.5	0	400	0.2
0.5	0.2	50	1.4
0.5	0.2	100	0.6
0.5	0.2	200	0.6
0.5	0.2	400	0
0.5	0.4	50	1.4
0.5	0.4	100	1.2
0.5	0.4	200	0
0.5	0.4	400	0
0.5	0.6	50	1.4
0.5	0.6	100	0.8
0.5	0.6	200	0.2
0.5	0.6	400	0
0.5	0.8	50	1
0.5	0.8	100	0.6
0.5	0.8	200	0
0.5	0.8	400	0
0.5	1	50	1.2
0.5	1	100	0.8
0.5	1	200	0.4
0.5	1	400	0.2
0.75	0	50	1.2
0.75	0	100	0.4
0.75	0	200	0.4
0.75	0	400	0
0.75	0.2	50	0.8
0.75	0.2	100	0.4
0.75	0.2	200	0
0.75	0.2	400	0

Таблица 4.3 – Результат параметризации для класса данных 1 (ч. 3)

a	p	tmax	delta
0.75	0.4	50	0.8
0.75	0.4	100	0.2
0.75	0.4	200	0.4
0.75	0.4	400	0
0.75	0.6	50	1
0.75	0.6	100	0.6
0.75	0.6	200	0
0.75	0.6	400	0
0.75	0.8	50	0.4
0.75	0.8	100	0.8
0.75	0.8	200	0
0.75	0.8	400	0
0.75	1	50	1.2
0.75	1	100	0.4
0.75	1	200	0
0.75	1	400	0
0.9	0	50	1.2
0.9	0	100	0.4
0.9	0	200	0
0.9	0	400	0
0.9	0.2	50	1
0.9	0.2	100	0.4
0.9	0.2	200	0
0.9	0.2	400	0
0.9	0.4	50	0.8
0.9	0.4	100	0.2
0.9	0.4	200	0
0.9	0.4	400	0
0.9	0.6	50	0.8
0.9	0.6	100	0.4
0.9	0.6	200	0
0.9	0.6	400	0
0.9	0.8	50	0.6
0.9	0.8	100	0.2
0.9	0.8	200	0
0.9	0.8	400	0
0.9	1	50	0.4
0.9	1	100	0.6
0.9	1	200	0
0.9	1	400	0

Таблица 4.4 – Результат параметризации для класса данных 2 (ч. 1)

a	p	tmax	delta
0.1	0	50	0
0.1	0	100	0
0.1	0	200	0
0.1	0	400	0
0.1	0.2	50	0
0.1	0.2	100	0
0.1	0.2	200	0
0.1	0.2	400	0
0.1	0.4	50	0.2
0.1	0.4	100	0
0.1	0.4	200	0
0.1	0.4	400	0
0.1	0.6	50	0.2
0.1	0.6	100	0
0.1	0.6	200	0
0.1	0.6	400	0
0.1	0.8	50	0
0.1	0.8	100	0
0.1	0.8	200	0
0.1	0.8	400	0
0.1	1	50	0
0.1	1	100	0
0.1	1	200	0
0.1	1	400	0
0.25	0	50	0.2
0.25	0	100	0
0.25	0	200	0
0.25	0	400	0
0.25	0.2	50	0.2
0.25	0.2	100	0
0.25	0.2	200	0
0.25	0.2	400	0
0.25	0.4	50	0
0.25	0.4	100	0
0.25	0.4	200	0
0.25	0.4	400	0
0.25	0.6	50	0
0.25	0.6	100	0
0.25	0.6	200	0
0.25	0.6	400	0

