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»

Студент ИУ7-52Б _____ Новиков А. А.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Строганов Д. В.
(Подпись, дата) (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача коммивояжера — одна из самых известных и старейших задач комбинаторной оптимизации. Ее корни уходят в 1831 год, когда в Германии была опубликована книга под названием "Кто такой коммивояжер и что он должен делать для процветания своего предприятия". В книге содержалась рекомендация: "Следует стремиться посетить как можно больше торговых точек, избегая повторного посещения". Это можно считать первым формулированием задачи коммивояжера [1].

Цель лабораторной работы — рассмотрение алгоритмов решения задачи коммивояжера.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— сформулировать задачу коммивояжера;

— рассмотреть алгоритмы решения: полным перебором, с использованием муравьиного алгоритма;

— реализовать данные алгоритмы;

— провести сравнительный анализ времени работы алгоритмов;

— выполнить параметризацию для муравьиного алгоритма.

1 Аналитический раздел

В данном разделе будет сформулирована задача коммивояжера, а также будут рассмотрены 2 метода решения этой задачи: муравьиный алгоритм и алгоритм полного перебора.

1.1 Формулировка задачи коммивояжера

Пусть задан граф $G = (V, E)$, где V — множество вершин ($|V| = n$), а E — множество ребер ($|E| = m$). Каждое ребро ($(i, j) \in E$ имеет длину c_{ij}, определяемую матрицей расстояний $C = ||c_{ij}||$. Если между вершинами i и j отсутствует ребро, соответствующий элемент матрицы принимается равным бесконечности ($c_{ij} = \infty$) [1].

Подмножество попарно несмежных ребер графа G называется паросочетанием. Паросочетание считается совершенным, если каждая вершина графа инцидентна ровно одному ребру из этого множества. Совокупность простых попарно непересекающихся циклов, покрывающая все вершины графа G, называется 2-фактором. Если 2-фактор состоит из одного цикла, то он называется гамильтоновым циклом [2].

Задача заключается в поиске гамильтонова цикла минимальной длины, то есть цикла, который проходит через каждую вершину графа ровно один раз и возвращается в начальную точку.

1.1.1 Алгоритм полного перебора

Алгоритм полного перебора для решения задачи коммивояжера основывается на проверке всех возможных маршрутов в графе для определения минимального. Этот метод заключается в полном переборе всех вариантов обхода городов и выборе маршрута с наименьшей длиной. Однако количество возможных маршрутов быстро растет с увеличением числа городов n, поскольку сложность алгоритма составляет $n!$. Несмотря на то, что данный подход гарантирует получение точного решения, его применение становится крайне неэффективным даже при относительно небольшом количестве городов из-за значительных вычислительных затрат.

1.1.2 Муравьиный алгоритм

В основе муравьиного алгоритма лежит идея моделирования поведения колонии муравьев. Каждый муравей определяет свой маршрут на основе оставленных другими муравьями феромонов, а также сам оставляет оставляет феромоны, чтобы последующие муравьи ориентировались по ним. В результате при прохождении каждым муравьем своего маршрута наибольшее число феромонов остается на самом оптимальном пути. Временная сложность алгоритма была оценена как $683-(42,467N)+(1,0696N^2)$ [3]. Однако главный недостаток алгоритма заключается в том, что, по сравнению с алгоритмом полного перебора, он даёт приближенное решение задачи, а не точное.

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

$$P_{kij} = \begin{cases} \frac{\tau_{ij}^a \eta_{ij}^b}{\sum_{q \in J_{ik}} \tau_{iq}^a \eta_{iq}^b}, & \text{если вершина } j \text{ еще не посещена муравьем } k, \\ 0, & \text{иначе,} \end{cases} \tag{1.1}$$

где:

— a — параметр влияния феромона;

— b — параметр влияния длины пути;

— τ_{ij} — количество феромонов на ребре (i, j);

— η_{ij} — видимость (величина обратная расстоянию до вершины).

По окончанию движения всех муравьев уровень феромонов на ребрах обновляется по формуле:

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}, \tag{1.2}$$

где p — коэффициент испарения феромона, а $\Delta\tau_{ij}$ определяется как:

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij}^k, \tag{1.3}$$

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{если ребро } (i, j) \text{ посещено муравьем } k, \\ 0, & \text{иначе,} \end{cases} \tag{1.4}$$

где Q — параметр, связанный с длиной оптимального пути, а L_k — длина маршрута муравья k.

