Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе № 6

Название: Муравьиный алгоритм

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) М.А. Козлов (И.О. Фамилия)

Преподователь _____ (Подпись, дата) Л.Л. Волкова (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Муравьиный алгоритм – один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Целью данной лабораторной работы является изучение муравьиных алгоритмов и приобретение навыков параметризации методов на примере муравьиного алгоритма, примененного к задаче коммивояжера.

Задачи данной лабораторной работы:

1) рассмотренть муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера;
2) реализовать эти алгоритмы;
3) сравнить время работы этих алгоритмов.

1 Аналитический раздел

В данной части будут рассмотрены теоретические основы задачи коммивояжера и муравьиного алгоритма.

1.1 Постановка задачи коммивояжера

Коммивояжёр (фр. commis voyageur) – бродячий торговец. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, необходимо объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры выгодности маршрута (точнее говоря, невыгодности) может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута.

Задача коммивояжёра – важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. В терминах теории графа она формулируется как задача поиска минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе с n вершинами. Где вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояния или стоимости проезда [1].

1.2 Метод полного перебора

Задача коммивояжёра может быть решена перебором всех вариантов объезда и выбором оптимального. Но точный переборный алгоритм её решения имеет факториальную сложность, так как количество возможных маршрутов очень быстро возрастает с ростом n. Оно равно n! — количеству способов упорядочения пунктов. К примеру, для 100 пунктов количество вариантов будет представляться 158-значным числом. Мощная ЭВМ, способная перебирать миллион вариантов в секунду, будет биться с задачей на протяжении примерно 3*10144 лет. Не спасает ситуацию даже то, что для каждого варианта маршрута имеется 2n равноценных, отличающихся выбором начального пункта (n вариантов) и направлением обхода (2 варианта). Перебор с учётом этого сокращается незначительно. Для проверки оптимальности маршрута необходимо проверить его со всеми имеющимися, что невозможно за полиномиальное время, поэтому задача коммивояжёра является NP-трудной.

1.3 Муравьиный алгоритм

Все муравьиные алгоритмы базируются на моделировании поведения колонии муравьев. Колония муравьев может рассматриваться как многоагентная система, в которой каждый агент (муравей) функционирует автономнопо очень простым правилам. В противовес почти примитивному поведению агентов, поведение всей системы получается на удивление разумным.

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений.

Идея муравьиного алгоритма – моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь. При своём движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьёв находить новый путь, если старый оказывается недоступным.

Какие же механизмы обеспечивают столь сложное поведение муравьев, и что можем мы позаимствовать у этих крошечных существ для решения своих глобальных задач? Основу «социального» поведения муравьев составляет самоорганизация – множество динамических механизмов, обеспечивающих достижение системой глобальной цели в результате низкоуровневого взаимодействия ее элементов. Принципиальной особенностью такого взаимодействия является использование элементами системы только локальной информации. При этом исключается любое централизованное управление и обращениек глобальному образу, репрезентирующему систему во внешнем мире. Самоорганизация является результатом взаимодействия следующих четырех компонентов:

1) случайность;
2) многократность;
3) положительная обратная связь;
4) отрицательная обратная связь.

Рассмотрим случай, когда на оптимальном доселе пути возникает преграда. В этом случае необходимо определение нового оптимального пути. Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако, те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащён феромоном. Поскольку движение муравьёв определяется концентрацией феромона, то следующие будут предпочитать именно этот путь, продолжая обогащать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Очевидная положительная обратная связь быстро приведёт к тому, что кратчайший путь станет единственным маршрутом движения большинства муравьёв. Моделирование испарения феромона – отрицательной обратной связи – гарантирует, что найденное локально оптимальное решение не будет единственным – муравьи будут искать и другие пути. Если мы моделируем процесс такого поведения на некотором графе, рёбра которого представляют собой возможные пути перемещения муравьёв, в течение определённого времени, то наиболее обогащённый феромоном путь по рёбрам этого графа и будет являться решением задачи, полученным с помощью муравьиного алгоритма.

Обобщим все выше сказанное. Любой муравьиный алгоритм, независимо от модификаций, представим в следующем виде:

1) создание муравьев;
2) поиск решения;
3) обновление феромона;
4) дополнительные действия (опционально).

Рассмотрим каждый шаг в цикле более подробно.

1.3.1 Создание муравьев

Стартовая точка, куда помещается муравей, зависит ограничений, накладываемых условиями задачи. Потому что для каждой задачи способ размещения муравьёв является определяющим. Либо все они помещаются в одну точку, либо в разные с повторения, либо без повторений.

