Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 6

Муравьиный алгоритм, решение задачи коммивояжера

Студент:
Сиденко Анастасия Генадьевна
Группа: й7-53Б

Преподаватели:
Строганов Юрий Владимирович
Волкова Лилия Леонидовна

2019 г.

Содержание

Введение

Задача коммивояжера занимает особое место в комбинаторной оптимизации и исследовании операций. Исторически она была одной из тех задач, которые послужили толчком для развития этих направлений. Простота формулировки, конечность множества допустимых решений, но колоссальные затраты на полный перебор до сих пор подталкивают к разработке все новых и новых численных методов.

С точки зрения практического применения, она не представляет интерес.

Куда важнее дополнения задачи для транспорта и логистики, когда несколько транспортных средств ограниченной грузоподъемности должны обслуживать клиентов, посещая их в заданные временные окна. [1]

Целью данной работы является создание приложения для наглядного представления работы муравьиного алгоритма и для проведения вычислительных экспериментов.

В работе ставятся следующие задачи.

1. Изучение существующих методов решения задачи коммивояжера.
2. Программная реализация алгоритма оптимизации подражанием муравьиной колонии.
3. Проведение вычислительного эксперимента.

1 Аналитическая часть

История задачи коммивояжёра насчитывает почти 200 лет: в 1832 г. в Германии была опубликована книга «Коммивояжёр: как он должен вести себя и что должен делать для того, чтобы доставлять товар и иметь успех в своих делах – советы старого курьера» [2].

1.1 Описание задачи

Задача коммивояжёра – одна из самых известных задач дискретной оптимизации. Задача заключается в нахождении самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город [3].

Постановка задачи коммивояжёра: пусть дана сеть из N городов. Коммивояжёр, выходящий из какого-нибудь города, желает посетить $(N-1)$ других городов и вернуться в изначальный пункт. Расстояния между всеми этими городами известны. Требуется установить, в каком порядке коммивояжёру следует посетить города, чтобы суммарное пройденное расстояние было минимальным.

1.2 Пути решения

Для нахождения решения задачи коммивояжёра разработано множество различных алгоритмов.

Среди них:

1. Метод полного перебора;
2. Жадный алгоритм;
3. Метод ветвей и границ;
4. Метод имитации отжига;
5. Генетический алгоритм;
6. Муравьиный алгоритм.

В данной работе проводится исследование двух из них: метода полного перебора и муравьиного алгоритма.

Алгоритм полного перебора осуществляет поиск всех решений путём перебора всех вариантов в количестве $N!$ штук, позволяя получить точное решение задачи.

Это является главным преимуществом алгоритма, ещё одно достоинство – возможность распараллеливания. Главный недостаток метода полного перебора – временные затраты [3].

Муравьиный алгоритм (алгоритм оптимизации подражанием муравьиной колонии) представляет собой имитацию поведения колонии муравьёв в природе. В основе муравьиного алгоритма лежит вероятностный подход к поиску оптимального пути, однако имеют большое значение дополнительные критерии.

Преимуществами алгоритма являются невысокая погрешность найденного решения, низкие временные затраты при работе с графами большой размерности, модифицируемость алгоритма и возможность распараллеливания [3].

Алгоритм считается одним из самых эффективных наряду с генетическим и также широко применяется на практике.

Метод решения задачи коммивояжёра на основании муравьиного алгоритма. [4]

У муравья есть 3 чувства.

1. Обоняние – муравей может чуять феромон и его концентрацию на ребре.
2. Зрение – муравей может оценить длину ребра.
3. Память – муравей запоминает посещенные города.

При старте матрица феромонов τ инициализируется равномерно некоторой константой.

Если муравей k находится в городе i и выбирает куда пойти, то делает это по вероятностному правилу.

$$P_{x,y}(t) = \begin{cases} \frac{\tau_{ij}(t)^{\alpha}\eta_{ij}^{\beta}}{\sum_{\text{q в списке посещенных городов}} \tau_{iq}(t)^{\alpha}\eta_{iq}^{\beta}}, \text{если город j в списке целей} \\ 0, \text{если город j принадлежит списку посещенных городов} \end{cases} \quad (1)$$

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

Ночью учитываем изменение феромона по формуле 2.

$$\tau(t+1) = \tau_{ij}(t) \cdot (1-\rho) + \Delta\tau_{ij}(t) \tag{2}$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{n} \Delta\tau_{k,ij}(t) \tag{3}$$

$$\Delta\tau_{k,ij}(t) = \begin{cases} \frac{Q}{L_k}, \text{ребро i,j в маршруте k-ого муравья} \\ 0, \text{иначе} \end{cases} \tag{4}$$

L_k – длина маршрута k-ого муравья.
ρ – коэффициент испарения феромона.
Q – нормировочная константа порядка длины наилучшего маршрута.