1.1.3 Описание алгоритма

1. Муравей исключает из дальнейшего выбора вершины, которые уже были посещены, ссылаясь на список посещенных вершин, хранящийся в его памяти (список запретов J_{ik}).

2. Муравей оценивает привлекательность вершин, основываясь на их видимости, которая обратно пропорциональна расстоянию между ними.

3. Муравей ощущает уровень феромонов на ребрах графа, что помогает ему определять предпочтительность маршрута.

4. После прохождения ребра (i, j) муравей оставляет на нем феромоны, количество которых зависит от длины маршрута L_k, пройденного муравьем, и параметра Q.

ВЫВОД

В данном разделе была представлена формулировка задача коммивояжера, а также рассмотрены 2 метода ее решения: муравьиный алгоритм и алгоритм полного перебора

2 Конструкторский раздел

В данном разделе будут определены требования к программному обеспечению и приведены схемы алгоритма полного перебора и муравьиного алгоритма для решения задачи коммивояжера.

2.1 Требования к программному обеспечению

Входные данные: матрица стоимостей взвешенного неориентированного графа.

Выходные данные: кратчайший гамильтонов цикл.

2.2 Представления алгоритмов

На рисунках 2.1 — 2.2 представлены схема алгоритма полного перебора и схема муравьиного алгоритма.

