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

Дисциплина: Анализ алгоритмов

Лабораторная работа № 6

ТЕМА
«Изучение муравьиного алгоритма»

Студент группы ИУ7-54,
Лозовский Алексей

2019 г.

Содержание

Введение

В данной работе речь пойдет о задаче Коммивояжера и способах ее решения. Идея задачи основана на поиске оптимального пути проходящего через заданные вершины хотя бы 1 раз.

Существующие алгоритмы можно использовать для составления маршрута человека, который должен посетить ряд пунктов и, в конце концов, вернуться в исходный пункт.Например, задача коммивояжера использовалась для составления маршрутов лиц, занимающихся выемкой монет из таксофонов. В этом случае вершинами являются места установки таксофонов и "базовый пункт". Стоимостью каждого ребра (отрезка маршрута) является время в пути между двумя точками (вершинами) на маршруте.

Еще одно применение – это задача о сверлильном станке. Сверлильный станок изготавливает металлические листы с некоторым количеством отверстий. Координаты отверстий известны. Необходимо найти кратчайший путь через все отверстия, а значит, и наименьшее время, затрачиваемое на изготовление одной детали.

1 Аналитическая часть

Решить задачу Коммивояжера можно решить следующими методами:

- Метод полного перебора;
- Эвристические методы.

1.1 Описание алгоритмов

Эвристический алгоритм - алгоритм решения задачи, включающий практический метод, не являющийся гарантированно точным или оптимальным, но достаточный для решения поставленной задачи. Позволяет ускорить решение задачи в тех случаях, когда точное решение не может быть найдено.

У алгоритмов данного типа можно выделить следующие достоинства:

- скорость. Когда нет времени производить масштабные вычисления, но необходимо получить первое приближение обычно применяют алгоритмы данного типа.

Основываясь на [3], главным недостатком можно отметить пониженную точность работы - эвристические алгоритмы не гарантируют нахождение глобального минимума.

В случае **полного перебора**, главным достоинством можно выделить точность работы алгоритмов, при этом основным недостатком будет являться время выполнения вычислений.

1.2 Муравьиный алгоритм

Одним из примеров эвристических алгоритомв является - муравьиный алгоритм.

Как описано в [4] идея алгоритма основана на принципе работы колонии муравьев. Колония муравьев рассматривается как многоагентная система, в которой каждый агент (муравей) функционирует автономно по очень простым правилам. В противовес почти примитивному поведению агентов, поведение всей системы получается разумным.

Каждый муравей определеяет 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:

- Случайнонсть – муравьи имеют случайную природу движения;

- 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;

- Целевая функция.

1.2.1 Математическая модель алгоритма

Пусть муравей обладает следующими характеристиками:

- зрение - определяет длину ребра;

- обоняние - чувствует феромон;

- память - запоминает маршрут, который прошел.

1) Введем целевую функцию $\eta = 1/D$, где D - расстояние до заданного пункта.
2) Считаются вероятности перехода в заданную точку по формуле(1):

$$P = \begin{cases} \frac{t^a \eta^b}{\sum_{i=1}^{m} t_i^a \eta_i^b}, \textit{вершина не была посещена ранее} \\ 0, \textit{иначе} \end{cases} \tag{1}$$

где a, b – настраиваемые параметры, t - концентрация феромона, причем $a + b = const$, а при $a = 0$ алгоритм вырождается в жадный.

3) Когда все муравьи завершили движение происходит обновление феромона по формуле (2):

$$t = \begin{cases} (1-p)(t_{prev} + \Delta t), \Delta t = Q/L_0, \textit{ребро посещено муравьем} \\ 0, \textit{иначе} \end{cases} \tag{2}$$

где
L_0 – длина пути на определенном участке пути,
Q – настраивает концентрацию нанесения/испарения феромона.

1.3 Задание на выполнение лабораторной работы

Цель: изучить муравьиный алгоритм на материале решения задачи Коммивояжера

Задачи:

- дать постановку задачи;
- описать методы полного перебора и эвристический, основанный на муравьином алгоритме;
- реализовать методы;
- выбрать и подготовить классы данных;
- провести параметризацию метода, основанного на муравьином алгоритме;
- интерпретировать результаты и сравнить их с результатами метода полного перебора.

