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: **Горохова И.Б.**
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 на языке Java были написаны класс Ant, приведенный в листинге 1, и несколько методов, представленных в листингах ..... 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: Класс Ant

```
private class Ant {
    public int tour[] = new int[graph.length];
    public boolean visited[] = new boolean[graph.length];

    public void visitTown(int town) {
        tour[currInd + 1] = town;
        visited[town] = true;
    }

    public boolean isVisited(int i) {
        return visited[i];
    }

    public int tourLength() {
        int length = graph[tour[townsN - 1]][tour[0]];
        for (int i = 0; i < townsN - 1; i++) {
            length += graph[tour[i]][tour[i + 1]];
        }
        return length;
    }

    public void clear() {
        for (int i = 0; i < townsN; i++)
            visited[i] = false;
    }
}
```

Поля класса Ant:

- tour - массив целых чисел, представляющий собой путь муравья, где каждый элемент массива - номер посещенного города ($graph.length$ - количество городов, $graph$ - матрица весов);
- visited - массив флагов, где i-тый элемент равен $true$, если муравей уже посетил i - тый город и $false$ - если еще не посетил.

Методы класса Ant:

- visitTown() - добавляет в массив с путём муравья номер города $town$ и помечает этот город как посещённый этим муравьем;
- isVisited() - возвращает $true$, если муравей уже был в городе с номером i, и $false$ - иначе;

- tourLength() - суммирует расстрояния из одного города в другой по пути муравья и возвращает значение длины полного пути;
- clear() - отчищает массив с флагами посещенных городов.

В главном классе AntAlgo созданы следующие поля:

Листинг 2: Поля класса AntAlgo

```
public class AntAlgo {
    private double amountOfPheromon = 1;
    private double prefPheromon = 0.5;          //alpha
    private double prefGreedy = 0.5;            //beta
    private double evaporationCoef = 0.5;       //p
    private double newPheromonCoef = 50;        //Q
    private double coefOfAntsCount = 0.8;

    private int townsN = 0;
    private int antsM = 0;

    private int graph[][] = null;
    private double pheromon[][] = null;
    private Ant ants[] = null;
    private double probs[] = null;

    private int currInd = 0;

    public int[] bestTour;
    public int bestTourLen;

    //...
```

- amountOfPheromon - начальное значение феромона;
- prefPheromon - величина, определяющая «стадность» алгоритма;
- prefGreedy - величина, определяющая «жадность» алгоритма;
- evaporationCoef - коэффициент испарения феромона;
- newPheromonCoef - параметр, имеющий значение порядка длины оптимального пути;
- coefOfAntsCount - коэффициент "муравьев на город". (Количество муравьев в колонии = количество городов * coefOfAntsCount);

- townsN - количество городов;
- antsM - количество муравьев (townsN * coefOfAntsCount);
- graph - матрица размерности N*N, где N = townsN, описывающая длины дорог между городами;
- pheromon - матрица, содержащая данные о количестве феромона на дорогах;
- ants - массив муравьев;
- probs - массив, содержащий в i-той ячейке вероятность похода муравья в i-тый город (заполняется для каждого перехода каждого муравья заново);
- currInd - количество уже посещенных городов, принимает значения от 0 до townsN-1;
- bestTour - массив, содержащий порядок лучшего на данный момент пути по длине;
- bestTourLen - длина пути из bestTour.

Главным методом, находящим кратчайший путь, является метод *solve*(). Колония муравьев ищет путь до источника питания в течение каждого момента времени жизни колонии, которое равно maxIterations, где одна итерация - один момент времени. В каждый момент времени муравьи начинают движение из рандомного города, проходят все города и возвращаются в начальный город, обновляется след феромона на дорогах и обновляется лучшее решение (кратчайший на данный момент путь). Метод представлен в листинге 3, а в листинге 4 представлены методы, вызываемые из solve().

Листинг 3: Метод solve()

```
public void solve() {
    for (int i = 0; i < townsN; i++)
        for (int j = 0; j < townsN; j++)
            pheromon[i][j] = amountOfPheromon;

    int iteration = 0;
    int maxIterations = 50;
    while (iteration < maxIterations) {
        startAnts();
```

```
10          moveAnts();
11          updatePheromone();
12          updateBestTour();
13          iteration++;
14      }
15      System.out.println("Best tour length: " + bestTourLen +
            ". Tour: " + tourToString(bestTour));
16  }
```