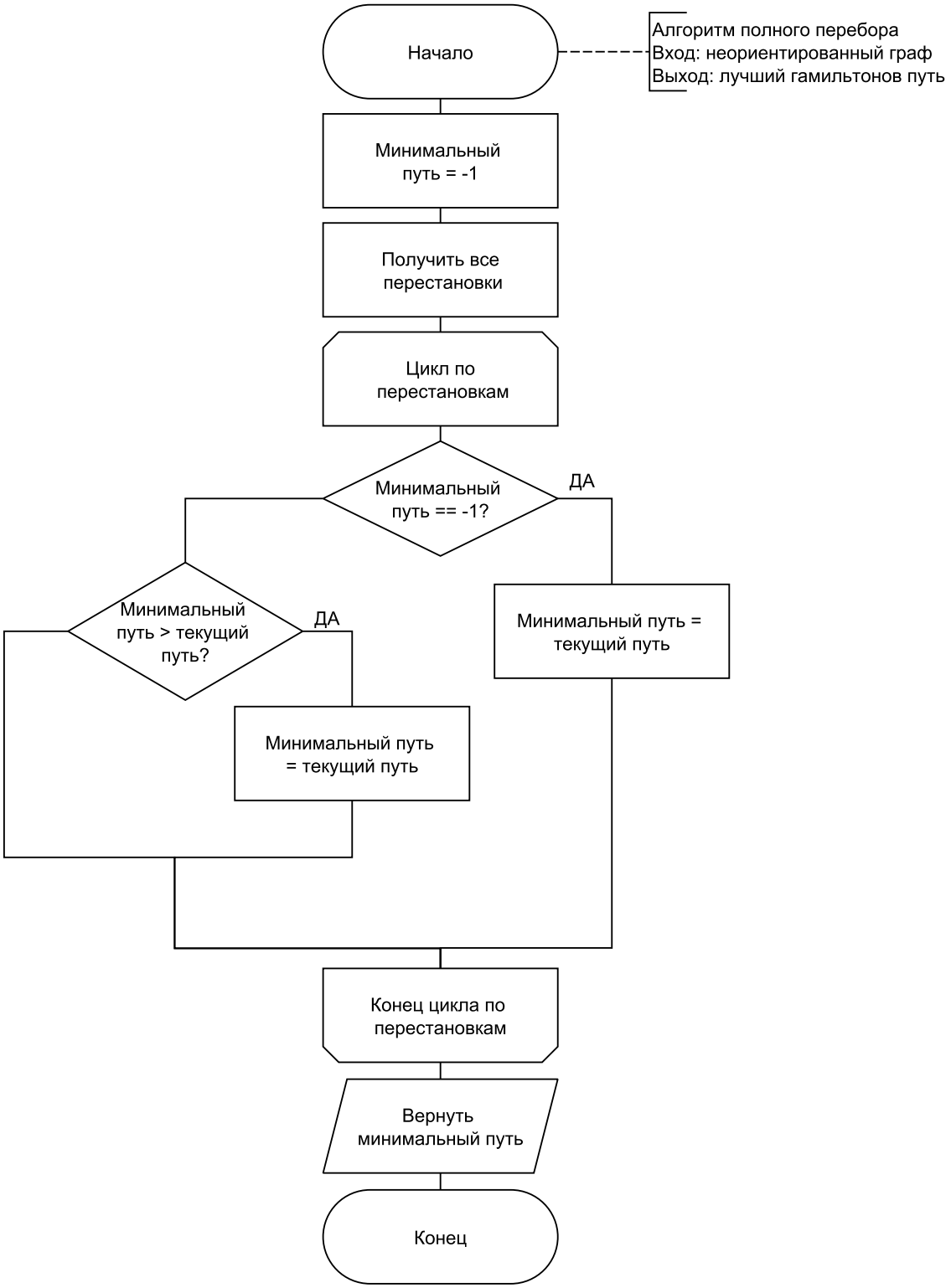

Рисунок 2.1 – Схема алгоритма полного перебора

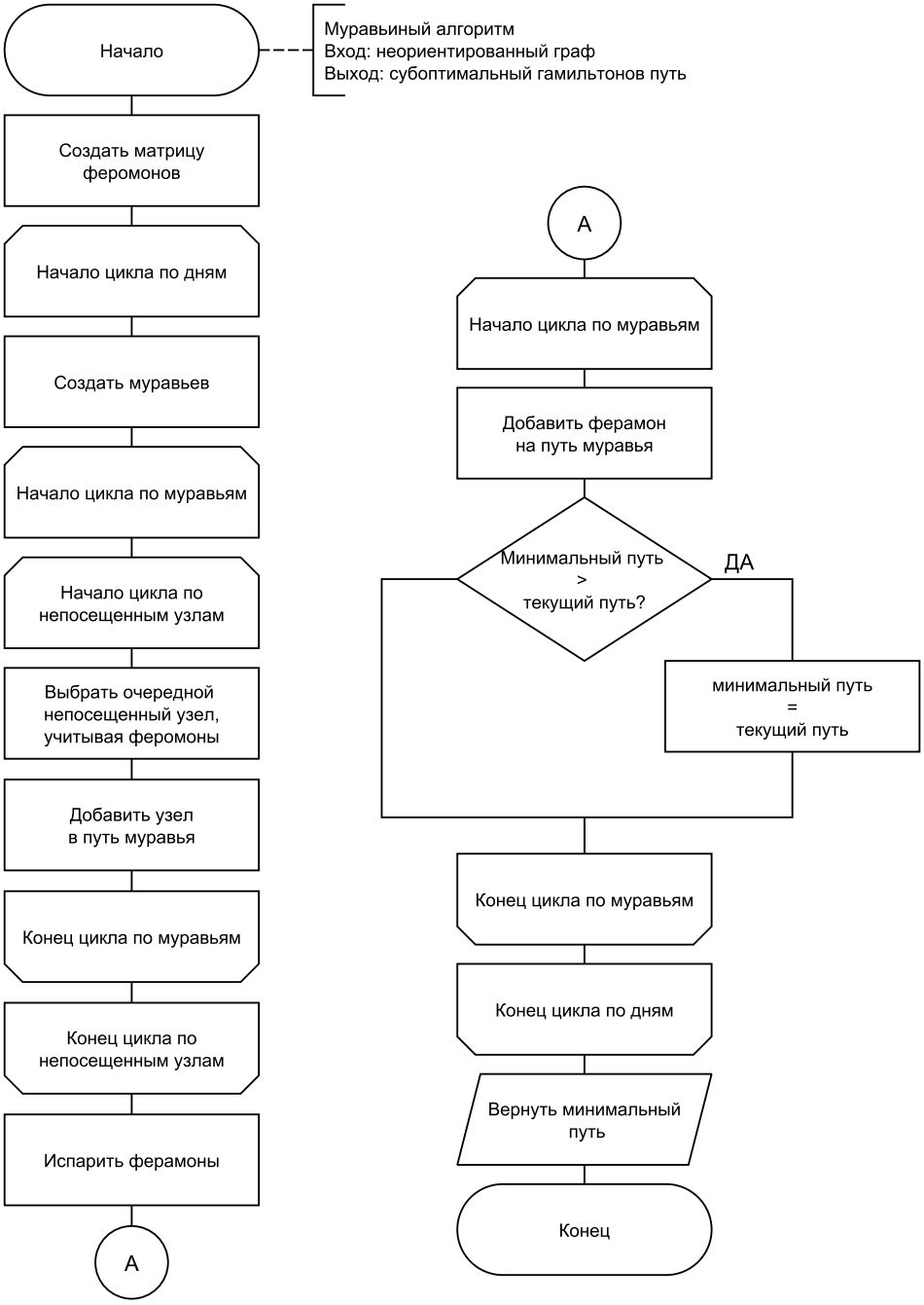

Рисунок 2.2 – Схема муравьиного алгоритма

ВЫВОД

В данном разделе были определены требования к программному обеспечению и приведены схемы алгоритмов полного перебора и муравьиного алгоритма для решения задачи коммивояжера.