На этом же этапе задается начальный уровень феромона. Он инициализируется небольшим положительным числом для того, чтобы на начальном шаге вероятности перехода в следующую вершину не были нулевыми.

1.3.2 Поиск решения

Вероятность перехода из вершины i в вершину j определяется по формуле (1.1):

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}, \tag{1.1}$$

где $\tau_{i,j}$ – расстояние от города i до j;
$\eta_{i,j}$ – количество феромонов на ребре ij;
α – параметр влияния длины пути;
β – параметр влияния феромона.

1.3.3 Обновление феромона

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах след, обратно пропорциональный длине пройденного пути. Новый след феромона вычисляется по формуле (1.2):

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}, \tag{1.2}$$

где $\rho_{i,j}$ –доля феромона, который испарится;
$\tau_{i,j}$ – количество феромона на дуге ij;
$\Delta\tau_{i,j}$ – количество отложенного феромона.

1.3.4 Дополнительные действия

Обычно здесь используется алгоритм локального поиска, однако он может также появиться и после поиска всех решений.

1.4 Муравьиный алгоритм в задаче коммивояжера

Рассмотрим, как реализовать четыре составляющие самоорганизации муравьев при оптимизации маршрута коммивояжера. Многократность взаимодействия реализуется итерационным поиском маршрута коммивояжера одновременно несколькими муравьями. При этом каждый муравей рассматривается как отдельный, независимый коммивояжер, решающий свою задачу. За одну итерацию алгоритма каждый муравей совершаетполный маршрут коммивояжера. Положительная обратная связь реализуется как имитация поведения муравьев типа «оставление следов – перемещение по следам». Чем больше следов оставлено на тропе – ребре графа в задаче коммивояжера, тем больше муравьев будет передвигаться по ней. При этом на тропе появляются новые следы, привлекающие дополнительных муравьев. Для задачи коммивояжера положительная обратная связь реализуется следующим стохастическим правилом: вероятность включения ребра графа в маршрут муравья пропорциональна количеству феромона на нем.

С учетом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьев при выборе пути.

Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов – список запретов. Обозначим через J 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.

Муравьи обладают «обонянием» – они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i,j) в момент времени t обозначим через $tau_{i,j}(t)$. На этом основании мы можем сформулировать вероятностнопропорциональное правило, определяющее вероятность перехода k-ого муравья из города i в город j.

Пройдя ребро (i,j), муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q – параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано формулой (1.3):

$$\Delta\tau_{i,j}^{k} = \begin{cases} Q/L_k & \text{если k-ый муравей прошел по ребру ij;} \\ 0 & \text{иначе,} \end{cases} \tag{1.3}$$

где Q – количество феромона, переносимого муравьем.

С учётом этого приходим к окончательному варианту формулы (1.4):

$$\Delta\tau_{i,j} = \tau_{i,j}^{0} + \tau_{i,j}^{1} + ... + \tau_{i,j}^{k}, \tag{1.4}$$

где k – количество муравьев в вершине графа с индексами i и j.

2 Констукторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и опредены способы тестирования.

2.1 Разработка алгоритмов

Ниже будут представлены схемы алгоритмов:

1) алгоритм полного перебора (рисунок 2.1);
2) муравьиный алгоритм (рисунок 2.2).

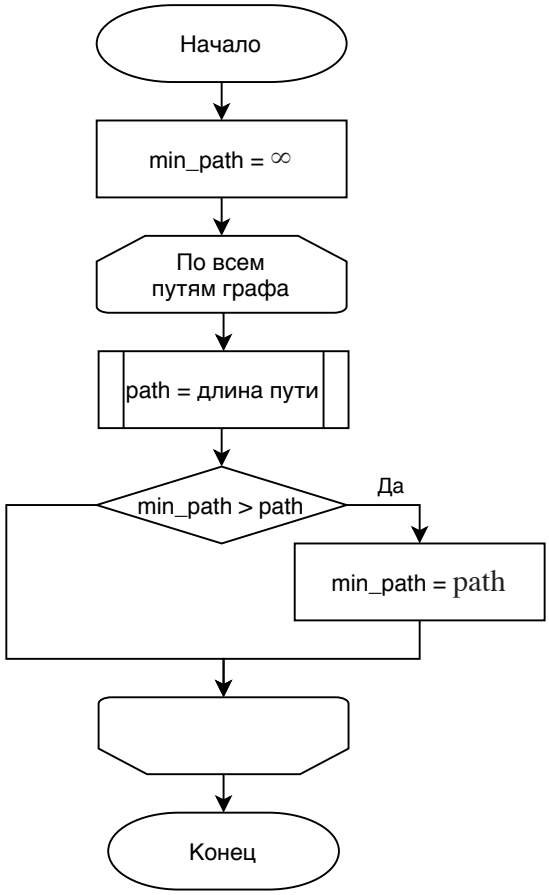

Рисунок 2.1 — Схема алгоритма полного перебора

Рисунок 2.2 — Схема муравьиного алгоритма

2.2 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность консольного приложения:

1) ;
2) .

2.3 Тестирование

Тестирование ПО будет проводиться методом чёрного ящика. Необходимо проверить работу системы на случаях, когда граф состоит из 2х и более вершин.

3 Технологический раздел

В данном разделе будут выбраны средства реализации ПО и представлен листинг кода.

3.1 Средства реализации

В данной работе используется язык программирования C# [2], так как он позволяет написать программу в относительно малый срок. В качестве среды разработки использовалась Visual Studio Code [3].

3.2 Листинг программы

Ниже представлены листинги кода алгоритмов:

1) полного перебора (листинг 3.1);
2) муравьиного (листинг 3.2).

Листинг 3.1 — Реализация алгоритма поиска полным перебором

```
1  public class BruteForceAlgorithm : IRouteAlgorithm
2  {
3      public Path GetRoute(Map map)
4      {
5          Path shortest = new Path(null, map);
6          foreach (var cur in GetAllRoutes(map.N))
7          {
8              cur.Add(0);
9              Path check = new Path(cur, map);
10             if (shortest.Distance > check.Distance)
11                 shortest = check;
12         }
13         return shortest;
14     }
15
16     private IEnumerable<List<int>> GetAllRoutes(int count)
17     {
18         List<int> cur = new List<int>();
19         for (int i = 0; i < count; i++)
20             cur.Add(i);
21
22         while (NextSet(cur, count))
23             yield return new List<int>(cur);
24     }
25
26     private bool NextSet(List<int> cur, int count)
27     {
28         int j = count - 2;
29         while (j != -1 && cur[j] >= cur[j + 1])
```

```
            j--;

        if (j == -1 || cur[j] == 0)
            return false;

        int k = count - 1;
        while (cur[j] >= cur[k])
            k--;
        Swap(cur, j, k);
        int left = j + 1;
        int right = count - 1;

        while (left < right)
            Swap(cur, left++, right--);
        return true;
    }

    private void Swap(List<int> cur, int i, int j)
    {
        int tmp = cur[i];
        cur[i] = cur[j];
        cur[j] = tmp;
    }
}
```

Листинг 3.2 — Реализация муравьиного алгоритма

```
public class AntsAlgorithm : IRouteAlgorithm
{
    public AntsAlgorithm(int maxTime, double alpha, double beta, double q, double
        pho)
    {
        MaxTime = maxTime;
        Alpha = alpha;
        Beta = beta;
        Q = q;
        Pho = pho;
    }

    public int MaxTime { get; set; }
    public double Alpha { get; set; }
    public double Beta { get; set; }
    public double Q { get; set; }
    public double Pho { get; set; }


    public Path GetRoute(Map map)
```

```
20      {
21          Random r = new Random();
22          Path shortest = new Path(null, map);
23
24          int count = map.N;
25          double[,] pher = InitPheromone(0.1, count);
26
27          for (int time = 0; time < MaxTime; time++)
28          {
29              var ants = InitAnts(map);
30              for (int i = 0; i < count - 1; i++)
31              {
32                  double[,] deltaPher = InitPheromone(0, count);
33                  foreach (var ant in ants)
34                  {
35                      int curTown = ant.LastVisited();
36
37                      double sum = 0;
38                      for (int town = 0; town < count; town++)
39                      {
40                          if (!ant.IsVisited(town))
41                          {
42                              double tau = pher[curTown, town];
43                              double eta = 1 / map[curTown, town];
44                              sum += Math.Pow(tau, Alpha) * Math.Pow(eta, Beta);
45                          }
46                      }
47
48                      double check = r.NextDouble();
49                      int newTown = 0;
50                      for (; check > 0; newTown++)
51                      {
52                          if (!ant.IsVisited(newTown))
53                          {
54                              double tau = pher[curTown, newTown];
55                              double eta = 1 / map[curTown, newTown];
56                              double chance = Math.Pow(tau, Alpha) * Math.Pow(eta,
                                    Beta) / sum;
57                              check -= chance;
58                          }
59                      }
60                      newTown--;
61                      ant.VisitTown(newTown);
62                      deltaPher[curTown, newTown] += Q / map[curTown, newTown];
63                  }
64                  for (int k = 0; k < count; k++)
65                      for (int t = 0; t < count; t++)
```

```
66                          pher[k, t] = (1 - Pho) * pher[k, t] + deltaPher[k, t];
67              }
68              foreach (var ant in ants)
69              {
70                  ant.VisitTown(ant.Start);
71                  if (ant.GetDistance() < shortest.Distance)
72                      shortest = ant.Path;
73              }
74          }
75
76          return shortest;
77      }
78
79      private List<Ant> InitAnts(Map map)
80      {
81          var ants = new List<Ant>(map.N);
82          for (int i = 0; i < map.N; i++)
83              ants.Add(new Ant(map, i));
84          return ants;
85      }
86
87      private double[,] InitPheromone(double num, int size)
88      {
89          double[,] phen = new double[size, size];
90          for (int i = 0; i < size; i++)
91              for (int j = 0; j < size; j++)
92                  phen[i, j] = num;
93          return phen;
94      }
95  }
```