Листинг 4: Методы класса AntAlgo

```
1   private void startAnts() {
2       currInd = -1;
3       for (Ant a: ants) {
4           a.clear();
5           a.visitTown(rand.nextInt(townsN));
6       }
7       currInd++;
8   }
9
10  private void moveAnts() {
11      while (currInd < townsN -1) {
12          for (Ant a: ants)
13              a.visitTown(selectNextTown(a));
14          currInd++;
15      }
16  }
17
18  private int selectNextTown(Ant ant) {
19      getProbability(ant);
20
21      double r = rand.nextDouble();
22      double total = 0;
23      for (int i = 0; i < townsN; i++) {
24          total += probs[i];
25          if (total >= r)
26              return i;
27      }
28  }
29
30  private void getProbability(Ant ant) {
31      int ind = ant.tour[currInd];
32
33      double znam = 0.0;
```

```
34      for (int i = 0; i < townsN; i++)
35          if (!ant.isVisited(i))
36              znam += Math.pow(pheromon[ind][i], prefPheromon
                    ) * Math.pow(1.0/graph[ind][i], prefGreedy);
37
38      for (int i = 0; i < townsN; i++) {
39          if (ant.isVisited(i))
40              probs[i] = 0.0;
41          else {
42              double chisl = Math.pow(pheromon[ind][i],
                    prefPheromon) * Math.pow(1.0/graph[ind][i],
                    prefGreedy);
43              probs[i] = chisl/znam;
44          }
45      }
46  }
47
48  private void updatePheromone() {
49      for (int i = 0; i < townsN; i++)
50          for (int j = 0; j < townsN; j++)
51              pheromon[i][j] *= (1 - evaporationCoef);
52
53      for (Ant a: ants){
54          double antContribution = newPheromonCoef/a.
                tourLength();
55          pheromon[a.tour[townsN-1]][a.tour[0]] +=
                antContribution;
56          for (int i = 0; i < townsN-1; i++) {
57              pheromon[a.tour[i]][a.tour[i+1]] +=
                    antContribution;
58          }
59      }
60  }
61
62  private void updateBestTour() {
63      if (bestTour == null) {
64          bestTour = ants[0].tour;
65          bestTourLen = ants[0].tourLength();
66      }
67      for (Ant a: ants) {
68          if (a.tourLength() < bestTourLen) {
69              System.out.println("Old: " + bestTourLen + "
                    New: " + a.tourLength());
```

```
70                  bestTour = a.tour.clone();
71                  bestTourLen = a.tourLength();
72              }
73          }
74  }
```

startAnts() - для каждого муравья выбирается рандомный город как начальный.

moveAnts() - строится маршрут для каждого муравья. Города выбираются с помощью метода selectNextTown().

selectNextTown() - для каждого муравья заполняется массив prob, где i-тый элемент - вероятность выбора i-того города следующим к посещению. Получают рандомное число r от 0 до 1. Вероятности определяют ширину зоны для каждого из городов. город, в зону которого попадает r, и выбирается следующим для похода.
Пример: Пусть есть три города, которые муравей может посетить. С вероятностью 0.35 он может пойти в 1ый город, с 0.5 - во второй, с 0.15 - в третий. Зоны распределяются таким образом:[0-0.35) - 1ый город, [0.35-0.85) - 2ой город, [0.85-1] - 3ий город . Пусть рандомное число r = 0.45. Оно попадает в зону второго города, следовательно для посещения муравья выбирается второй город.

getProbability() - заполняется массив probs, используемый в функции selectNextTown(). Для заполнения массива используется формула (1):

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

где α, β в программе заданы, как prefPheromon и prefGreedy соответственно (величина, определяющая «стадность» алгоритма и «жадность» алгоритма); τ - feromon[i][j], а η - $\frac{1}{graph[i][j]}$ (матрица размерности, описывающая длины дорог между городами и матрица, содержащая данные о количестве феромона на дорогах)

updatePheromone() - 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; m - количество муравьев в колонии.

updateBestTour() - 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.

Так же для решения задачи использовались функции graphFilling() - заполнение матрицы смежностей данными, printMatrix() - печать матрицы смежностей в консоли, tourToString() - функция печати последовательности номеров городов в консоль.

Функция Main() выглядит следующим образом:

Листинг 5: Точка входа в программу. Функция Main()

```
public static void main(String[] args) throws IOException {
    AntAlgo algorithm = new AntAlgo();
    algorithm.graphFilling(true, 100);

    antsM = (int)(townsN*coefOfAntsCount);
    pheromon = new double[townsN][townsN];
    ants = new Ant[antsM];
    for (int i = 0; i < antsM; i++)
        ants[i] = new Ant();
    probs = new double[townsN];

    algorithm.printMatrix(true);
    algorithm.solve();
}
```

Эксперимент

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

- количество итераций maxIterations = 100;
- размерность матрицы townsN = 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 эффективность работы реализованного алгоритма в зависимости от количества итераций (времени жизни колонии муравьев).
Входные данные:

- $\alpha = 0.5$;
- $\beta = 0.5$;
- размерность матрицы townsN = 50;
- количество итераций maxIterations (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	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).

Заключение

В ходе лабораторной работы я реализовала муравьиный алгоритм на языке программирования Java и провела сравнение работы алгоритма при разных значениях параметров, задающих веса феромона и времени жизни колонны.