МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Звіт

З лабораторної роботи №1

На тему «Алгоритм мурахи»

З дисципліни «Основи проектування систем штучного інтелекту та розпізнавання образів»

Підготував: Коваль Ігор Леонідович

Студент навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки

група КН-51м

Рівне 2022

Мета роботи: реалізувати задачу комівояжера з використанням алгоритму мурахи.

Теоретичні відомості

Алгоритми мурахи (Ant algorithms), або оптимізація за принципом мурашиної колонії (ця назва була придумана винахідником алгоритму, Марко Доріго (Marco Dorigo)) мають специфічні властивості, що властиві мурахам, і використовують їх для орієнтації у фізичному просторі. Природа пропонує різні методики для вирішення різних задач (як буде показано в інших лабораторних роботах, наприклад, генетичний алгоритм). Алгоритми мурахи особливо цікаві тому, що їх можна використати для вирішення не лише статичних, але і динамічних проблем, наприклад, проблем маршрутизації в мережах, що змінюються.

Хоча мурахи і сліпі, вони уміють орієнтуватися на складній місцевості, знаходити їжу на великій відстані від мурашника і успішно повертатися додому. Виділяючи ферменти під час переміщення, мурахи змінюють довкілля, забезпечують комунікацію, а також знаходять зворотний шлях в мурашник.

Найдивовижніше в цьому процесі - це те, що мурахи уміють знаходити найоптимальніший шлях між мурашником і зовнішніми точками. Чим більше мурашок використовують один і той же шлях, тим вище концентрація ферментів на цьому шляху. Чим ближче зовнішня точка до мурашника, тим більше разів до неї переміщалися мурахи. Що стосується більш віддаленої точки, то її мурахи досягають рідше, тому по дорозі до неї вони застосовують сильніші ферменти. Чим вище концентрація ферментів на шляху, тим прийнятніше він для мурах в порівнянні з іншими доступними. Так мурашина «логіка» дозволяє вибирати коротший шлях між кінцевими точками.

Алгоритм мурахи

Мураха - це програмний агент, який є членом великої колонії і використовується для вирішення певної проблеми. Нехай ми розглянемо проблему задачі комівояжера. Мураха забезпечується набором простих

правил, які дозволяють їй вибирати дорогу в графі. Вона підтримує список табу (tabu list), тобто список вузлів, які вона вже відвідала. Таким чином, мураха повинна проходити через кожен вузол лише один раз. Дорога між двома вузлами графа, по якому мураха відвідала кожен вузол лише один раз, називається шляхом Гамільтона (Hamiltonian path),названа на часть математики сера Уїльяма Гамільтона (Sir William Hamilton).

Вузли в списку - «поточної подорожі» розташовуються в тому порядку, в якому мураха відвідувала їх. Пізніше список використовується для визначення протяжності дороги між вузлами.

Справжня мураха під час переміщення по дорозі залишатиме за собою фермент. У алгоритмі мурахи агент залишає фермент на гранях мережі після завершення подорожі. Про те, як це відбувається, розповідається в розділі «Подорож мурахи».

Початкова популяція

Після створення популяція мурах порівну розподіляється по вузлах мережі. Необхідне рівне розділення мурах між вузлами, щоб всі вузли мали однакові шанси стати відправною точкою. Якщо всі мурахи почнуть рух з однієї точки, це означатиме, що дана точка є оптимальною для старту, а насправді ми цього не знаємо.

Рух мурахи

Рух мурахи грунтується на одному і дуже простому імовірнісному рівнянні. Якщо мураха ще не закінчила дорогу (path), тобто не відвідала всі

вузли мережі, для визначення наступної грані дороги використовується рівняння (1)

$$P = \frac{\tau(r,u)^{\alpha} * \eta(r,u)^{\beta}}{\sum_k \tau(r,u)^{\alpha} * \eta(r,u)^{\beta}} \quad (1)$$

Тут $\tau(r,u)$ - інтенсивність ферменту на грані між вузлами r і u,

η (r,u) - функція, яка представляє вимір зворотної відстані для грані (рівняння (2)),

α - вага ферменту, β - коефіцієнт евристики

$$\eta(r,u) = \frac{1}{len(r,u)} \quad (2)$$

Параметри α і β визначають відносну значимість двох параметрів, а також їх вплив на рівняння. Памятаймо, що мураха подорожує лише по вузлах, які ще не були відвідані (як вказано списком табу). Тому ймовірність наступного вибору розраховується лише для граней, які ведуть до ще не відвіданих вузлів.