1.4 Вывод по аналитическому разделу

По итогам аналитического раздела было выполнено следующее:

- поставлена задача;
- описаны методы полного перебора и эвристический, основанный на муравьином алгоритме.

2 Конструкторская часть

В конструкторской части необходимо сделать следующее:

- определить структуру кода и способ представления данных;
- представить схему муравьиного алгоритма.

2.1 Структура кода и представление данных

Колония муравьев представлена массивом объектов класса *Ant*, который хранит в себе список посещенных вершин, длину пройденного пути, количество вершин, необходимых к посещению, а также карту, по которой двигаться. Карта описывается отдельным классом *Map*, который хранит в себе информацию о посещении ребер, матрицу ребер и матрицу феромона на каждом ребре.

2.2 Разработка алгоритмов

Алгоритм разбит на следующие этапы:

- день - на этом этапе муравьи совершают обход по карте и прокладывают путь;
- ночь - этап обновления феромона, наступает после завершения работы последнего муравья.

2.2.1 Схема работы муравьиного алгоритма

На рис. 1 отображена работа одного муравья.

Рис. 1. Схема работы одного муравья.

Описание этапов:

- инициализация среды - установка значений α, β, P, Q ;
- инициализация муравья - установка муравья в стартовый город;
- цикл продолжает работу до тех пор, пока все вершины не будут посещены, внутри тела цикла высчитывается вероятность посещения следующего города по формуле (1), для определения конкретного города применяется метод рулетки, который случайно выбирает значение $0, \ldots, 1$, на основе полученного значения определяется следующий город.

Описанный выше алгоритм применяется N раз в зависимости от количества вершин. Каждый муравей помещается в отдельную вершину на карте, после чего ищет маршрут. Когда последний муравей завершил обход всех вершин, производится поиск оптимального из полученных маршрутов, после чего феромон обновляется и в случае, если полученный результат не удовлетворяет поставленной задаче, алгоритм запускается заново.

Ниже представлена работа алгоритма для всей колонии.

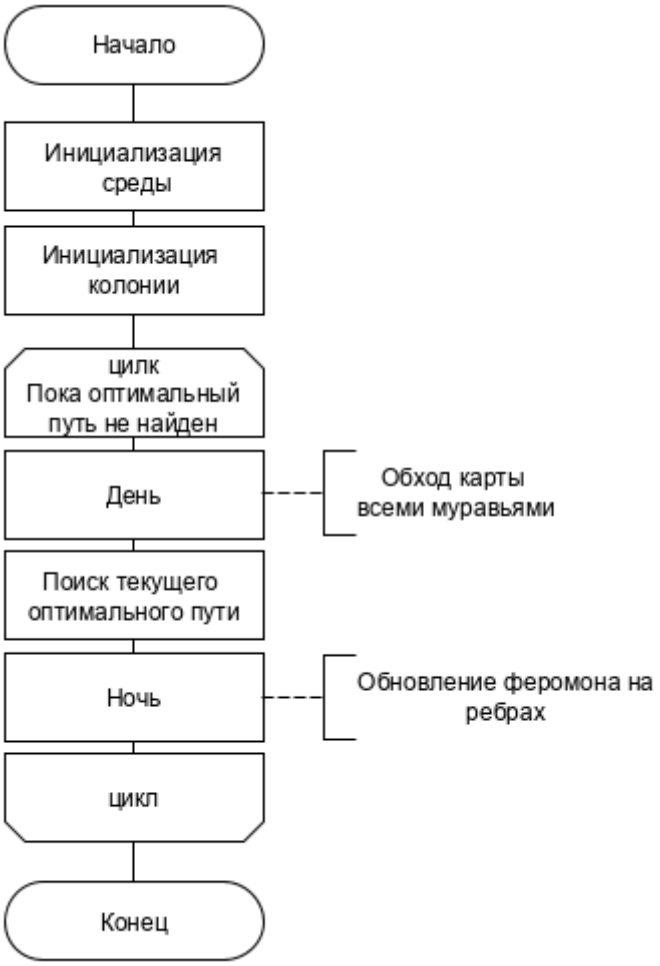

Рис. 2. Схема работы колонии.

Ниже представлена idef-диаграмма муравьиного алгоритма.

