Министерство образования Российской Федерации
Московский Государственный Технический
Университет им. Н.Э. Баумана

Отчет по лабораторной работе №7
По курсу «Анализ алгоритмов»

Тема: «Муравьиный алгоритм»

Студент: **Бадалян Д.А.**
Группа: **ИУ7-51**
Преподаватель: **Волкова Л.Л.**

Москва, 2017

Постановка задачи

1. реализовать муравьиный алгоритм;
2. сравнить работу алгоритма при разных значениях параметров, задающих веса феромона и времени жизни колонны.

Реализация

Суть алгоритма

Муравьиный алгоритм (алгоритм оптимизации подражанием муравьиной колонии) — один из эффективных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания и представляет собой метаэвристическую оптимизацию.

Моделирование поведения муравьев связано с распределением феромона на тропе — ребре графа в задаче коммивояжера. При этом вероятность включения ребра в маршрут отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его ребрах, следовательно, большее количество муравьев будет включать его в синтез собственных маршрутов.

Программная реализация алгоритма

Листинг

```
import random as rnd

MAX_DIS = 10  # maximum distance
MIN_DIS = 1  # minimum distance


m = 5  # count of ants and cities
e = 2  # count of elite ants
```

```
9
10 a = 2
11 b = 1
12 Q = MIN_DIS * m
13 t_max = 500
14 p = 0.5
15
16 def get_desire_matr(m):
17     res = []
18     n = len(m)
19     for i in range(n):
20         temp = []
21         for j in range(n):
22             if m[i][j] == 0:
23                 temp.append(0)
24             else:
25                 temp.append(1/m[i][j])
26         res.append(temp)
27     return res
28
29 def update_feramon(p, teta_e, teta_k, teta):
30     res = []
31     n = len(teta)
32     for i in range(n):
33         temp = []
34         for j in range(n):
35             temp.append((1 - p)*teta[i][j] + teta_k[i][j] +
                   teta_e)
36         res.append(temp)
37     return res
38
39 def aco(m, e, d, t_max, alpha, beta, p, q):
40     nue = get_desire_matr(d)
41     teta = [[rnd.uniform(0,1) for i in range(m)] for j in
          range(m)]
42     T_min = None
43     L_min = None
44
45     t = 0
46
47     while t < t_max:
48         teta_k = [[0 for i in range(m)] for j in range(m)]
49
```

```
        for k in range(m):
            Tk = [k]
            Lk = 0
            i = k

            while len(Tk) != m:
                J = [r for r in range(m)]
                for c in Tk:
                    J.remove(c)

                P = [0 for a in J]

                for j in J:
                    if d[i][j] != 0:
                        buf = sum((teta[i][l] ** alpha) * (
                            nue[i][l] ** beta) for l in J)
                        P[J.index(j)] = (teta[i][j] **
                            alpha) * (nue[i][j] ** beta) /
                            buf
                    else:
                        P[J.index(j)] = 0

                Pmax = max(P)
                if Pmax == 0:
                    break

                index = P.index(Pmax)
                Tk.append(J[index])
                Lk += d[i][J[index]]
                i = J.pop(index)

            if L_min is None or (Lk + d[Tk[0]][Tk[-1]]) <
                L_min:
                L_min = Lk + d[Tk[0]][Tk[-1]]
                T_min = Tk

            for g in range(len(Tk) - 1):
                a = Tk[g]
                b = Tk[g + 1]
                teta_k[a][b] += q / Lk

        teta_e = (e * q / L_min) if L_min else 0
        teta = update_feramon(p, teta_e, teta_k, teta)
```

```
89            t += 1
90
91        return T_min, L_min
```

Эксперимент

В качестве первого эксперимента проверяется эффективность работы реализованного алгоритма в зависимости от параметров α β в формуле (1).
Входные данные:

- количество итераций $t_max = 100; m = 100;$

- α принимает значения от 0 до 1 с шагом 0.1 (при $\alpha = 0$ лгоритм вырождается до жадного алгоритма (будет выбран ближайший город));

- β принимает значения от 1 до 0 с шагом 0.1.

Полученные данные представлены в Таблице 1.

Таблица 1. Результаты эксперимента 1.

α	β	Длина найденного кратчайшего маршрута
0.0	1.0	413
0.1	0.9	401
0.2	0.8	345
0.3	0.7	343
0.4	0.6	309
0.5	0.5	309
0.6	0.4	304
0.7	0.3	304
0.8	0.2	304
0.9	0.1	304
1.0	0.0	304

В качестве второго эксперимента проверяется эффектривность работы реализованного алгоритма в зависимости от количества итераций (времени жизни колонии муравьев).
Входные данные:

- $\alpha = 0.5;$

- $\beta = 0.5$;
- размерность матрицы m = 50;

- количество итераций $t_max()1001000100$.

Полученные данные приведены в Таблице 2.

Таблица 2. Результаты эксперимента 2.

Время жизни колонии	Длина найденного кратчайшего маршрута
100	425
200	409
300	409
400	408
500	388
600	388
700	385
800	384
900	377
1000	377

Выводы из эксперимента

По результатам исследования результатов муравьиного алгоритма на разных значениях "жадности"и "стадности"можно прийти в выводу, о том, что эффективность повышается в случае увеличения коэффициента "жадности"(при $\alpha = 0$ муравей просто будет выбирать призжайший к нему непосещенный город). Так же можно сделать вывод, что выбор $\alpha = 0.5$ и больше ($\beta = 0.5$ и меньше соответственно) несущественно будет влиять на результат работы алгоритма.
По результатам эксперимента с поиском кратчайшего маршрута с разным временем жизни колонии муравьев подтвердилось предположение, о том, что более точный результат может быть получен на большом количестве итераций (времени жизни колонии муравьев).

Вывод

В ходе лабораторной работы был реализован муравьиный алгоритм на языке программирования Python и проведены сравнения работы алгоритма при разных значениях параметров, задающих веса феромона и времени жизни колонны.