Подорож мурахи

Пройдений мурахою шлях відображується, коли мураха від- відає всі вузли діаграми. Зверніть увагу, що цикли заборонені, оскільки в алгоритм включений список табу. Після завершення довжина дороги може бути підрахована - вона рівна сумі всіх граней, по яких подорожувала мураха. Рівняння (3) показує кі- лькість ферменту, який був залишений на кожній грані дороги для мурахи k. Змінна Q є константою.

$$\Delta\tau_{ij}^{k}(t) = \frac{Q}{L^{k}(t)} \quad (3)$$

Результат рівняння є засобом виміру дороги - коротка дорога характеризується високою концентрацією ферменту, а довша дорога - нижчою. Потім отриманий результат використовується в рівнянні (4), щоб збільшити кількість ферменту уздовж кожної грані пройденого мурахою шляху.

$$\tau_{ij}(t) = (\tau_{ij}(t) + \Delta\tau_{ij}^{k}(t)) * \rho \quad (4)$$

Дане рівняння застосовується до всієї дороги, при цьому кожна грань позначається ферментом пропорційно довжині дороги. Тому слід дочекатися, поки мураха закінчить подорож і тільки потім оновити значення ферменту, інакше дійсна довжина дороги залишиться невідомою. Константа ρ - значення між 0 і 1.

Випаровування ферменту На початку шляху в кожної грані є шанс бути вибраною. Щоб поступово видалити грані, які входять в гірші шляхи в мережі, до всіх граней застосовується процедура випаровування ферменту (Pheromoiie evaporation). Використовуючи константу ρ з рівняння (4), ми отримуємо рівняння (5).

Тому для випаровування ферменту використовується зворотний

$$\tau_{ij}(t) = \tau_{ij}(t) * (1 - \rho) \qquad (5)$$

коефіцієнт оновлення дороги.

Повторний запуск

Після того, як шлях мурахи завершений, грані оновлені відповідно до довжини шляху і сталося випаровування ферменту на всіх гранях, алгоритм запускається повторно. Список табу очищається і довжина шляху стає рівною нулю. Мурахам дозволя-ється рухатися по мережі згідно рівняння (1). Цей процес може виконуватися до тих пір, коли впродовж декількох запусків не було відмічено повторних змін. Потім визначається кращий шлях, який і є рішенням.

