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

```
1
2  internal struct Ant
3  {
4      public int StartCity;
5      public int CurrCity;
6      public double Lk;
7      public double[] Route;
8      public double[] Jk;
9  };
10
11 public struct Answer
12 {
13     public double Len;
14     public double[] Route;
15 }
16
17 public class Colony
18 {
19
20     private readonly Random _rnd;
21     private readonly int _n;
22     private double[,] _graph;
23
24
25     public Colony(int n)
26     {
27         _n = n;
28         _rnd = new Random();
29
30     }
31
32     private void CopyArray(double[] dst, double[] src)
33     {
34         for (int i = 0; i < dst.Length; i++)
35             dst[i] = src[i];
36     }
37
38
39
40     public void CreateRandomMatrix()
41     {
42         _graph = new double[_n, _n];
43
```

```
44          for (int i = 0; i < _n; i++)
45          {
46              for (int j = i; j < _n; j++)
47              {
48                  if (i == j)
49                      _graph[i, j] = 0;
50                  else
51                  {
52                      _graph[i, j] = _rnd.Next(30) + 1;
53                      _graph[j, i] = _graph[i, j];
54                  }
55              }
56          }
57      }
58
59
60
61
62      private void InitAnt(Ant ant)
63      {
64          int start = ant.StartCity;
65          ant.CurrCity = start;
66          ant.Lk = 0;
67          for (int i = 0; i < ant.Route.Length; i++)
68          {
69              ant.Route[i] = -1;
70              ant.Jk[i] = 1;
71          }
72
73          ant.Route[0] = start;
74          ant.Jk[start] = 0;
75      }
76
77
78      private Ant[] CreateAnts()
79      {
80          Ant[] ants = new Ant[_n];
81          for (int i = 0; i < _n; i++)
82          {
83              ants[i] = new Ant
84              {
85                  StartCity = i,
86                  CurrCity = i,
```

```
87                  Lk = 0,
88                  Route = new double[_n],
89                  Jk = new double[_n]
90              };
91
92          }
93
94          return ants;
95      }
96
97
98      private void RecalcWeight(double[,] weight, double[,]
            pheromon, double[,] visib, double a, double b)
99      {
100
101         for (int i = 0; i < _n; i++)
102         for (int j = 0; j < _n; j++)
103             weight[i, j] = Math.Pow(pheromon[i, j], a) *
                    Math.Pow(visib[i, j], b);
104     }
105
106
107     private void RecalcPheromon(double[,] pheromon, double
            [,] dPheromon, double p)
108     {
109         for (int i = 0; i < _n; i++)
110         for (int j = 0; j < _n; j++)
111         {
112             pheromon[i, j] = pheromon[i, j] * (1 - p) +
                    dPheromon[i, j];
113             dPheromon[i, j] = 0;
114         }
115     }
116
117
118     private int ChooseNext(double[] prob)
119     {
120         int i = 0;
121         double rand = _rnd.NextDouble();
122         if (rand == 0)
123         {
124             while (prob[i++] <= 0) ;
125             return --i;
```

```
126             }
127
128             while (rand > 0)
129                 rand -= prob[i++];
130
131             return --i;
132         }
133
134
135         private double LengthOfRoute(double[] route)
136         {
137             double length = 0;
138             for (int i = 0; i < _n - 1; i++)
139                 length += _graph[(int) route[i], (int) route[i
                        + 1]];
140             return length;
141         }
142
143
144
145         private void AddPheromon(double[,] dPheromon, double[]
                route, double lk, int q)
146         {
147             double dFer = q / lk;
148             for (int i = 0; i < _n - 1; i++)
149                 dPheromon[(int) route[i], (int) route[i + 1]]
                        += dFer;
150         }
151
152         private void GoAnt(ref Ant ant, double[,] weight,
                double[,] dPheromon, int q)
153         {
154
155             int i = 1;
156             double[] prob = new double[_n];
157             while (i < _n)
158             {
159                 double sumWeight = 0;
160                 for (int j = 0; j < _n; j++)
161                     sumWeight += weight[ant.CurrCity, j] * ant.
                            Jk[j];
162                 for (int j = 0; j < _n; j++)
163                 {
```

```
164                 prob[j] = weight[ant.CurrCity, j] /
                        sumWeight * ant.Jk[j];
165             }
166
167             int next = ChooseNext(prob);
168             ant.CurrCity = next;
169             ant.Jk[next] = 0;
170             ant.Route[i++] = next;
171         }
172
173         ant.Lk = LengthOfRoute(ant.Route);
174         AddPheromon(dPheromon, ant.Route, ant.Lk, q);
175     }
176
177
178     public Answer Solve(double alfa, double betta, double p
        , int q, int tMax)
179     {
180
181         //
182         Ant[] ants = CreateAnts();
183
184         //

185         double[,] pheromon = new double[_n, _n];
186         for (int i = 0; i < _n; i++)
187         for (int j = 0; j < _n; j++)
188             pheromon[i, j] = 0.5;
189
190         //"            "
191         double[,] visib = new double[_n, _n];
192         for (int i = 0; i < _n; i++)
193         {
194             for (int j = i; j < _n; j++)
195             {
196                 if (i != j)
197                 {
198                     visib[i, j] = 1 / _graph[i, j];
199                     visib[j, i] = visib[i, j];
200                 }
201                 else
202                     visib[i, j] = 666;
203             }
```

```
204             }
205
206             double[,] weight = new double[_n, _n];
207             RecalcWeight(weight, pheromon, visib, alfa, betta);
208
209             double[,] dPheromon = new double[_n, _n];
210             for (int i = 0; i < _n; i++)
211             for (int j = 0; j < _n; j++)
212                 dPheromon[i, j] = 0;
213
214             int bestL = Int32.MaxValue;
215             double[] route = new double[_n];
216
217             for (int t = 0; t < tMax; t++)
218             {
219                 for (int k = 0; k < _n; k++)
220                 {
221                     InitAnt(ants[k]);
222                     GoAnt(ref ants[k], weight, pheromon, q);
223                 }
224
225                 int best = -1;
226                 for (int i = 0; i < _n; i++)
227                     if (ants[i].Lk < bestL)
228                     {
229                         best = i;
230                         bestL = (int) ants[i].Lk;
231                     }
232
233                 if (best != -1)
234                     CopyArray(route, ants[best].Route);
235
236
237                 RecalcPheromon(pheromon, dPheromon, p);
238                 RecalcWeight(weight, pheromon, visib, alfa,
                        betta);
239             }
240
241             return new Answer() {Len = bestL, Route = route};
242         }
243 }
```

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

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).

GoAnt() - 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):

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

где

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

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 эффектривность 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.