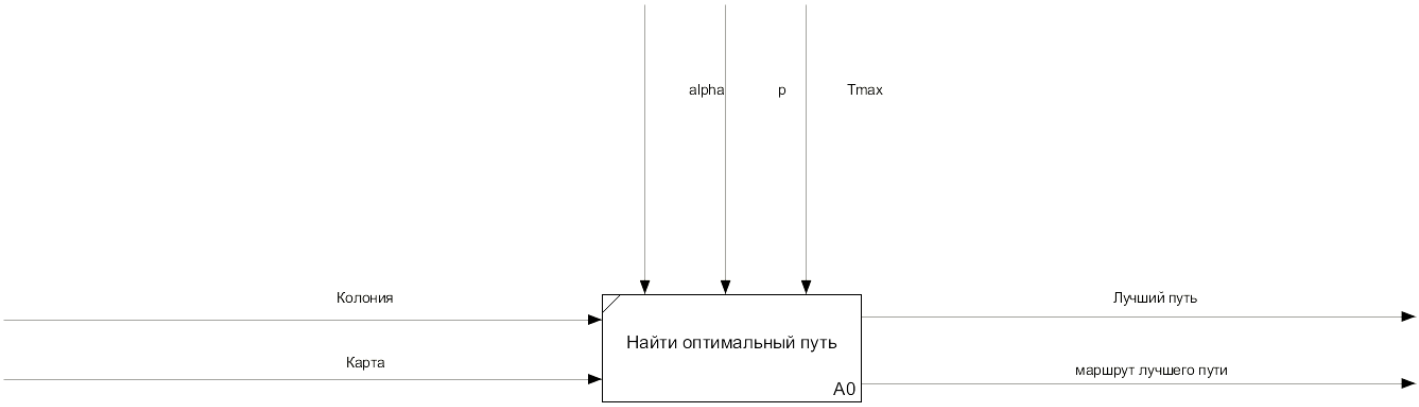

Рис. 3. idef0-диаграмма.

2.3 Выводы по конструкторскому разделу

Таким образом, была определена структура кода и способ представления данных, также была представлена схема муравьиного алгоритма.

3 Технологическая часть

3.1 Требования к программному обеспечению

Программа должна обрабатывать матрицу смежностей методами полного перебора и эвристическим, основанным на муравьином алгоритме, подбирать параметры, на основе которых производятся оптимальные вычисления, выводить заданную матрицу смежностей, результат работы полного перебора и муравьиного алгоритма.

3.2 Средства реализации

Поскольку программа реализована в структурном стиле, из языков, поддерживающих его был выбран C++, стандарта 2011 года[5], так как на нем имеется наибольший опыт работы.

3.3 Листинг кода

Ниже представлены листинги кода для класса муравья(Листинг 1), для класса карты(Листинг 2), и реализация полного перебора(Листинг 3).

Листинг 1. Класс муравья

```
class Ant{

public:
    Ant();
    Ant(Map*);
    Ant(Map*, size_t);

    void findPath();
    void findPath(size_t);  // параметр вершина старта

    size_t getPathDistance();

    void addToVisited(size_t); // пометит город, как просмотренный
    size_t countVisited();

    size_t chooseNextCity();
    double countPossibility(size_t, size_t);
    double mult(size_t, size_t);

    bool isVisited(size_t);
    bool isFinished();

    size_t last(); // вернет последний посещенный город
    size_t first(); // вернет первый элемент пути

    void update();

    void show();
```

```
private:
    path_t visited; // посещенные города

    size_t length; // длина пути

    size_t vert; // Количество вершин

    Map *map;

    std::mt19937 choice;
    std::uniform_real_distribution<> range;
};
```

Листинг 2. Класс карты

```
class Map
{
public:
    double a;
    double b;


    Map();
    Map(map_t<size_t>&, double, double, double);

    void update(); //обновление феромона

    double getPheromone(size_t, size_t);
    size_t getDistance(size_t, size_t);

    size_t countCities(); //вернет число вершин в маршруте

    double f(size_t); // целевая функция

    void setVisited(size_t, size_t);
    bool isVisited(size_t, size_t);

    void restore(double&); // восстановление феромона

private:
    map_t<size_t> map;
    map_t<bool> visited;
    map_t<double> pheromone;


    double p; // коэфф скорости испарение
    double q; // концентрация нанесения феромона
};
```