Результат роботи програми

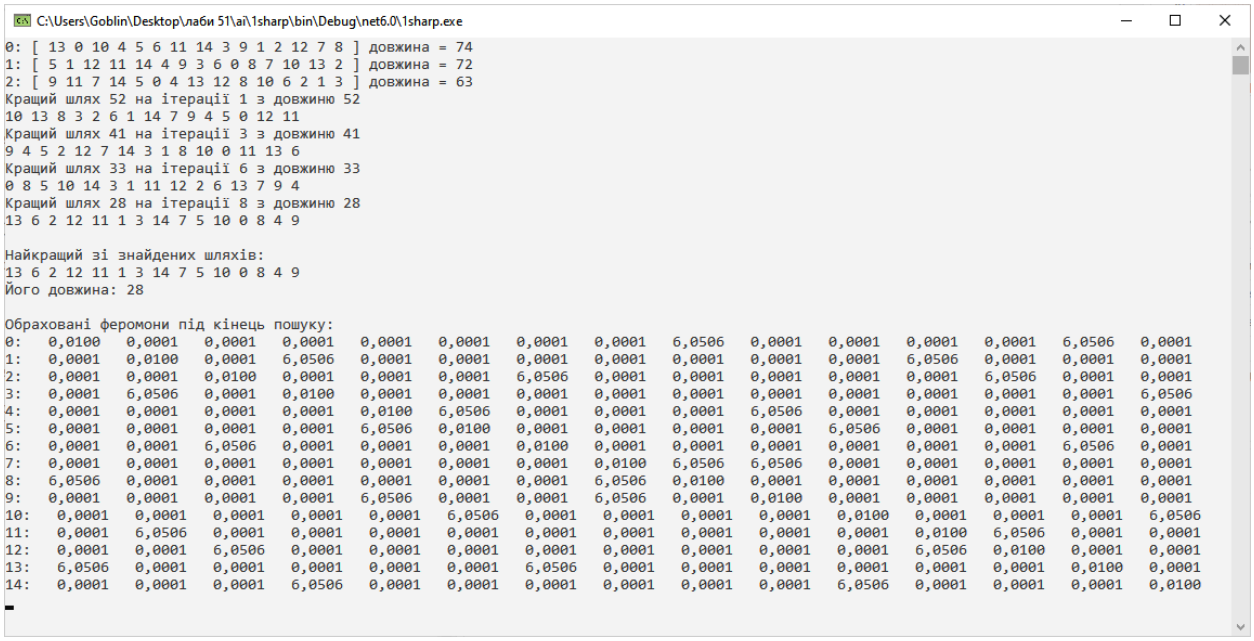

Рис. 1. Результат пошуку оптимального шляху (10 міст, 3 мурахи)

```
0: [ 13 19 12 16 9 11 1 18 6 0 4 8 5 10 17 7 15 3 2 14 ] довжина = 97
1: [ 13 4 10 12 8 9 5 17 6 15 1 16 14 2 3 19 18 0 7 11 ] довжина = 83
Кращий шлях 69 на ітерації 1 з довжиною 69
16 8 18 5 3 2 13 4 11 10 15 9 0 12 1 19 14 17 7 6
Кращий шлях 61 на ітерації 10 з довжиною 61
10 4 18 8 19 1 12 16 7 0 5 6 3 14 15 11 2 9 17 13
Кращий шлях 47 на ітерації 13 з довжиною 47
5 14 3 10 4 6 7 16 0 2 12 1 19 18 8 15 17 9 13 11
Кращий шлях 37 на ітерації 16 з довжиною 37
3 14 15 5 12 1 6 17 9 13 11 7 16 0 10 4 2 18 8 19
Кращий шлях 36 на ітерації 18 з довжиною 36
1 12 4 10 17 6 7 16 0 5 15 14 3 2 9 13 11 19 18 8
Кращий шлях 32 на ітерації 22 з довжиною 32
9 17 6 7 16 0 5 15 14 3 2 8 18 19 1 12 11 13 10 4
Кращий шлях 27 на ітерації 23 з довжиною 27
10 4 2 8 18 19 1 12 11 13 9 17 6 7 16 0 5 15 14 3
Кращий шлях 26 на ітерації 34 з довжиною 26
5 0 16 7 6 17 9 13 11 12 1 19 18 8 2 4 10 15 14 3

Найкращий зі знайдених шляхів:
5 0 16 7 6 17 9 13 11 12 1 19 18 8 2 4 10 15 14 3
Його довжина: 26
```

Рис. 2. Результат пошуку оптимального шляху (20 міст, 2 мурахи)

Код програми

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;

namespace Ants
{
  internal class Ants
  {
    private static Random random = new Random(0);
    private static int alpha = 3;
    private static int beta = 2;
    private static double rho = 0.01;
    private static double Q = 2.0;

    public static void Main(string[] args)
    {
      int numCities = 20;
      int numAnts = 2;
      int maxIter = 1000;

      int[][] dists = MakeGraphDistances(numCities);
      int[][] ants = InitAnts(numAnts, numCities);
      ShowAnts(ants, dists);

      int[] bestTrail = BestTrail(ants, dists);
      double bestLength = Length(bestTrail, dists);

      double[][] pheromones = InitPheromones(numCities);

      int cx = 0;
      while (cx < maxIter)
      {
        UpdateAnts(ants, pheromones, dists);
        UpdatePheromones(pheromones, ants, dists);

        int[] currBestTrail = BestTrail(ants, dists);
        double currBestLength = Length(currBestTrail, dists);
        if (currBestLength < bestLength)
        {
          bestLength = currBestLength;
          bestTrail = currBestTrail;
          Console.WriteLine("Кращий шлях " + bestLength.ToString() + " на ітерації " + cx + " з довжиню " +
currBestLength.ToString());
          Display(currBestTrail);
        }
        cx += 1;
```

```
    }

    Console.WriteLine("\nНайкращий зі знайдених шляхів:");
    Display(bestTrail);
    Console.WriteLine("Його довжина: " + bestLength.ToString());