3 Технологический раздел

В данном разделе будут приведены средства реализации, листинги кода.

3.1 Средства реализации

Для реализации данной лабораторной работы выбран язык программирования *C++* [4]. Выбор обусловлен скоростью выполнения и наличием множества библиотек, упрощающих разработку.

3.2 Реализация алгоритмов

В листингах 3.1 — 3.4 представлены реализации алгоритмов.

Листинг 3.1 – Реализация алгоритма полного перебора

```
1  std::vector<int> bruteForceTSP(const
       std::vector<std::vector<int>>& graph) {
2      int n = graph.size();
3      std::vector<int> vertices(n);
4      for (int i = 0; i < n; ++i) vertices[i] = i;
5
6      std::vector<int> bestCycle;
7      double minDistance = numeric_limits<double>::max();
8
9      do {
10         double currentDistance = calculateDistance(vertices,
               graph);
11         if (currentDistance < minDistance) {
12             minDistance = currentDistance;
13             bestCycle = vertices;
14         }
15     } while (next_permutation(vertices.begin() + 1,
           vertices.end()));
16
17     return bestCycle;
18 }
```

Листинг 3.2 – Реализация функции обновления ферамонов

```
1  void updatePheromones(std::vector<std::vector<double>>&
       pheromones, const std::vector<int>& bestCycle, double
       bestDistance) {
```

```
    int n = pheromones.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            pheromones[i][j] *= (1.0 - EVAPORATION_RATE);
        }
    }

    for (size_t i = 0; i < bestCycle.size() - 1; ++i) {
        int u = bestCycle[i];
        int v = bestCycle[i + 1];
        pheromones[u][v] += 1.0 / bestDistance;
        pheromones[v][u] += 1.0 / bestDistance;
    }
}
```

Листинг 3.3 – Реализация функции выбора следующей вершины

```
int chooseNextCity(int current, const std::vector<bool>&
   visited, const std::vector<std::vector<double>>& pheromones,
   const std::vector<std::vector<int>>& graph) {
    int n = graph.size();
    std::vector<double> probabilities(n, 0.0);
    double sum = 0.0;

    for (int next = 0; next < n; ++next) {
        if (!visited[next]) {
            probabilities[next] = pow(pheromones[current][next],
               ALPHA) * pow(1.0 / graph[current][next], BETA);
            sum += probabilities[next];
        }
    }

    double randomValue = ((double)rand() / RAND_MAX) * sum;
    double cumulative = 0.0;
    for (int next = 0; next < n; ++next) {
        if (!visited[next]) {
            cumulative += probabilities[next];
            if (cumulative >= randomValue) {
                return next;
            }
        }
    }
```

```
}
```

Листинг 3.4 – Реализация муравьиного алгоритма

```
vector<int> antColonyOptimization(const
   std::vector<std::vector<int>>& graph) {
    size_t n = graph.size();
    std::vector<std::vector<double>> pheromones(n,
       vector<double>(n, 1.0));
    std::vector<int> bestCycle;
    double bestDistance = numeric_limits<double>::max();

    srand(time(0));

    for (int iter = 0; iter < ITERATIONS; ++iter) {
        for (size_t start = 0; start < n; ++start) {
            std::vector<bool> visited(n, false);
            std::vector<int> cycle;
            int current = start;
            cycle.push_back(current);
            visited[current] = true;

            while (cycle.size() < n) {
                int next = chooseNextCity(current, visited,
                   pheromones, graph);
                if (next == -1) break;
                cycle.push_back(next);
                visited[next] = true;
                current = next;
            }

            if (cycle.size() == n &&
               graph[cycle.back()][cycle[0]] > 0) {
                cycle.push_back(cycle[0]);
                double distance = calculateDistance(cycle,
                   graph);
                if (distance < bestDistance) {
                    bestDistance = distance;
                    bestCycle = cycle;
                }
            }
        }
```

```
34
35          if (!bestCycle.empty()) {
36              updatePheromones(pheromones, bestCycle,
                    bestDistance);
37          }
38      }
39
40      return bestCycle;
41  }
```

ВЫВОД

В данном разделе были представлены реализации алгоритмов решения задачи коммивояжера: алгоритм полного перебора, муравьиный алгоритм.