Листинг 3. Рекурсивная реализация полного перебора

```
template <typename T>
```

```
using map_t = std::vector<std::vector<T>>;

template <typename T>
void find_best(const map_t<T> map, Path &path, Path &best){

    if (path.isFinished()){

        // добавить первый элемент
        size_t first = path.first();
        size_t last = path.last();

        // добавить к длине пути длину расстояния до старта
        path.addToVisited(first);
        path() += map[last][first];

        if (best() == 0 or path() < best())
        best = path;

        return;
    }


size_t start = path.last();
size_t distance;
Path tmp_path;


for (size_t target = 0; target < path.verticles(); target++){

    // Вычисляем расстояние между стартовым и целевым городом
    distance = map[start][target];
    tmp_path = path;

    if (!tmp_path.isVisited(target) and distance != 0){

        tmp_path.addToVisited(target); // добавляем город в посещенные
        tmp_path() += distance; // увеличиваем длину пути

        find_best(map, tmp_path, best); // уходим на рекурсивное дно
        }
    }
}

template <typename T>
Path get_best_path(const map_t<T> map, size_t start){
    Path best;

    Path path = Path(map.size());
    path.addToVisited(start);

    find_best(map, path, best);

    return best;
```

```
}
```

3.4 Выводы по конструкторскому разделу

Были описаны требования к ПО, приведен листинг кода с классами муравья и карты, был предоставлен листинг с реализацией метода полного перебора, а также был выбран язык программирования.

4 Экспериментальная часть

4.1 Постановка эксперимента

В эвристическом методе, основанном на 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. Для простоты нахождения точного результата методом перебора будем рассматривать матрицы размером 10×10, элементов в строке и столбце, соответственно, так как на матрицах большего размера, время ожидания метода доходит до получаса.

В качестве первого класса данных выделим матрицу смежностей, в которой все значения незначительно отличаются друг от друга, например, в диапазоне от $[0, 10]$. Вторым классом будут матрицы, где значения могут значительно отличаться, например $[1, 15000]$.

Будем запускать муравьиный алгоритм для всех значений $\alpha, P \in [0, 1]$, с шагом $= 0.1$ до тех пор пока для каждого набора значений не будет найдено точное значение, в случае превышения допустимого количества итераций работа алгоритма для данных параметров завершится.

В результате тестирования будет выведена таблица со значениями $\alpha, \beta, P,$, $iters$, $dist$, где $iters$ - число истераций, за которое алгоритм нашел оптимальный путь, а dist - путь, который был найден, а α, β, p - настроечные параметры.

Поиск коэффициентов произведен на 64-разрядной операционной системе, процессоре Intel Core i5-8250U, 1.6 GHz.

4.2 Примеры работы

Ниже будут представлены результаты работы алгоритма для двух классов данных.

4.2.1 Класс данных №1

$$Map = \begin{pmatrix} 0, & 3 & 10 & 9 & 8 & 2 & 1 & 7 & 2 & 10 \\ 3, & 0 & 8 & 10 & 3 & 9 & 6 & 8 & 4 & 2 \\ 10, & 8 & 0 & 10 & 3 & 2 & 7 & 3 & 10 & 7 \\ 9, & 10 & 10 & 0 & 3 & 2 & 2 & 5 & 10 & 4 \\ 8, & 3 & 3 & 3 & 0 & 3 & 6 & 8 & 5 & 1 \\ 2, & 9 & 2 & 2 & 3 & 0 & 10 & 10 & 2 & 9 \\ 1, & 6 & 7 & 2 & 6 & 10 & 0 & 10 & 4 & 8 \\ 7, & 8 & 3 & 5 & 8 & 10 & 10 & 0 & 6 & 2 \\ 2, & 4 & 10 & 10 & 5 & 2 & 4 & 6 & 0 & 10 \\ 10, & 2 & 7 & 4 & 1 & 9 & 8 & 2 & 10 & 0 \end{pmatrix}$$