    Console.WriteLine("\nОбраховані феромони під кінець пошуку:");
    Display(pheromones);

    Console.ReadLine();
}

private static int[][] InitAnts(int numAnts, int numCities)
{
    int[][] ants = new int[numAnts][];
    for (int k = 0; k <= numAnts - 1; k++)
    {
        int start = random.Next(0, numCities);
        ants[k] = RandomTrail(start, numCities);
    }
    return ants;
}

private static int[] RandomTrail(int start, int numCities)
{
    int[] trail = new int[numCities];

    for (int i = 0; i <= numCities - 1; i++)
    {
        trail[i] = i;
    }

    for (int i = 0; i <= numCities - 1; i++)
    {
        int r = random.Next(i, numCities);
        int tmp = trail[r];
        trail[r] = trail[i];
        trail[i] = tmp;
    }

    int idx = IndexOfTarget(trail, start);
    int temp = trail[0];
    trail[0] = trail[idx];
    trail[idx] = temp;

    return trail;
}

private static int IndexOfTarget(int[] trail, int target)
```

```
{
  for (int i = 0; i <= trail.Length - 1; i++)
  {
    if (trail[i] == target)
    {
      return i;
    }
  }
  return -1;
}

private static double Length(int[] trail, int[][] dists)
{
  double result = 0.0;
  for (int i = 0; i <= trail.Length - 2; i++)
  {
    result += Distance(trail[i], trail[i + 1], dists);
  }
  return result;
}

private static int[] BestTrail(int[][] ants, int[][] dists)
{
  double bestLength = Length(ants[0], dists);
  int idxBestLength = 0;
  for (int k = 1; k <= ants.Length - 1; k++)
  {
    double len = Length(ants[k], dists);
    if (len < bestLength)
    {
      bestLength = len;
      idxBestLength = k;
    }
  }
  int numCities = ants[0].Length;
  int[] bestTrail_Renamed = new int[numCities];
  ants[idxBestLength].CopyTo(bestTrail_Renamed, 0);
  return bestTrail_Renamed;
}

private static double[][] InitPheromones(int numCities)
{
  double[][] pheromones = new double[numCities][];
  for (int i = 0; i <= numCities - 1; i++)
  {
    pheromones[i] = new double[numCities];
  }
  for (int i = 0; i <= pheromones.Length - 1; i++)
  {
```

```
      for (int j = 0; j <= pheromones[i].Length - 1; j++)
      {
        pheromones[i][j] = 0.01;
      }
    }
    return pheromones;
}

private static void UpdateAnts(int[][] ants, double[][] pheromones, int[][] dists)
{
    int numCities = pheromones.Length;
    for (int k = 0; k <= ants.Length - 1; k++)
    {
      int start = random.Next(0, numCities);
      int[] newTrail = BuildTrail(k, start, pheromones, dists);
      ants[k] = newTrail;
    }
}

private static int[] BuildTrail(int k, int start, double[][] pheromones, int[][] dists)
{
    int numCities = pheromones.Length;
    int[] trail = new int[numCities];
    bool[] visited = new bool[numCities];
    trail[0] = start;
    visited[start] = true;
    for (int i = 0; i <= numCities - 2; i++)
    {
      int cityX = trail[i];
      int next = NextCity(k, cityX, visited, pheromones, dists);
      trail[i + 1] = next;
      visited[next] = true;
    }
    return trail;
}

private static int NextCity(int k, int cityX, bool[] visited, double[][] pheromones, int[][] dists)
{
    double[] probs = MoveProbs(k, cityX, visited, pheromones, dists);

    double[] cumul = new double[probs.Length + 1];
    for (int i = 0; i <= probs.Length - 1; i++)
    {
      cumul[i + 1] = cumul[i] + probs[i];
    }

    double p = random.NextDouble();

    for (int i = 0; i <= cumul.Length - 2; i++)
```

```
    {
      if (p >= cumul[i] && p < cumul[i + 1])
      {
        return i;
      }
    }