Таблица 4.5 – Результат параметризации для класса данных 2 (ч. 2)

a	p	tmax	delta
0.25	0.8	50	0
0.25	0.8	100	0
0.25	0.8	200	0
0.25	0.8	400	0
0.25	1	50	0
0.25	1	100	0
0.25	1	200	0
0.25	1	400	0
0.5	0	50	0.2
0.5	0	100	0
0.5	0	200	0
0.5	0	400	0
0.5	0.2	50	0
0.5	0.2	100	0
0.5	0.2	200	0
0.5	0.2	400	0
0.5	0.4	50	9.2
0.5	0.4	100	0
0.5	0.4	200	0
0.5	0.4	400	0
0.5	0.6	50	0
0.5	0.6	100	0
0.5	0.6	200	0
0.5	0.6	400	0
0.5	0.8	50	0.2
0.5	0.8	100	0
0.5	0.8	200	0
0.5	0.8	400	0
0.5	1	50	0
0.5	1	100	0
0.5	1	200	0
0.5	1	400	0
0.75	0	50	0.4
0.75	0	100	9.4
0.75	0	200	0
0.75	0	400	0
0.75	0.2	50	9.4
0.75	0.2	100	0.2
0.75	0.2	200	0
0.75	0.2	400	0

Таблица 4.6 – Результат параметризации для класса данных 2 (ч. 3)

a	p	tmax	delta
0.75	0.4	50	27.6
0.75	0.4	100	9.6
0.75	0.4	200	0.2
0.75	0.4	400	0
0.75	0.6	50	0.4
0.75	0.6	100	9.4
0.75	0.6	200	0
0.75	0.6	400	0
0.75	0.8	50	36.8
0.75	0.8	100	9.4
0.75	0.8	200	0
0.75	0.8	400	0
0.75	1	50	27.6
0.75	1	100	9.2
0.75	1	200	0
0.75	1	400	0
0.9	0	50	63.8
0.9	0	100	27.8
0.9	0	200	18.6
0.9	0	400	0.2
0.9	0.2	50	86.4
0.9	0.2	100	18.6
0.9	0.2	200	9.2
0.9	0.2	400	9.2
0.9	0.4	50	86.6
0.9	0.4	100	64
0.9	0.4	200	0
0.9	0.4	400	9.2
0.9	0.6	50	18.6
0.9	0.6	100	27.6
0.9	0.6	200	18.6
0.9	0.6	400	0
0.9	0.8	50	88.8
0.9	0.8	100	45.4
0.9	0.8	200	9.4
0.9	0.8	400	0
0.9	1	50	86.4
0.9	1	100	27.6
0.9	1	200	18.6
0.9	1	400	0.4

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система — Ubuntu 22.04.1 Linux x86_64;

— оперативная память — 8 ГБ;

— процессор — AMD Ryzen 5 3550H [7].

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время замеров ноутбук не был нагружен сторонними приложениями.

4.3 Время выполнения алгоритмов

Для проведения замеров времени, программа запускалась на различных размерах графов. Реализация муравьиного алгоритма работала при лучших параметриах.

На рисунке 4.1 представлены результаты замеров времени выполнения реализаций алгоритмов для класса данных 1.