1.3 Выводы

Рассмотрев теорию вопроса, необходимо реализовать данный алгоритм, подобрать необходимые коэффициенты и сравнить результаты с результатами алгоритма полного перебора.

2 Конструкторская часть

Целью данной работы является: создание приложения для наглядного представления работы муравьиного алгоритма и проведения вычислительных экспериментов.

2.1 Функциональная модель

На рисунке 1 представлена функциональная модель нашей задачи.

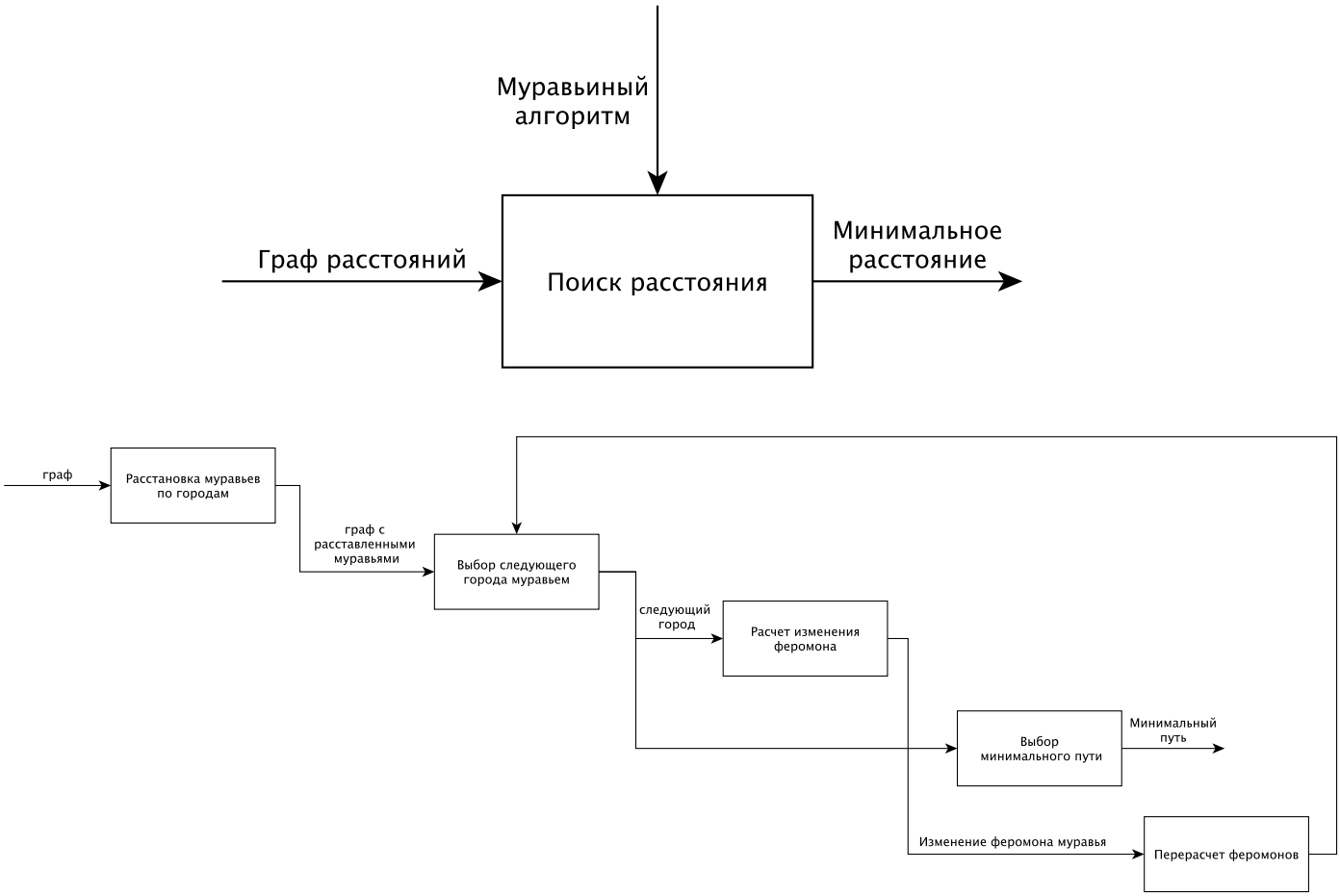

Рис. 1 - Функциональная модель муравьиного алгоритма.

2.2 Схемы алгоритмов

Приведем схему алгоритма (см. рисунок 2).