4 Исследовательский раздел

В данном разделе проведен сравнительный анализ алгоритмов по используемому процессорному времени.

4.1 Технические характеристики

Технические характеристики используемого устройства:

— операционная система — Ubuntu Linux x86_64 [5];

— память — 16 Гб;

— процессор — AMD Ryzen 5 5500U (6x2.10 ГГц) [6].

4.2 Время выполнения алгоритмов

Замеры времени проводились на графах с одинаковым количеством вершин. Каждое значение получено путем взятия среднего из 10 измерений. Результаты замеров приведены в таблице 4.1.

Таблица 4.1 – Время работы алгоритмов (в мс)

Размер матрицы	Полный перебор	Муравьиный алгоритм
1	0.115	0.863
2	0.159	0.943
3	0.081	0.948
4	0.071	1.472
5	0.952	2.460
6	8.413	3.565
7	85.019	4.527
8	862.511	5.422
9	11282.250	6.619

Зависимости времени решения задачи коммивояжера от количества вершин графа для двух алгоритмов представлены на рисунке 4.1.

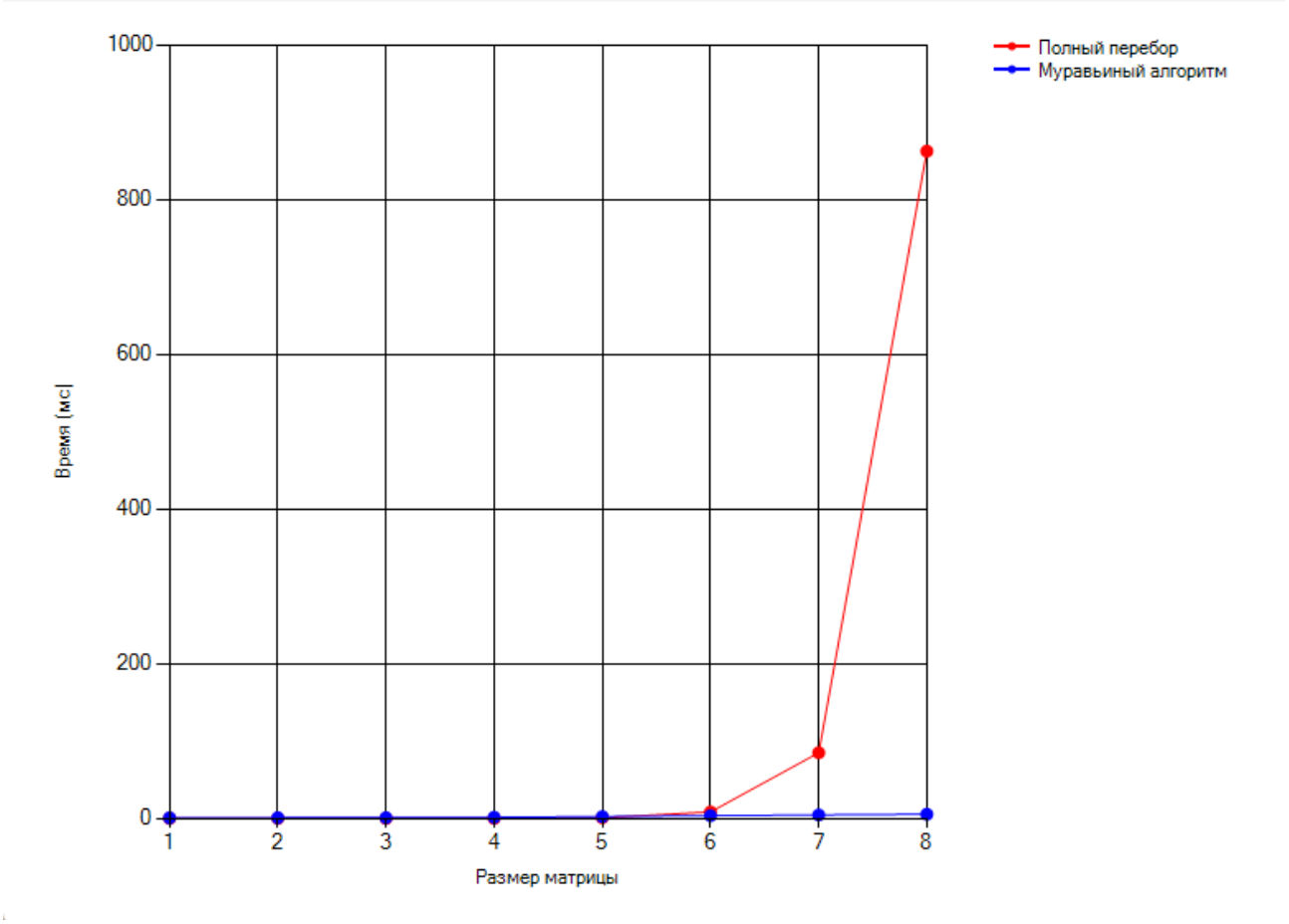

Рисунок 4.1 – Зависимость времени выполнения программы от количества вершин

4.3 Классы данных

В качестве класс данных для параметризации используются графы, построенные на городах Африке, при этом используется всего 3 графа. В качестве весов использовались расстояния между этими городами по прямой в км [7].

4.4 Результаты параметризации

В результате параметризации оказалось, что самыми лучшими параметрами по заданному классу данных оказались при $\alpha = 0.1$, $\rho = 0.5$ и количеством дней равным 50. При этом данные параметры дают лучший результат на всех классах данных и по всем параметрам сравнения. Результаты параметризации для лучших параметров представлены в таблице 4.2, а вся

таблица параметризации и класс данных представлены в приложении А.

Таблица 4.2 – Результаты параметризации муравьиного алгоритма

Параметры			Граф 1			Граф 2			Граф 3		
α	ρ	Дни	min	max	avg	min	max	avg	min	max	avg
0.1	0.25	100	0	0	0	0	0	0	0	0	0
0.1	0.25	200	0	0	0	0	0	0	0	0	0
0.1	0.5	50	0	0	0	0	0	0	0	0	0
0.1	0.5	100	0	0	0	0	0	0	0	0	0
0.1	0.75	200	0	0	0	0	0	0	0	0	0
0.25	0.1	100	0	0	0	0	0	0	0	0	0
0.25	0.1	100	0	0	0	0	0	0	0	0	0
0.5	0.1	200	0	0	0	0	0	0	0	0	0

ВЫВОД