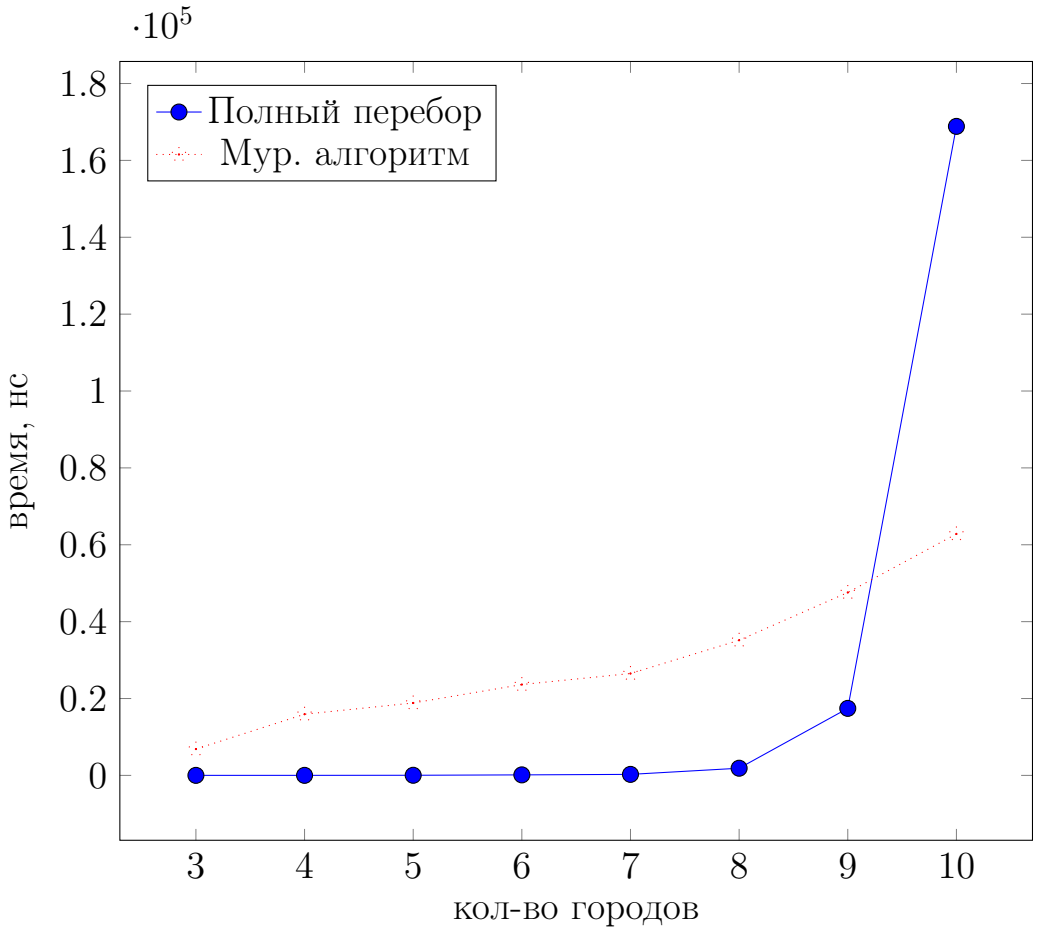

Рисунок 4.1 – Время выполнения реализаций алгоритмов для класса данных 1

На рисунке 4.2 представлены результаты замеров времени выполнения реализаций алгоритмов для класса данных 2.