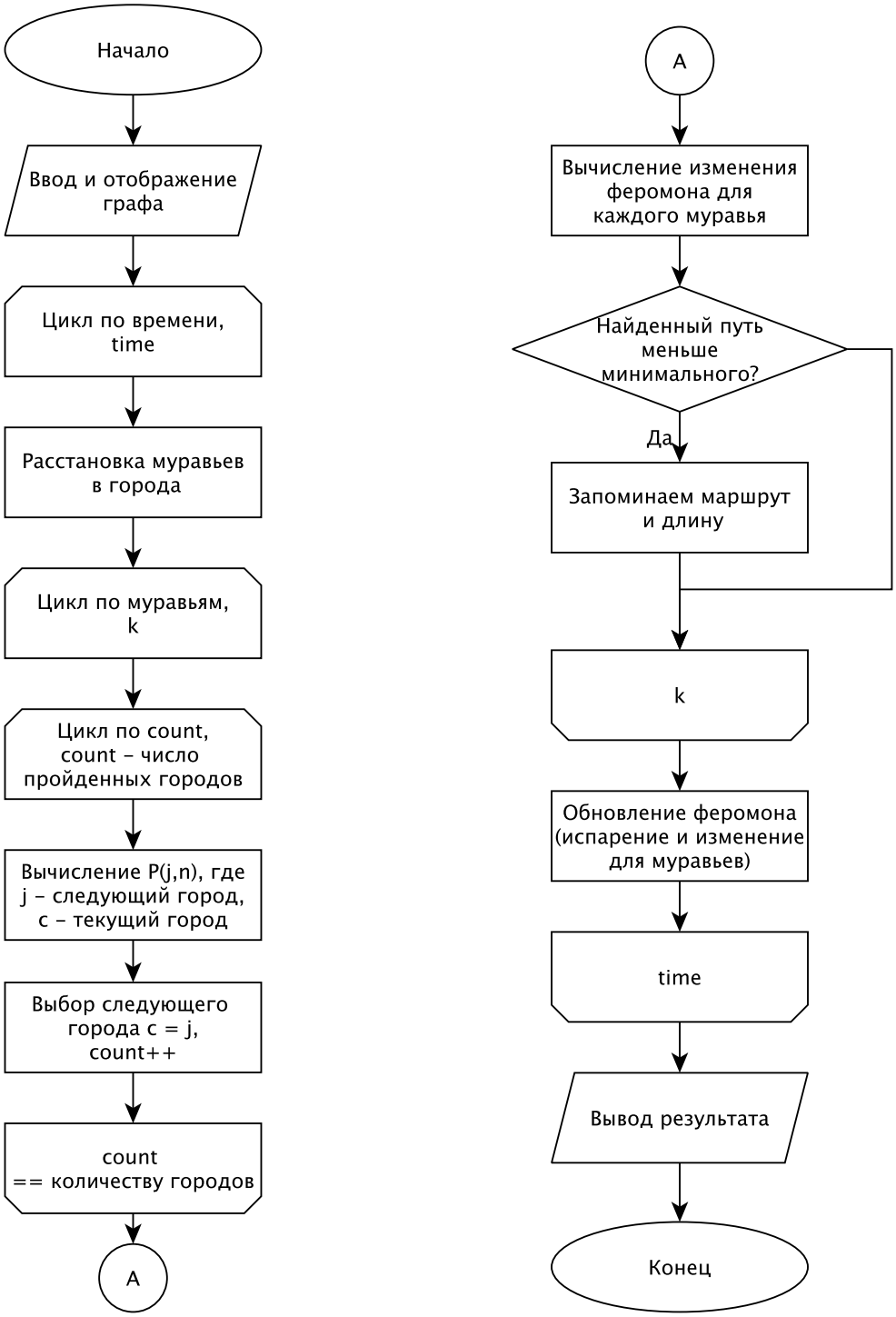

Рис. 2 - Алгоритм работы с программы

2.3 Выводы

Необходимо подобрать коэффициенты, при которых алгоритм будет вычислять реально минимальное расстояние.

3 Технологическая часть

В данном разделе происходит реализация необходимого алгоритма.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность замеров процессорного времени выполнения реализации каждого алгоритма. Требуется строить графы для различных данных.

3.2 Средства реализации

В качестве языка программирования был выбран C++, так как я знакома с этих языком программирования и он удовлетворяет требованиям об использовании dot. [5]

Для генерации случайных чисел использовался метод *rand*().

3.3 Листинг кода

В листинге 1 представлена реализация муравьиного алгоритма.