Таблица 1. Таблица коэффициентов для класса данных №1

a	b	p	iters	distance	a	b	p	iters	distance
0,	1	0	50	22	0.4,	0.6	0	20	22
0,	1	0.1	50	22	0.4,	0.6	0.1	20	22
0,	1	0.2	50	22	0.4,	0.6	0.2	20	22
0,	1	0.3	50	22	0.4,	0.6	0.3	20	22
0,	1	0.4	50	22	0.4,	0.6	0.4	20	22
0,	1	0.5	50	22	0.4,	0.6	0.5	20	22
0,	1	0.6	50	22	0.4,	0.6	0.6	20	22
0,	1	0.7	50	22	0.4,	0.6	0.7	49	22
0,	1	0.8	50	22	0.4,	0.6	0.8	20	22
0,	1	0.9	50	22	0.4,	0.6	0.9	20	22
0,	1	1	7	22	0.4,	0.6	1	8	22
0.1,	0.9	0	7	22	0.5,	0.5	0	20	22
0.1,	0.9	0.1	24	22	0.5,	0.5	0.1	20	22
0.1,	0.9	0.2	174	22	0.5,	0.5	0.2	20	22
0.1,	0.9	0.3	174	22	0.5,	0.5	0.3	20	22
0.1,	0.9	0.4	24	22	0.5,	0.5	0.4	20	22
0.1,	0.9	0.5	24	22	0.5,	0.5	0.5	20	22
0.1,	0.9	0.6	24	22	0.5,	0.5	0.6	20	22
0.1,	0.9	0.7	174	22	0.5,	0.5	0.7	20	22
0.1,	0.9	0.8	174	22	0.5,	0.5	0.8	85	22
0.1,	0.9	0.9	119	22	0.5,	0.5	0.9	163	22
0.1,	0.9	1	22	22	0.5,	0.5	1	26	22
0.2,	0.8	0	24	22	0.6,	0.4	0	20	22
0.2,	0.8	0.1	24	22	0.6,	0.4	0.1	20	22
0.2,	0.8	0.2	24	22	0.6,	0.4	0.2	20	22
0.2,	0.8	0.3	24	22	0.6,	0.4	0.3	20	22
0.2,	0.8	0.4	174	22	0.6,	0.4	0.4	20	22
0.2,	0.8	0.5	174	22	0.6,	0.4	0.5	20	22
0.2,	0.8	0.6	24	22	0.6,	0.4	0.6	20	22
0.2,	0.8	0.7	24	22	0.6,	0.4	0.7	124	22
0.2,	0.8	0.8	24	22	0.6,	0.4	0.8	49	22
0.2,	0.8	0.9	30	22	0.6,	0.4	0.9	14	22
0.2,	0.8	1	2	22	0.6,	0.4	1	65	22
0.3,	0.7	0	20	22	0.7,	0.3	0	20	22
0.3,	0.7	0.1	20	22	0.7,	0.3	0.1	20	22
0.3,	0.7	0.2	20	22	0.7,	0.3	0.2	20	22
0.3,	0.7	0.3	20	22	0.7,	0.3	0.3	20	22
0.3,	0.7	0.4	20	22	0.7,	0.3	0.4	20	22
0.3,	0.7	0.5	20	22	0.7,	0.3	0.5	20	22
0.3,	0.7	0.6	20	22	0.7,	0.3	0.6	20	22
0.3,	0.7	0.7	20	22	0.7,	0.3	0.7	20	22
0.3,	0.7	0.8	20	22	0.7,	0.3	0.8	13	22
0.3,	0.7	0.9	102	22	0.7,	0.3	0.9	16	22
0.3,	0.7	1	34	22	0.7,	0.3	1	7	22

a	b	p	iters	distance
0.4,	0.6	0	20	22
0.4,	0.6	0.1	20	22
0.4,	0.6	0.2	20	22
0.4,	0.6	0.3	20	22
0.4,	0.6	0.4	20	22
0.4,	0.6	0.5	20	22
0.4,	0.6	0.6	20	22
0.4,	0.6	0.7	49	22
0.4,	0.6	0.8	20	22
0.4,	0.6	0.9	20	22
0.4,	0.6	1	8	22
0.5,	0.5	0	20	22
0.5,	0.5	0.1	20	22
0.5,	0.5	0.2	20	22
0.5,	0.5	0.3	20	22
0.5,	0.5	0.4	20	22
0.5,	0.5	0.5	20	22
0.5,	0.5	0.6	20	22
0.5,	0.5	0.7	20	22
0.5,	0.5	0.8	85	22
0.5,	0.5	0.9	163	22
0.5,	0.5	1	26	22
0.6,	0.4	0	20	22
0.6,	0.4	0.1	20	22
0.6,	0.4	0.2	20	22
0.6,	0.4	0.3	20	22
0.6,	0.4	0.4	20	22
0.6,	0.4	0.5	20	22
0.6,	0.4	0.6	20	22
0.6,	0.4	0.7	124	22
0.6,	0.4	0.8	49	22
0.6,	0.4	0.9	14	22
0.6,	0.4	1	65	22

