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: **Медведев А.В.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

Постановка задачи

В ходе лабораторной работы предстоит:

1. реализовать муравьиный алгоритм на языке программирования;
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

Для реализации муравьиного алгоритма был выбран язык C#. Граф,описывающий города и дороги между ними, в программе представлен симметричной относительно главной диагонали матрицей смежности размером N*N, где N - количество городов (вершин графа), (i,j)-й элемент которой равен длине пути из i-той вершины в j-тую (весу ребра/дуги).

Листинг 1: Исходный код

```
internal struct Ant
{   public int StartCity;
    public int CurrCity;
    public double Lk;
    public double[] Route;
    public double[] Jk;
};

public struct Answer
{   public double len;
    public double[] route;
}

public class Colony
{
    private readonly Random _rnd;
    private readonly int _n;
    private double[,] _graph;

    public Colony(int n)
    {     _n = n;
        _rnd = new Random();
        create_random_matrix();
    }
    private void copy_array(double[] dst, double[] src)
    {
        for (int i = 0; i < dst.Length; i++)
            dst[i] = src[i];
    }

    private void create_random_matrix()
    {
        _graph = new double[_n, _n];
        for (int i = 0; i < _n; i++)
        {
            for (int j = i; j < _n; j++)
            {
                if (i == j)
                    _graph[i, j] = 0;
                else
                {
                    _graph[i, j] = _rnd.Next(30) + 1;
                    _graph[j, i] = _graph[i, j];
```

```
                }
            }
        }
    }

    private void init_ant(Ant ant)
    {
        int start = ant.StartCity;
        ant.CurrCity = start;
        ant.Lk = 0;
        for (int i = 0; i < ant.Route.Length; i++)
        {
            ant.Route[i] = -1;
            ant.Jk[i] = 1;
        }
        ant.Route[0] = start;
        ant.Jk[start] = 0;
    }

    private Ant[] create_ant_array()
    {
        Ant[] ants = new Ant[_n];
        for (int i = 0; i < _n; i++)
        {
            ants[i] = new Ant
            {
                StartCity = i,
                CurrCity = i,
                Lk = 0,
                Route = new double[_n],
                Jk = new double[_n]
            };
        }
        return ants;
    }
    private void recalc_weight(double[,] weight, double[,]
       pheromon, double[,] visib, double a, double b)
    {
        int n = weight.GetLength(0);
        for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            weight[i, j] = Math.Pow(pheromon[i, j], a) *
               Math.Pow(visib[i, j], b);
```

```
85      }
86
87      private void recalc_pheromon(double[,] pheromon, double
            [,] dPheromon, double p)
88      {
89          p = 1 - p;
90          int n = pheromon.GetLength(0);
91          for (int i = 0; i < n; i++)
92          for (int j = 0; j < n; j++)
93          {
94              pheromon[i, j] = pheromon[i, j] * p + dPheromon
                    [i, j];
95              dPheromon[i, j] = 0;
96          }
97      }
98      private int choose_next(double[] prob)
99      {
100         int i = 0;
101         double rand = _rnd.NextDouble();
102         if (rand == 0)
103         {
104             while (prob[i++] <= 0) ;
105             return --i;
106         }
107
108         while (rand > 0)
109             rand -= prob[i++];
110
111         return --i;
112     }
113
114     private double length_of_route(double[] route)
115     {
116         double length = 0;
117         for (int i = 0; i < route.Length - 1; i++)
118             length += _graph[(int) route[i], (int) route[i
                    + 1]];
119         return length;
120     }
121     private void add_pheromon(double[,] dPheromon, double[]
            route, double lk, int q)
122     {
123         double dFer = q / lk;
```

```
124             for (int i = 0; i < route.Length - 1; i++)
125                 dPheromon[(int) route[i], (int) route[i + 1]]
                        += dFer;
126         }
127         private void gogo_ant(ref Ant ant, double[,] weight,
                double[,] dPheromon, int q)
128         {
129             int i = 1;
130             double[] prob = new double[_n];
131             while (i < _n)
132             {
133                 double sumWeight = 0;
134                 for (int j = 0; j < _n; j++)
135                     sumWeight += weight[ant.CurrCity, j] * ant.
                            Jk[j];
136                 for (int j = 0; j < _n; j++)
137                 {
138                     prob[j] = weight[ant.CurrCity, j] /
                            sumWeight * ant.Jk[j];
139                 }
140 
141                 int next = choose_next(prob);
142                 ant.CurrCity = next;
143                 ant.Jk[next] = 0;
144                 ant.Route[i++] = next;
145             }
146 
147             ant.Lk = length_of_route(ant.Route);
148             add_pheromon(dPheromon, ant.Route, ant.Lk, q);
149         }
150 
151         public Answer Solve(double alfa, double betta, double p
                , int q, int tMax)
152         {
153             Ant[] ants = create_ant_array();
154 
155             double[,] pheromon = new double[_n, _n];
156             for (int i = 0; i < _n; i++)
157             for (int j = 0; j < _n; j++)
158                 pheromon[i, j] = 0.5;
159 
160             double[,] visib = new double[_n, _n];
161             for (int i = 0; i < _n; i++)
```

```
162         {
163             for (int j = i; j < _n; j++)
164             {
165                 if (i != j)
166                 {
167                     visib[i, j] = 1 / _graph[i, j];
168                     visib[j, i] = visib[i, j];
169                 }
170                 else
171                     visib[i, j] = 666;
172             }
173         }
174 
175         double[,] weight = new double[_n, _n];
176         recalc_weight(weight, pheromon, visib, alfa, betta)
                ;
177 
178         double[,] dPheromon = new double[_n, _n];
179         for (int i = 0; i < _n; i++)
180         for (int j = 0; j < _n; j++)
181             dPheromon[i, j] = 0;
182 
183         int bestL = Int32.MaxValue;
184         double[] route = new double[_n];
185 
186         for (int t = 0; t < tMax; t++)
187         {
188             for (int k = 0; k < _n; k++)
189             {
190                 init_ant(ants[k]);
191                 gogo_ant(ref ants[k], weight, pheromon, q);
192             }
193 
194             int best = -1;
195             for (int i = 0; i < _n; i++)
196                 if (ants[i].Lk < bestL)
197                 {
198                     best = i;
199                     bestL = (int) ants[i].Lk;
200                 }
201 
202             if (best != -1)
203                 copy_array(route, ants[best].Route);
```

```
204
205
206                 recalc_pheromon(pheromon, dPheromon, p);
207                 recalc_weight(weight, pheromon, visib, alfa,
                        betta);
208             }
209
210             return new Answer() {len = bestL, route = route};
211         }
```