Листинг 1: Муравьиный алгоритм

```
1  int ant_algorithm(int **matrix, int n, double alpha, double beta, int timemax, double rho)
2    srand(time(NULL));
3    double **pheromone = new double * [n];
4    for (int i = 0; i < n; i++) {
5      pheromone[i] = new double [n];
6    }
7    for (int i = 0; i < n; i++) {
8      for (int j = 0; j < n; j++) {
9        pheromone[j][i] = 0.1;
10     }
11   }
12   int minlen = 1000;
13   int minway[n];
14
15   for (int time = 0; time < timemax; time++) {
16     int *ants = new int [n];
17     for (int i = 0; i < n; i++) {
18       ants[i] = i + 1;
19     }
20
21     double **delta = new double * [n];
22     for (int i = 0; i < n; i++) {
23       delta[i] = new double [n];
24     }
25
26     for (int k = 0; k < n; k++) {
27       int count = 0;
28       int cities[n];
29       for (int i = 0; i < n; i++) {
30         cities[i] = 0;
31       }
32       cities[k] = 1;
33       int way[n];
34       for (int i = 0; i < n; i++) {
35         way[i] = 0;
36       }
37       way[0] = k;
```

```
38        int len = 0;
39
40        while (count + 1 < n) {
41          double p[n];
42
43          for (int j = 0; j < n; j++) {
44            if (0 == cities[j]) {
45              p[j] = pow(pheromone[way[count]][j], alpha) * pow(matrix[way[count]][j],
46                                                                          beta);
47              double all = 0;
48              for (int q = 0; q < count; q++) {
49                all += pow(pheromone[way[q]][j], alpha) * pow(matrix[way[q]][j], beta);
50              }
51
52              p[j] /= all;
53            } else {
54              p[j] = 0;
55            }
56          }
57          int arr[n - count - 1];
58          int cyc = 0;
59          for (int i = 0; i < n; i++) {
60            if (0 == cities[i]) {
61              arr[cyc] = i;
62              cyc++;
63            }
64          }
65          int rdm = rand() % (n - count - 1);
66          len += matrix[way[count]][arr[rdm]];
67          count++;
68          way[count] = arr[rdm];
69          cities[arr[rdm]] = 1;
70        }
71        len += matrix[way[0]][way[n - 1]];
72
73        for (int i = 0; i < n - 1 ; i++) {
74          delta[way[i]][way[i + 1]] += minlen / len;
75          delta[way[i + 1]][way[i]] += minlen / len;
76        }
77        if (len < minlen) {
78          minlen = len;
79          for (int i = 0; i < n; i++)
80            minway[i] = way[i];
81        }
82      }
83
84      for (int i = 0; i < n - 1; i++) {
85        for (int j = i + 1; j < n; j++) {
86          pheromone[i][j] *= (1 - rho);
87          pheromone[i][j] += delta[i][j];
88        }
89      }
90    }
91
92    return minlen;
93  }
```

3.4 Выводы

Необходимо провести исследовани о выборе коэффициентов в муравьином алгоритме.

4 Экспериментальная часть

4.1 Примеры работы

На рисунке 4 представлены примеры работы программы на разных входных данных. В параметрах командной строки можно указать желаемое число количества чисел, по умолчанию 1.

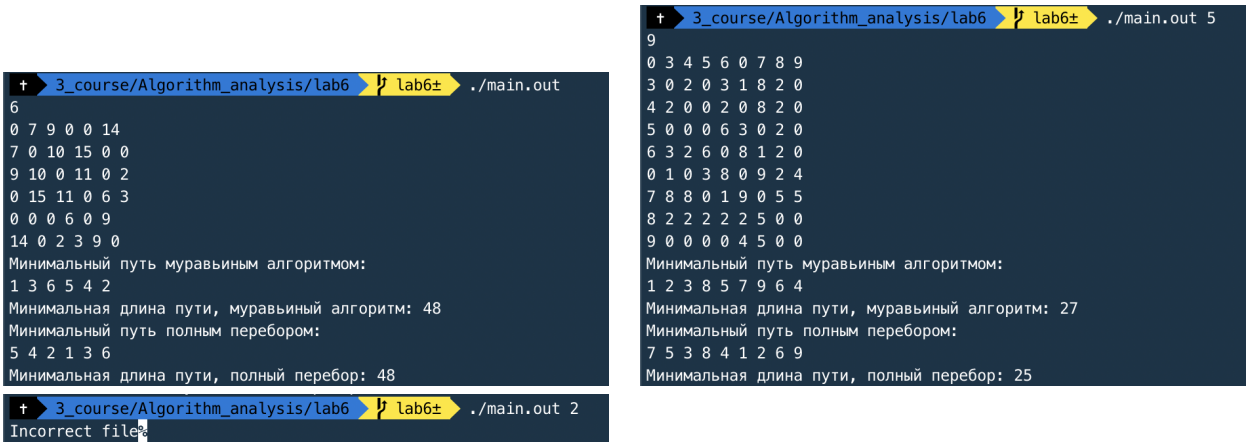

Рис. 4 - Примеры работы

4.2 Выбор коэффициентов

Было создано 5 файлов с различными графовыми моделями.

Проводилось исследование сравнения результата алгоритма полного перебора и муравьиного с различными коэффициентами ($0.2 \leq \rho \leq 1, 0.2 \leq \alpha \leq 1, 0.2 \leq \beta \leq 1, 10 \leq time \leq 700$).