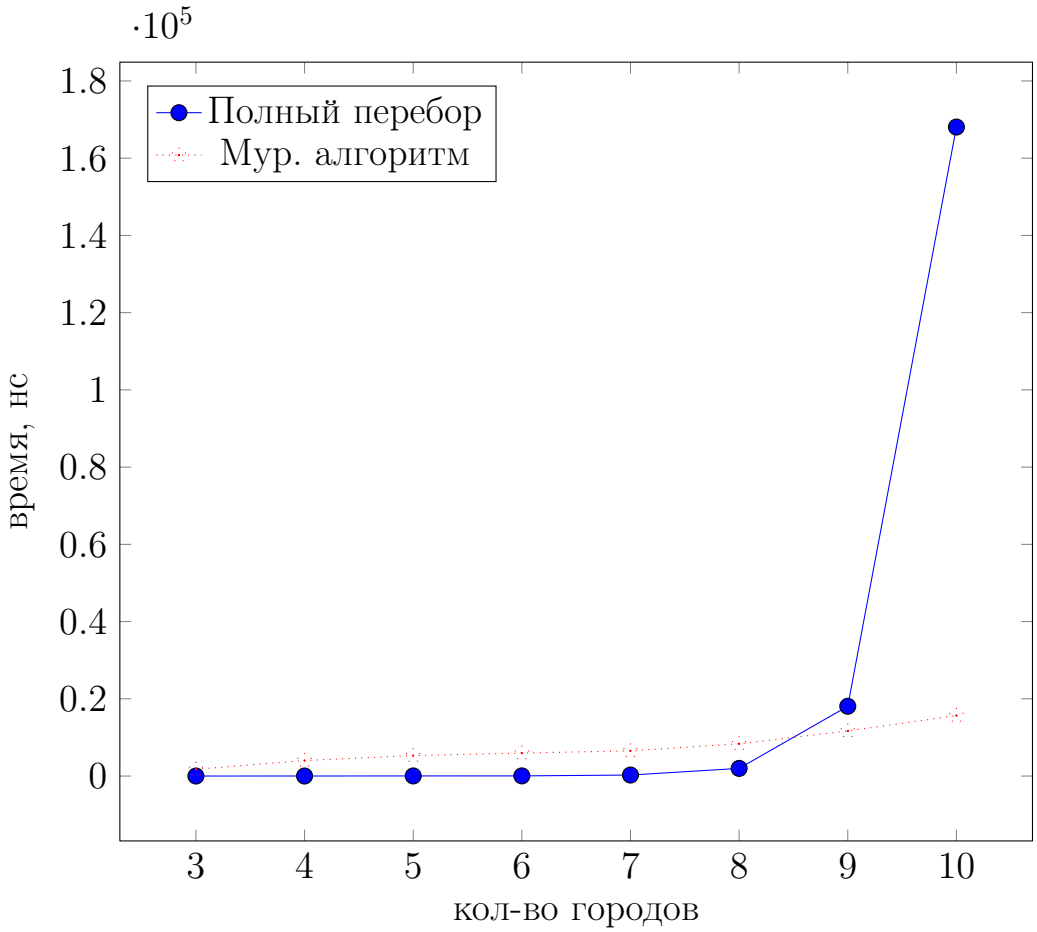

Рисунок 4.2 – Время выполнения реализаций алгоритмов для класса данных 2

Вывод

Для обоих классов данных при кол-ве городов до 9 время работы реализации муравьиного алгоритма больше, чем при полном переборе: для 8 городов в 3 раза больше на графе первого класса данных и в 1.5 раз больше на графе второго класса. При большем кол-ве городов реализация муравьиного алгоритма выигрывает по временной эффективности: в 8 раз для графа с 10 вершинами класса данных 1 и в 30 раз для графа той же размерности класса данных 2.

Заключение

Цель достигнута: были описаны и реализованы решения задачи коммивояжера с помощью полного перебора и с помощью муравьиного алгоритма. В ходе выполнения лабораторной работы были решены все задачи:

1) описаны и реализованы алгоритм полного перебора и муравьиный алгоритм;

2) проведена параметризация муравьиного алгоритма на двух классах данных;

3) проведено сравнение затрат реализаций по времени выполнения.

Были также сделаны выводы на основе полученных данных. Эвристический метод, основанный на муравьином алгоритме имеет преимущество перед методом полного перебора за счет того, что способен работать с данными достаточно большого объема, в то время как полный перебор сильно ограничен размером данных. Также были подобраны параметры для оптимальной работы метода на двух классах данных. Однако, в отличии от полного перебора, эвристический алгоритм не гарантирует точность найденного им пути, есть вероятность, что путь будет не оптимален.

Список использованных источников

1. Dorigo M. Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale (Optimization, Learning, and Natural Algorithms). Politecnico di Milano, 1992.

2. TSPr [Электронный ресурс]. URL: `http://www.math.nsc.ru/LBRT/k5/OR-MMF/TSPr.pdf` (дата обращения: 25.11.2022).

3. Задача коммирояжера [Электронный ресурс]. Режим доступа: `http://galyautdinov.ru/post/zadacha-kommivoyazhera` (дата обращения: 30.11.2022).

4. Гамильтонов цикл [Электронный ресурс]. Режим доступа: `https://bigenc.ru/mathematics/text/2343214` (дата обращения: 30.11.2022).

5. Документация по языку C++ | Microsoft Learn [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-150` (дата обращения: 30.11.2022).

6. <ctime> | Microsoft Learn [Электронный ресурс]. Режим доступа: `https://learn.microsoft.com/en-us/cpp/standard-library/ctime?redirectedfrom=MSDN&view=msvc-160` (дата обращения: 30.09.2022).

7. AMD Ryzen™ 5 3550H [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/apu/amd-ryzen-5-3550h` (дата обращения: 30.11.2022).