4.2.2 Класс данных №2

$$Map = \begin{pmatrix} 0, & 13220 & 5777 & 10272 & 2509 & 12737 & 11202 & 13053 & 2014 & 3140 \\ 13220, & 0 & 9305 & 8955 & 3974 & 12863 & 4135 & 509 & 13839 & 2603 \\ 5777, & 9305 & 0 & 10978 & 5521 & 9191 & 13678 & 3453 & 6444 & 13320 \\ 10272, & 8955 & 10978 & 0 & 13342 & 10270 & 8814 & 14032 & 1896 & 6665 \\ 2509, & 3974 & 5521 & 13342 & 0 & 6897 & 3215 & 1483 & 11523 & 6752 \\ 12737, & 12863 & 9191 & 10270 & 6897 & 0 & 9091 & 5338 & 9966 & 6815 \\ 11202, & 4135 & 13678 & 8814 & 3215 & 9091 & 0 & 3973 & 6879 & 10087 \\ 13053, & 509 & 3453 & 14032 & 1483 & 5338 & 3973 & 0 & 5463 & 8252 \\ 2014, & 13839 & 6444 & 1896 & 11523 & 9966 & 6879 & 5463 & 0 & 4997 \\ 3140, & 2603 & 13320 & 6665 & 6752 & 6815 & 10087 & 8252 & 4997 & 0 \end{pmatrix}$$

Таблица 2. Таблица коэффициентов для класса данных №2

a	b	p	iters	distance	a	b	p	iters	distance
0,	1	0	12	40402	0.4,	0.6	0	28	40402
0,	1	0.1	12	40402	0.4,	0.6	0.1	28	40402
0,	1	0.2	12	40402	0.4,	0.6	0.2	206	40402
0,	1	0.3	62	40402	0.4,	0.6	0.3	206	40402
0,	1	0.4	62	40402	0.4,	0.6	0.4	28	40402
0,	1	0.5	62	40402	0.4,	0.6	0.5	28	40402
0,	1	0.6	62	40402	0.4,	0.6	0.6	10	40402
0,	1	0.7	62	40402	0.4,	0.6	0.7	10	40402
0,	1	0.8	62	40402	0.4,	0.6	0.8	10	40402
0,	1	0.9	62	40402	0.4,	0.6	0.9	10	40402
0,	1	1	62	40402	0.4,	0.6	1	10	40402
0.1,	0.9	0	62	40402	0.5,	0.5	0	28	40402
0.1,	0.9	0.1	62	40402	0.5,	0.5	0.1	28	40402
0.1,	0.9	0.2	62	40402	0.5,	0.5	0.2	28	40402
0.1,	0.9	0.3	62	40402	0.5,	0.5	0.3	28	40402
0.1,	0.9	0.4	62	40402	0.5,	0.5	0.4	54	40402
0.1,	0.9	0.5	62	40402	0.5,	0.5	0.5	10	40402
0.1,	0.9	0.6	62	40402	0.5,	0.5	0.6	10	40402
0.1,	0.9	0.7	62	40402	0.5,	0.5	0.7	10	40402
0.1,	0.9	0.8	62	40402	0.5,	0.5	0.8	10	40402
0.1,	0.9	0.9	62	40402	0.5,	0.5	0.9	10	40402
0.1,	0.9	1	41	40402	0.5,	0.5	1	10	40402
0.2,	0.8	0	62	40402	0.6,	0.4	0	28	40402
0.2,	0.8	0.1	62	40402	0.6,	0.4	0.1	28	40402
0.2,	0.8	0.2	62	40402	0.6,	0.4	0.2	28	40402
0.2,	0.8	0.3	62	40402	0.6,	0.4	0.3	54	40402
0.2,	0.8	0.4	62	40402	0.6,	0.4	0.4	127	40402
0.2,	0.8	0.5	92	40402	0.6,	0.4	0.5	89	40402
0.2,	0.8	0.6	62	40402	0.6,	0.4	0.6	26	40402
0.2,	0.8	0.7	62	40402	0.6,	0.4	0.7	150	40402
0.2,	0.8	0.8	10	40402	0.6,	0.4	0.8	139	40402
0.2,	0.8	0.9	10	40402	0.6,	0.4	0.9	193	40402
0.2,	0.8	1	10	40402	0.6,	0.4	1	63	40402
0.3,	0.7	0	28	40402	0.7,	0.3	0	193	40402
0.3,	0.7	0.1	28	40402	0.7,	0.3	0.1	240	40402
0.3,	0.7	0.2	206	40402	0.7,	0.3	0.2	192	40402
0.3,	0.7	0.3	102	40402	0.7,	0.3	0.3	193	40402
0.3,	0.7	0.4	28	40402	0.7,	0.3	0.4	193	40402
0.3,	0.7	0.5	10	40402	0.7,	0.3	0.5	35	40402
0.3,	0.7	0.6	10	40402	0.7,	0.3	0.6	35	40402
0.3,	0.7	0.7	10	40402	0.7,	0.3	0.7	60	40402
0.3,	0.7	0.8	10	40402	0.7,	0.3	0.8	35	40402
0.3,	0.7	0.9	10	40402	0.7,	0.3	0.9	80	40402
0.3,	0.7	1	10	40402	0.7,	0.3	1	35	40402