3.3 Тестирование

В таблице **??** отображён возможный набор тестов для тестирования методом чёрного ящика, результаты которого, представленные на рисунке 3.1, подтверждают прохождение программы перечисленных тестов.

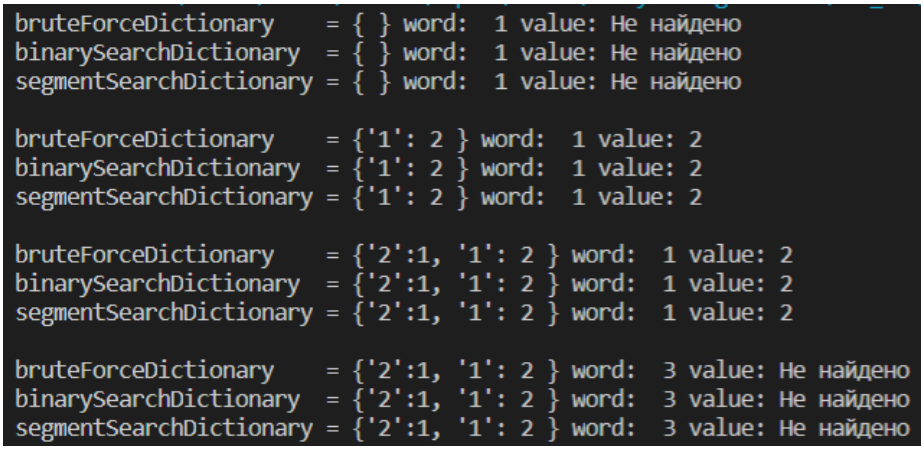

Рисунок 3.1 — Результаты тестирования алгоритмов.

4 Экспериментальный раздел

В данном разделе будут проведен сравнительный анализ алгоритмов по затрачиваемому процессорному времени и рассмотрена параметризация муравьиного алгоритма. Тестирование проводилось на ноутбуке с процессором Intel(R) Core(TM) i5-7200U CPU 2.50 GHz [4] под управлением Windows 10 с 8 Гб оперативной памяти.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данного проекта были проведёны эксперименты по замеру времени работы алгоритмов:

1) на малых размерностях (график **??**);
2) на больших размерностях (график **??**).

4.2 Параметризация муравьиного алгоритма

Для различных значение параметров

4.3 Вывод

В ходе экспериментов по замеру времени работы было установлено, что

Заключение

В ходе выполнения данной лабораторной работы были изучены

Список использованных источников

1. *Ульянов, М.В.* Рекурсивно-эффективные компьютерные алгоритмы. Разработка и анализ / М.В. Ульянов. — Наука Физматлит, 2007.

2. Python. // [Электронный ресурс]. Режим доступа: https://www.python.org/, (дата обращения: 01.10.2020).

3. Visual Studio Code - Code Editing. // [Электронный ресурс]. Режим доступа: https://code.visualstudio.com, (дата обращения: 01.10.2020).

4. Intel® Core™ i5-7200U Processor. // [Электронный ресурс]. Режим доступа: https://www.intel.com/content/www/us/en/products/processors/core/i5-processors/i5-7200u.html, (дата обращения: 26.09.2020).