α – коэффициент стадности.

β – коэффициент жадности.

time – количество итераций.

ρ – скорость испарения феромонов.

Таблица 1: Таблица, соответствующая графу на рисунке 4

α	β	time	ρ	Длина по алгоритму муравья	Длина по алгоритму полного перебора
0.2	0.2	10	0.2	48	48
0.2	0.2	10	0.4	48	48
0.2	0.2	10	0.6	48	48
...	...	...	...	...	...
0.8	0.8	640	0.4	48	48
0.8	0.8	640	0.6	48	48
0.8	0.8	640	0.8	48	48

Представленная таблица 1 соответствует графу на рисунке 4.

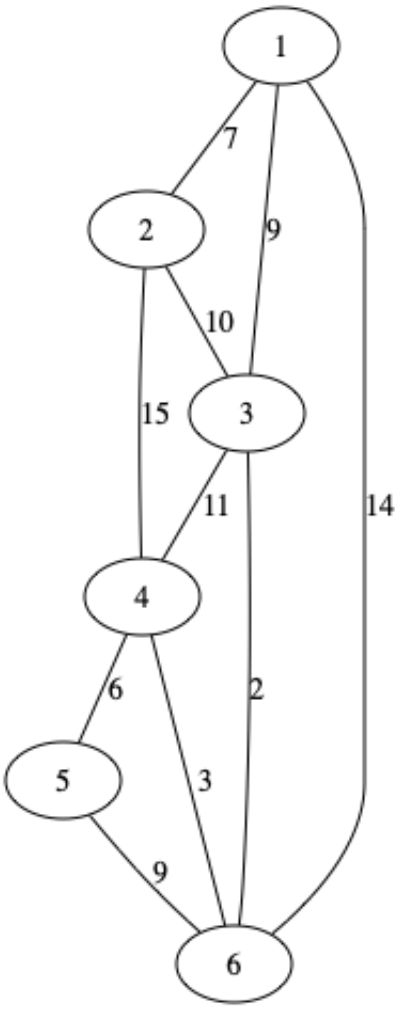

Рис. 4 - Граф 1, соответствующий тестовым данным 1.

Графы представленные на рисунка 5 и 6, вычисляются также при различных значениях.

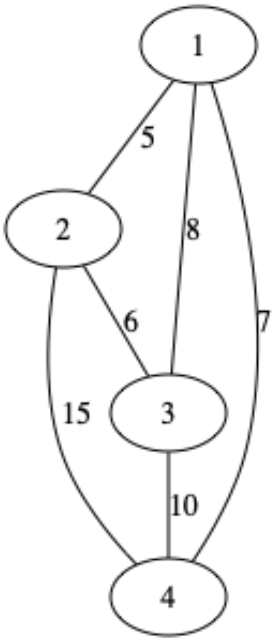

Рис. 5 - Граф 2, соответствующий тестовым данным 2.

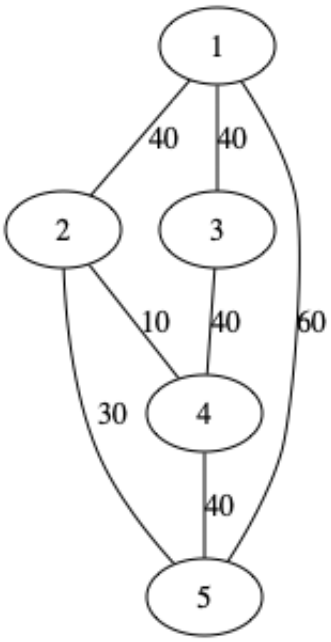

Рис. 6 - Граф 3, соответствующий тестовым данным 3.

Рассмотрим граф с рисунка 7.

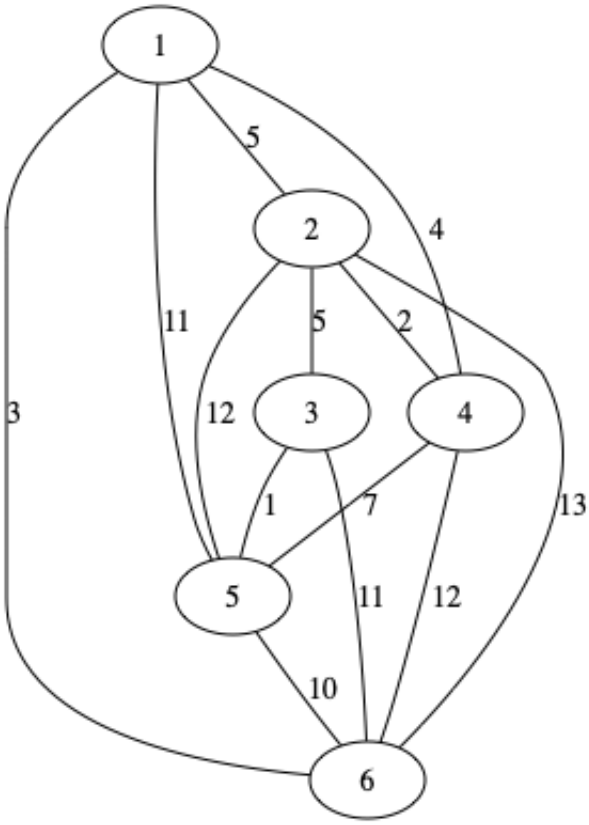

Рис. 7 - Граф 4, соответствующий тестовым данным 4.