Методы:

- recalcweight() - Пересчет матрицы весов после каждой итерации
- recalcpheromon() - Обновление значения феромона на дорогах
- choosenext() - выбор пути на основе массивавероятностей
- lengthofroute() - Длинна маршрута
- addpheromon() - Изменение количества феромонов
- lengthofroute() - Длинна маршрута
- lengthofroute() - Длинна маршрута

Главным методом, находящим кратчайший путь, является метод $Solve()$. Колония муравьев ищет путь до источника питания в течение каждого момента времени жизни колонии, которое равно tMax, где одна итерация - один момент времени. В каждый момент времени муравьи начинают движение из случайного города, проходят все города и возвращаются в начальный город, обновляется след феромона на дорогах и обновляется лучшее решение (кратчайший на данный момент путь).

gogoant() - строится маршрут для каждого муравья. Города выбираются с помощью метода choosenext(). Для заполнения массива проб используется формула (1):

$$P_{ij,k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum\limits_{l=J_{i,k}} [\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}} & j \in J_{i,k} \\ 0 & j \notin J_{i,k} \end{cases} \quad (1)$$

choosenext() - для каждого муравья заполняется массив prob, где i-тый элемент - вероятность выбора i-того города следующим к посещению.

В конце каждого похода обновляется значение феромона на дорогах, используется формула (2):

$$\tau_{ij}(t+1) = (1-p) * \tau_{ij}(t) + \Delta\tau_{ij}(t); \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \qquad (2)$$

где

$$\Delta\tau_{ij,k}(t) \quad = \quad \begin{cases} \frac{Q}{L_k(t)}, & (i,j) \in T_k(t) \\ 0, & (i,j) \notin T_k(t) \end{cases} \qquad (3)$$

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

Solve() - для каждого муравья колонии, пройденный им маршрут в данный момент времени сравнивается с маршрутом минимальной на данный момент длины. Если найден маршрут меньшей длины, то данные о лучшем маршруте обновляются.

Эксперимент

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

- количество итераций - 200;
- размерность матрицы - 100;
- α принимает значения от 0 до 1 с шагом 0.1 (при $\alpha = 0$ алгоритм 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	536
0.1	0.9	504
0.2	0.8	488
0.3	0.7	477
0.4	0.6	411
0.5	0.5	419
0.6	0.4	403
0.7	0.3	433
0.8	0.2	459
0.9	0.1	612
1.0	0.0	1088

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

- $\alpha = 0.5$;
- $\beta = 0.5$;
- коэфициент испарения = 0.5;
- размерность матрицы 100*100;
- количество итераций (время жизни колонии) принимает значение от 100 до 1000 с шагом 100.

Полученные данные приведены в Таблице 2.

Таблица 2. Результаты эксперимента 2.

Время жизни колонии	Длина найденного кратчайшего маршрута
100	405
200	416
300	405
400	412
500	412
600	400
700	404
800	367
900	385
1000	390

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 $\alpha = 0.6$ и ($\beta = 0.4$) дает достаточно оптимальный результат.
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).

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.