a	b	p	iters	distance
0.8,	0.2	0	35	40402
0.8,	0.2	0.1	58	40402
0.8,	0.2	0.2	35	40402
0.8,	0.2	0.3	60	40402
0.8,	0.2	0.4	60	40402
0.8,	0.2	0.5	58	40402
0.8,	0.2	0.6	60	40402
0.8,	0.2	0.7	58	40402
0.8,	0.2	0.8	35	40402
0.8,	0.2	0.9	58	40402
0.8,	0.2	1	96	40402
0.9,	0.1	0	58	40402
0.9,	0.1	0.1	35	40402
0.9,	0.1	0.2	58	40402
0.9,	0.1	0.3	60	40402
0.9,	0.1	0.4	80	40402
0.9,	0.1	0.5	80	40402
0.9,	0.1	0.6	60	40402
0.9,	0.1	0.7	41	40402
0.9,	0.1	0.8	3	40402
0.9,	0.1	0.9	3	40402
0.9,	0.1	1	3	40402
1,	0	0	41	40402
1,	0	0.1	41	40402
1,	0	0.2	41	40402
1,	0	0.3	41	40402
1,	0	0.4	41	40402
1,	0	0.5	41	40402
1,	0	0.6	41	40402
1,	0	0.7	28	40402
1,	0	0.8	41	40402
1,	0	0.9	41	40402
1,	0	1	64	40402

4.3 Вывод по экспериментальной части

Таким образом, на основе полученных таблиц можно сделать вывод, что при классе данных, содержащем приблизительно равные значения наилучими наборами стали ($\alpha = 0.2, \beta = 0.8, P = 1$), при данных значениях алгоритм нашел лучший путь за 2 запуска. При наборах ($\alpha = 0, \beta = 1, P = 1$), ($\alpha = 0.7, \beta = 0.3, P = 1$) алгоритм нашел путь за 7 итераций.

Для второго класса данных было определено, что при ($\alpha = 0.9, \beta = 0.1, P = 0.8$), ($\alpha = 0.9, \beta = 0.1, P = 0.9$), ($\alpha = 0.9, \beta = 0.1, P = 1$) алгоритм отработал за 3 запуска.

Заключение

Таким образом, в ходе лабораторной работы было сделано следующее:

- дана постановку задачи;
- описаны методы полного перебора и эвристический, основанный на муравьином алгоритме;
- реализованы методы;
- выбраны и подготовлены классы данных;
- проведена параметризация метода, основанного на муравьином алгоритме;

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 литературы

[1] M. Dorigo., "Ant Algorithms for Discrete Optimization"

[2] С. Гудман., "Введение в разработку и анализ алгоритмов"

[3] Д. Д. Ульман. "Структуры данных и алгоритмы"

[4] E. Bonabeau, M. Dorigo et G. Theraulaz, 1999., "Swarm Intelligence: From Natural to Artificial Systems"

[5] Стандарт языка C++11 согласно ISO. [Электронный ресурс]. URL: https://isocpp.org/std/the-standard