Получим таблицу.

Таблица 2: Таблица, соответствующая графу на рисунке 7

α	β	time	ρ	Длина по алгоритму муравья	Длина по алгоритму полного перебора
0.2	0.2	10	0.2	29	25
0.2	0.2	10	0.4	29	25
0.2	0.2	10	0.6	29	25
0.2	0.2	10	0.8	29	25
0.2	0.2	20	0.2	29	25
0.2	0.2	20	0.4	29	25
0.2	0.2	20	0.6	29	25
0.2	0.2	20	0.8	29	25
0.2	0.2	40	0.2	25	25
0.2	0.2	40	0.4	25	25
0.2	0.2	40	0.6	25	25
0.2	0.2	40	0.8	25	25
0.2	0.2	80	0.2	25	25
0.2	0.2	80	0.4	25	25
0.2	0.2	80	0.6	25	25
0.2	0.2	80	0.8	25	25
0.2	0.2	160	0.2	25	25
0.2	0.2	160	0.4	25	25
0.2	0.2	160	0.6	25	25
0.2	0.2	160	0.8	25	25
0.2	0.2	320	0.2	25	25
0.2	0.2	320	0.4	25	25
0.2	0.2	320	0.6	25	25
0.2	0.2	320	0.8	25	25
0.2	0.2	640	0.2	25	25
0.2	0.2	640	0.4	25	25
0.2	0.2	640	0.6	25	25
0.2	0.2	640	0.8	25	25
...	...	...	...	...	...
0.8	0.8	10	0.2	29	25
0.8	0.8	10	0.4	29	25
0.8	0.8	10	0.6	29	25
0.8	0.8	10	0.8	29	25
0.8	0.8	20	0.2	29	25
0.8	0.8	20	0.4	29	25
0.8	0.8	20	0.6	29	25
0.8	0.8	20	0.8	29	25
0.8	0.8	40	0.2	25	25
0.8	0.8	40	0.4	25	25
0.8	0.8	40	0.6	25	25
0.8	0.8	40	0.8	25	25
0.8	0.8	80	0.2	25	25
0.8	0.8	80	0.4	25	25
0.8	0.8	80	0.6	25	25
0.8	0.8	80	0.8	25	25
0.8	0.8	160	0.2	25	25
0.8	0.8	160	0.4	25	25
0.8	0.8	160	0.6	25	25
0.8	0.8	160	0.8	25	25
0.8	0.8	320	0.2	25	25
0.8	0.8	320	0.4	25	25
0.8	0.8	320	0.6	25	25
0.8	0.8	320	0.8	25	25
0.8	0.8	640	0.2	25	25
0.8	0.8	640	0.4	25	25
0.8	0.8	640	0.6	25	25
0.8	0.8	640	0.8	25	25

Из таблицы 2 видно, что при алгоритм муравья работает при времени (количествах итераций) большим 40. Рассмотрим граф с рисунка 8.

