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

Отчёт по лабораторной работе № 1
По курсу: "Анализ алгоритмов"
Тема:"Муравьиный алгоритм."

Студент: Орехова Е.О. ИУ7-51

Преподаватель: Волкова Л.Л.

6 марта 2018 г.

Содержание

1 Постановка задачи

Реализовать муравьиный алгоритм на языке программирования. Сравнить работу алгоритма при разных значениях параметров задающих веса феромона и времени жизни колонии

2 Идея

Муравьиный алгоритм - один из эффективных полиномиальных 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. С полным разбором алгоритма можно ознакомиться в книге М.В.Ульянова «Ресурсно- эффективные компьютерные алгоритмы. Разработка и анализ»

Локальные правила поведения муравьев при выборе пути:

- муравьи имеют собственную «память». Поскольку каждый город может быть посещен только один раз, у каждого муравья сеть список уже посещенных городов — список запретов. Обозначим через $J_{i,k}$ список городов, которые необходимо посетить муравью k , находящемуся в городе i;

- муравьи обладают «зрением» — видимость есть эвристическое желание посетить город j , если муравей находится в городе i . Будем считать, что видимость обратно пропорциональна расстоянию между городами i и j - D_{ij}

$$\eta_{ij} = \frac{1}{D_{ij}}; \tag{1}$$

- муравьи обладают «обонянием» — они могут улавливать след феромона, подтверждающий желание посетить город j из города i, на основании опыта других муравьев. Количество феромона на ребре (i,j) в момент времени t обозначим через $\tau_{ij}(t)$.

вероятность перехода k-ого муравья из города i в город j:

$$P_{ij,k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum\limits_{l=J_{i,k}} [\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}} & j \in J_{i,k} \\ 0 & j \notin J_{i,k} \end{cases} \tag{2}$$

где α, β - коэффициенты, определяющие «стадность» алгоритма и «жадность» алгоритма).

в конце каждого похода обновляется значение феромона на дорогах, используется следующая формула:

$$\tau_{ij}(t+1) = (1-p) * \tau_{ij}(t) + \Delta\tau_{ij}(t); \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \quad (3)$$

где

$$\Delta\tau_{ij,k}(t) \quad = \{ \begin{matrix} \frac{Q}{L_k(t)}, & (i,j) \in T_k(t) \\ 0, & (i,j) \notin T_k(t) \end{matrix} \} \quad (4)$$

p - коэффициент испарения феромона; Q - параметр, имеющий значение порядка длины оптимального пути; L_k - длина маршруту, пройденная муравьем k к моменту времени t; m - количество муравьев в колонии.

3 Реализация

Листинг 1: Муравьиный алгоритм

```
static void gogo_ant(ref ant_t ant, matrix_t adj_mat,
   matrix_t weight, matrix_t d_pheromon, int q)
{
        int N = weight.n;
        int i = 1;
        int next = 0;
        array_t prob = create_array(N);
        while (i < N)
        {
                double sum_weight = 0;
                for (int j = 0; j < N; j++)
                        sum_weight +=
                           weight.matr[ant.curr_city,j]
                           * ant.Jk.arr[j];
                for (int j = 0; j < N; j++)
                {
                        prob.arr[j] =
                           weight.matr[ant.curr_city,j]
                           / sum_weight * ant.Jk.arr[j];
                }
                next = choose_next(prob);
                ant.curr_city = next;
```

```
                ant.Jk.arr[next] = 0;
                ant.route.arr[i++] = next;
        }
        ant.Lk = (int)(length_of_route(adj_mat,
           ant.route));
        add_pheromon(d_pheromon, ant.route, ant.Lk, q);
}

static solution_t solve(matrix_t adj_mat, double a,
   double b, double p, int q, int t_max)
{
        int N = adj_mat.n;
        ant_t[] ants = create_ant_array(N); //array of
           ants, 1 per city;

        matrix_t pheromon = create_matrix(N, N);
        for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                        pheromon.matr[i,j] = 0.5;

        matrix_t visib = create_matrix(N, N);
        for (int i = 0; i < N; i++)
                for (int j = i; j < N; j++)
                        if (i != j)
                        {
                                visib.matr[i,j] = 1 /
                                   adj_mat.matr[i,j];
                                visib.matr[j,i] =
                                   visib.matr[i,j];
                        }
                        else
                                visib.matr[i,j] = 1000;

        matrix_t weight = create_matrix(N, N);
        recalc_weight(ref weight, pheromon, visib, a, b);

        matrix_t d_pheromon = create_matrix(N, N);
        for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
        d_pheromon.matr[i,j] = 0;

        int best_l = int.MaxValue;
        array_t route = create_array(N);
```

```
        for (int t = 0; t < t_max; t++)
        {
                for (int k = 0; k < N; k++)
                {
                        init_ant(ref ants[k]);
                        gogo_ant(ref ants[k], adj_mat,
                           weight, pheromon, q);
                }
                int best = -1;
                for (int i = 0; i < N; i++)
                        if (ants[i].Lk < best_l)
                        {
                                best = i;
                                best_l = ants[i].Lk;
                        }
                if (best != -1)
                copy_array(route, ants[best].route);

                recalc_pheromon(ref pheromon, ref
                   d_pheromon, p);
                recalc_weight(ref weight, pheromon,
                   visib, a, b);
        }

        solution_t solv = new solution_t();
        solv.l = best_l;
        solv.route = route;
        return solv;
}
```

4 Эксперимент

Alpha	Betta	Длина
0	1	112
0,1	0,9	105
0,2	0,8	90
0,3	0,7	93
0,4	0,6	93
0,5	0,5	92
0,6	0,4	90
0,7	0,3	95
0,8	0,2	103
0,9	0,1	110
1	0	159

Рис. 1: Поиск наименьшей длины в зависимости от α и β.
Наилучший результат алгоритм даёт при $\alpha = 0.6$ и $\beta = 0.4$.

Время жизи	Длина
100	95
200	91
300	89
400	90
500	93
600	90
700	91
800	90
900	87
1000	85

Рис. 2: Поиск наименьшей длины в зависимости от времени жизни колонии.

Увеличение времени жизни колонии приводит к лучшему результату.

5 Заключение

В ходе лабораторной работы был реализован муравьиный алгоритм,а также было проведено сравнение работы алгоритма при различных параметрах, задающих веса феромона и временя жизни колонии.