    return -1;
  }

  private static double[] MoveProbs(int k, int cityX, bool[] visited, double[][] pheromones, int[][] dists)
  {
    int numCities = pheromones.Length;
    double[] taueta = new double[numCities];
    double sum = 0.0;
    for (int i = 0; i <= taueta.Length - 1; i++)
    {
      if (i == cityX)
      {
        taueta[i] = 0.0;
      }
      else if (visited[i] == true)
      {
        taueta[i] = 0.0;
      }
      else
      {
        taueta[i] = Math.Pow(pheromones[cityX][i], alpha) * Math.Pow((1.0 / Distance(cityX, i, dists)), beta);
        if (taueta[i] < 0.0001)
        {
          taueta[i] = 0.0001;
        }
        else if (taueta[i] > (double.MaxValue / (numCities * 100)))
        {
          taueta[i] = double.MaxValue / (numCities * 100);
        }
      }
      sum += taueta[i];
    }

    double[] probs = new double[numCities];
    for (int i = 0; i <= probs.Length - 1; i++)
    {
      probs[i] = taueta[i] / sum;
    }
    return probs;
  }

  private static void UpdatePheromones(double[][] pheromones, int[][] ants, int[][] dists)
```

```
{
  for (int i = 0; i <= pheromones.Length - 1; i++)
  {
    for (int j = i + 1; j <= pheromones[i].Length - 1; j++)
    {
      for (int k = 0; k <= ants.Length - 1; k++)
      {
        double length = Length(ants[k], dists);
        double decrease = (1.0 - rho) * pheromones[i][j];
        double increase = 0.0;
        if (EdgeInTrail(i, j, ants[k]) == true)
        {
          increase = (Q / length);
        }

        pheromones[i][j] = decrease + increase;

        if (pheromones[i][j] < 0.0001)
        {
          pheromones[i][j] = 0.0001;
        }
        else if (pheromones[i][j] > 100000.0)
        {
          pheromones[i][j] = 100000.0;
        }

        pheromones[j][i] = pheromones[i][j];
      }
    }
  }
}

private static bool EdgeInTrail(int cityX, int cityY, int[] trail)
{
  int lastIndex = trail.Length - 1;
  int idx = IndexOfTarget(trail, cityX);

  if (idx == 0 && trail[1] == cityY)
  {
    return true;
  }
  else if (idx == 0 && trail[lastIndex] == cityY)
  {
    return true;
  }
  else if (idx == 0)
  {
    return false;
  }
```

```
    else if (idx == lastIndex && trail[lastIndex - 1] == cityY)
    {
        return true;
    }
    else if (idx == lastIndex && trail[0] == cityY)
    {
        return true;
    }
    else if (idx == lastIndex)
    {
        return false;
    }
    else if (trail[idx - 1] == cityY)
    {
        return true;
    }
    else if (trail[idx + 1] == cityY)
    {
        return true;
    }
    else
    {
        return false;
    }
}


private static int[][] MakeGraphDistances(int numCities)
{
    int[][] dists = new int[numCities][];
    for (int i = 0; i <= dists.Length - 1; i++)
    {
        dists[i] = new int[numCities];
    }
    for (int i = 0; i <= numCities - 1; i++)
    {
        for (int j = i + 1; j <= numCities - 1; j++)
        {
            int d = random.Next(1, 9);
            dists[i][j] = d;
            dists[j][i] = d;
        }
    }
    return dists;
}

private static double Distance(int cityX, int cityY, int[][] dists)
{
    return dists[cityX][cityY];
```

```
    }

    private static void Display(int[] trail)
    {
        for (int i = 0; i <= trail.Length - 1; i++)
        {
            Console.Write(trail[i] + " ");
            if (i > 0 && i % 20 == 0)
            {
                Console.WriteLine("");
            }
        }
        Console.WriteLine("");
    }


    private static void ShowAnts(int[][] ants, int[][] dists)
    {
        for (int i = 0; i <= ants.Length - 1; i++)
        {
            Console.Write(i + ": [ ");

            for (int j = 0; j < ants[i].Length; j++)
            {
                Console.Write(ants[i][j] + " ");
            }

            Console.Write("] довжина = ");
            double len = Length(ants[i], dists);
            Console.Write(len.ToString());
            Console.WriteLine("");
        }
    }

    private static void Display(double[][] pheromones)
    {
        for (int i = 0; i <= pheromones.Length - 1; i++)
        {
            Console.Write(i + ": ");
            for (int j = 0; j <= pheromones[i].Length - 1; j++)
            {
                Console.Write(pheromones[i][j].ToString("F4").PadLeft(8) + " ");
            }
            Console.WriteLine("");
        }

    }

}
```

```
}
```