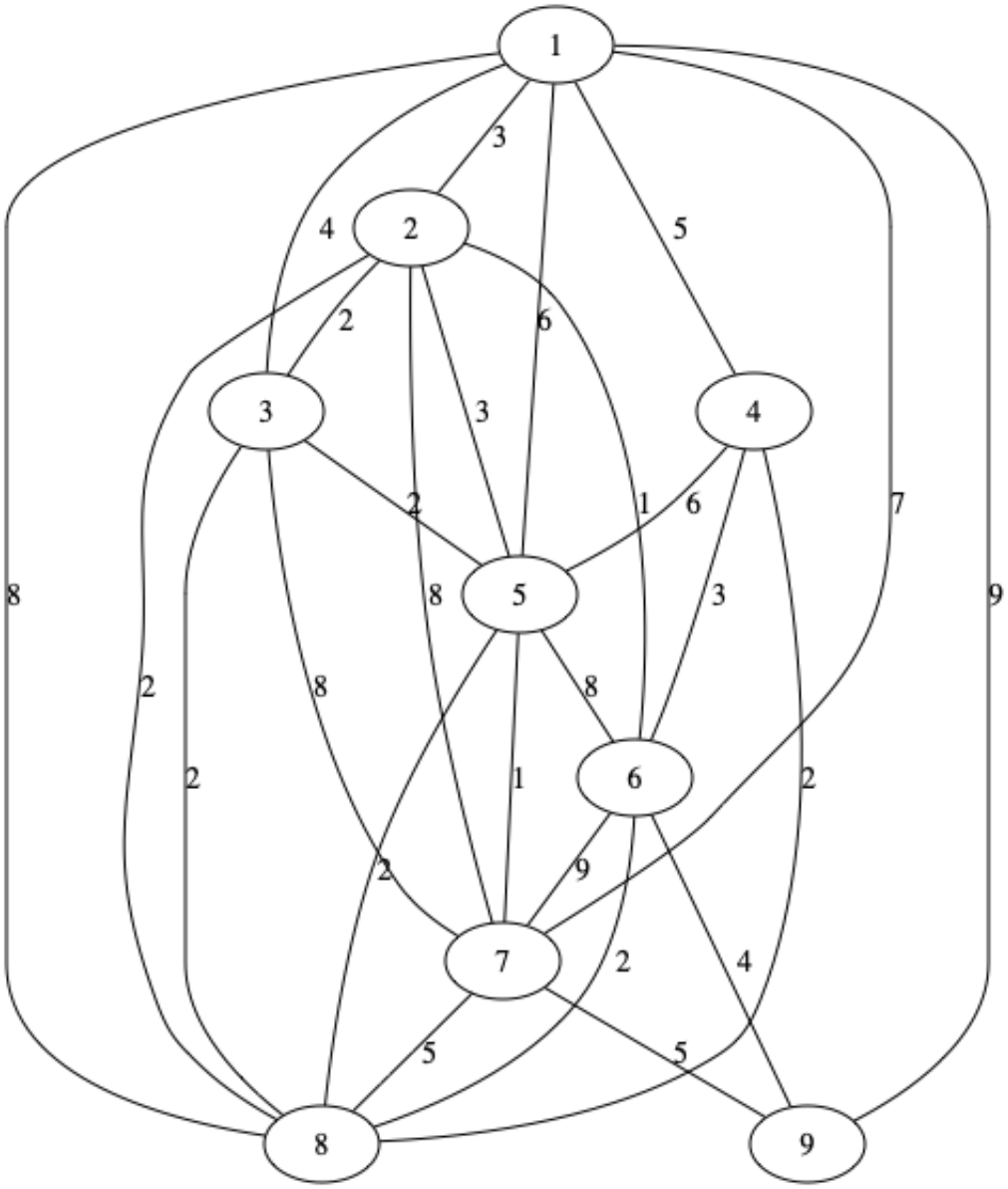

Рис. 8 - Граф 5, соответствующий тестовым данным 5.

Получим таблицу.

Таблица 3: Таблица, соответствующая графу на рисунке 8

α	β	time	ρ	Длина по алгоритму муравья	Длина по алгоритму полного перебора
0.2	0.2	10	0.2	43	25
0.2	0.2	10	0.4	43	25
0.2	0.2	10	0.6	43	25
0.2	0.2	10	0.8	43	25
0.2	0.2	20	0.2	38	25
0.2	0.2	20	0.4	38	25
0.2	0.2	20	0.6	38	25
0.2	0.2	20	0.8	38	25
0.2	0.2	40	0.2	35	25
0.2	0.2	40	0.4	35	25
0.2	0.2	40	0.6	35	25
0.2	0.2	40	0.8	35	25
0.2	0.2	80	0.2	30	25
0.2	0.2	80	0.4	30	25
0.2	0.2	80	0.6	30	25
0.2	0.2	80	0.8	30	25
0.2	0.2	160	0.2	29	25
0.2	0.2	160	0.4	29	25
0.2	0.2	160	0.6	29	25
0.2	0.2	160	0.8	29	25
0.2	0.2	320	0.2	28	25
0.2	0.2	320	0.4	28	25
0.2	0.2	320	0.6	28	25
0.2	0.2	320	0.8	28	25
0.2	0.2	640	0.2	26	25
0.2	0.2	640	0.4	26	25
0.2	0.2	640	0.6	26	25
0.2	0.2	640	0.8	26	25
...	...	...	...	...	...
0.2	0.4	320	0.2	28	25
0.2	0.4	320	0.4	28	25
0.2	0.4	320	0.6	28	25
0.2	0.4	320	0.8	28	25
0.2	0.4	640	0.2	26	25
0.2	0.4	640	0.4	26	25
0.2	0.4	640	0.6	26	25
0.2	0.4	640	0.8	26	25
...	...	...	...	...	...
0.2	0.8	160	0.2	25	25
0.2	0.8	160	0.4	25	25
0.2	0.8	160	0.6	25	25
0.2	0.8	160	0.8	25	25
0.2	0.8	320	0.2	25	25
0.2	0.8	320	0.4	25	25
0.2	0.8	320	0.6	25	25
0.2	0.8	320	0.8	25	25
0.2	0.8	640	0.2	25	25
0.2	0.8	640	0.4	25	25
0.2	0.8	640	0.6	25	25
0.2	0.8	640	0.8	25	25
...	...	...	...	...	...
0.4	0.2	160	0.2	25	25
0.4	0.2	160	0.4	25	25
0.4	0.2	160	0.6	25	25
0.4	0.2	160	0.8	25	25
0.4	0.2	320	0.2	25	25
0.4	0.2	320	0.4	25	25
0.4	0.2	320	0.6	25	25
0.4	0.2	320	0.8	25	25
0.4	0.2	640	0.2	25	25
0.4	0.2	640	0.4	25	25
0.4	0.2	640	0.6	25	25
0.4	0.2	640	0.8	25	25