В результате исследования было получено, что на графе с вершинами меньше 7 муравьиный алгоритм и алгоритм полного перебора выполняют задачу за примерно одинаковое время, а при количестве вершин большем или равном 7 муравьиный алгоритм работает около в раз быстрее, чем алгоритм полного перебора.

Также было выявлено, что при $\alpha = 0.1$, $\rho = 0.5$ и количеством дней равным 50 муравьиный алгоритм дает наилучшие результаты.

ЗАКЛЮЧЕНИЕ

В процессе работы были исследованы временные и алгоритмические сложности муравьиного алгоритма и метода полного перебора. Также были проведены замеры времени выполнения и параметризация муравьиного алгоритма, что дало возможность определить оптимальные параметры для набора данных, представленного в приложении А. Наилучшие параметры были установлены на уровне: $\alpha = 0.1$, $\rho = 0.5$ и количество дней равным 50.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

— сформулирована задача коммивояжера;

— рассмотреть алгоритмы решения: полным перебором, с использованием муравьиного алгоритма;

— реализовать данные алгоритмы;

— провести сравнительный анализ времени работы алгоритмов;

— выполнить параметризацию для муравьиного алгоритма.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Меламед И. И.*, *Сергеев С. И.*, *Сигал И. Х.* Задача коммивояжера. Вопросы теории // Автоматика и телемеханика. — 1989. — № 9. — С. 3—33.
2. *Пережогин А. Л.*, *Потапов В. Н.* О числе гамильтоновых циклов в булевом кубе // Дискретный анализ и исследование операций. — 2001. — Т. 8, № 2. — С. 52—62.
3. *Штовба С.* Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях. — 2003. — Т. 4, № 4. — С. 70—75.
4. C++ zprogramminglanguage [Электронный ресурс]. — URL: `https://isocpp.org/` (дата обращения 12.11.2024).
5. Ubuntu technical documentation for developers and IT pros [Электронный ресурс]. — URL: https://ubuntu.com/tutorials (дата обращения 01.10.2024).
6. AMD Ryzen 5 5500U Processor [Электронный ресурс]. — https://www.amd.com/en/products/apu/amd-ryzen-5-5500u (дата обращения 01.10.2024).
7. Google Maps [Электронный ресурс]. — URL: `https://www.google.ru/maps/` (дата обращения 20.12.2024).