α	β	time	ρ	Длина по алгоритму муравья	Длина по алгоритму полного перебора
...	...	...	...	...	...
0.6	0.2	10	0.2	31	25
0.6	0.2	10	0.4	31	25
0.6	0.2	10	0.6	31	25
0.6	0.2	10	0.8	31	25
0.6	0.2	20	0.2	31	25
0.6	0.2	20	0.4	31	25
0.6	0.2	20	0.6	31	25
0.6	0.2	20	0.8	31	25
0.6	0.2	40	0.2	31	25
0.6	0.2	40	0.4	31	25
0.6	0.2	40	0.6	31	25
0.6	0.2	40	0.8	31	25
0.6	0.2	80	0.2	31	25
0.6	0.2	80	0.4	31	25
0.6	0.2	80	0.6	31	25
0.6	0.2	80	0.8	31	25
0.6	0.2	160	0.2	31	25
0.6	0.2	160	0.4	31	25
0.6	0.2	160	0.6	31	25
0.6	0.2	160	0.8	31	25
0.6	0.2	320	0.2	31	25
0.6	0.2	320	0.4	31	25
0.6	0.2	320	0.6	31	25
0.6	0.2	320	0.8	31	25
0.6	0.2	640	0.2	29	25
0.6	0.2	640	0.4	29	25
0.6	0.2	640	0.6	29	25
0.6	0.2	640	0.8	29	25
...	...	...	...	...	...
0.6	0.4	320	0.2	31	25
0.6	0.4	320	0.4	31	25
0.6	0.4	320	0.6	31	25
0.6	0.4	320	0.8	31	25
0.6	0.4	640	0.2	29	25
0.6	0.4	640	0.4	25	25
0.6	0.4	640	0.6	25	25
0.6	0.4	640	0.8	25	25
...	...	...	...	...	...
0.6	0.6	640	0.2	25	25
0.6	0.6	640	0.4	25	25
0.6	0.6	640	0.6	25	25
0.6	0.6	640	0.8	25	25
...	...	...	...	...	...
0.6	0.8	640	0.4	25	25
0.6	0.8	640	0.6	25	25
0.6	0.8	640	0.8	25	25
...	...	...	...	...	...
0.8	0.2	640	0.2	25	25
0.8	0.2	640	0.4	25	25
0.8	0.2	640	0.6	26	25
0.8	0.2	640	0.8	26	25
...	...	...	...	...	...
0.8	0.8	640	0.4	26	25
0.8	0.8	640	0.6	26	25
0.8	0.8	640	0.8	26	25

Из таблицы 3 видно, что при алгоритм муравья работает при времени (количествах итераций) большим 160 и коэффициентах α, β 0.2, 0.8 или 0.4, 0.2 соотвественно.

И при итерациях от 640 и α 0.6 и β от 0.4.

И при итерациях от 640 и α большем 0.8 и β любое.

4.3 Выводы

Таким образом, оптимальные коэффициенты:

α	β	time	ρ
0.2	0.8	>160	любое
0.4	0.2	>160	любое
0.6	0.4	>640	любое
>0.8	любое	>640	любое

α – коэффициент стадности.
β – коэффициент жадности.
time – количество итераций.
ρ – скорость испарения феромонов.

Заключение

Таким образом было создано приложения для наглядного представления работы муравьиного алгоритма и проведен вычислительный эксперимент.

Выполнены следующие задачи.

1. Изучен существующих методов решения задачи коммивояжера.
2. Программно реализован алгоритм оптимизации подражанием муравьиной колонии.
3. Проведен вычислительный эксперимент, определены оптимальные коэффциенты.

Список литературы

[1] http://www.math.nsc.ru/LBRT/k5/OR-MMF/TSPr.pdf

[2] H. Mueller-Merbach. Zweimal travelling Salesman // DGOR-Bulletin, 1983

[3] Левитин А. Алгоритмы: введение в разработку и анализ // М.: Вильямс, 2006. – 575 с.

[4] Семинары по анализу алгоритмов.

[5] https://www.graphviz.org