МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра САПР

Отчет

по лабораторной работе № 2

по дисциплине «Основы искусственного интеллекта»

Тема: Муравьиный алгоритм

|  |  |  |
| --- | --- | --- |
| Студенты гр. 0302 |  | Хаматов В.Р. |
|  |  | Блюдин А.И. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2023

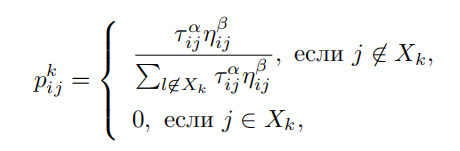
# Цель работы

Приобретение и закрепление знаний, получение практических навыков работы с муравьиным алгоритмом.

# Краткие теоретические сведения

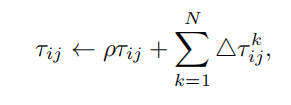
Данный алгоритм решает задачу коммивояжера. По условию нам даны n городов и расстояния между этими городами. Необходимо найти самый короткий замкнутый путь, проходящий через все города ровно по одному разу. То есть эту задачу можно представить в виде графа.

Для работы алгоритма создается колония из N муравьев и каждый из них расставляется в какую-то вершину графа. Изначально каждый путь между двумя вершинами помечается одинаковым значением феромона. После этого каждый муравей начинает перемещаться по графу, запоминая свой путь, чтобы посетить только тот город, который он еще не посещал. При выборе следующего города для посещения муравей опирается на значения двух величин: расстояние между текущим и следующим городом и количество феромона на дороге от текущего и до следующего города. Тогда вероятность, что k-ый муравей, находясь в вершине i, и имея посещенные города X\_k будет считаться по формуле:

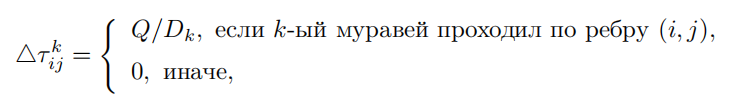


где коэффициенты альфа и бета отвечают за относительную важность значения феромона на дороге и расстояния между двумя городами.

После того, как все муравьи вернуться домой, сделав ровно по n перемещений, производится глобальное обновление значений феромона на дорогах между городами. Каждый муравей обновляет только те ребра, которые он посетил. Концентрация феромона на ребре (i, j) пересчитывается по формуле:



где — коэффициент испарения феромона, — количество феромона оставляемое на этом ребре k-ым муравьем:



Q — некоторый коэффициент, задаваемый опытным путем. Таким образом, чем короче получился путь k-го муравья, тем больше феромона он оставил на ребрах, по которым он ходил.

Такие итерации повторяются до того момента, пока не будет выполнено какое-нибудь условие прекращение работы алгоритма — исчерпано количество итераций, достигнута нужная точность, получен единственный путь (алгоритм сошелся к некоторому решению).

# Описание реализации

Диаграмма классов представлена на Рис. 3.1:

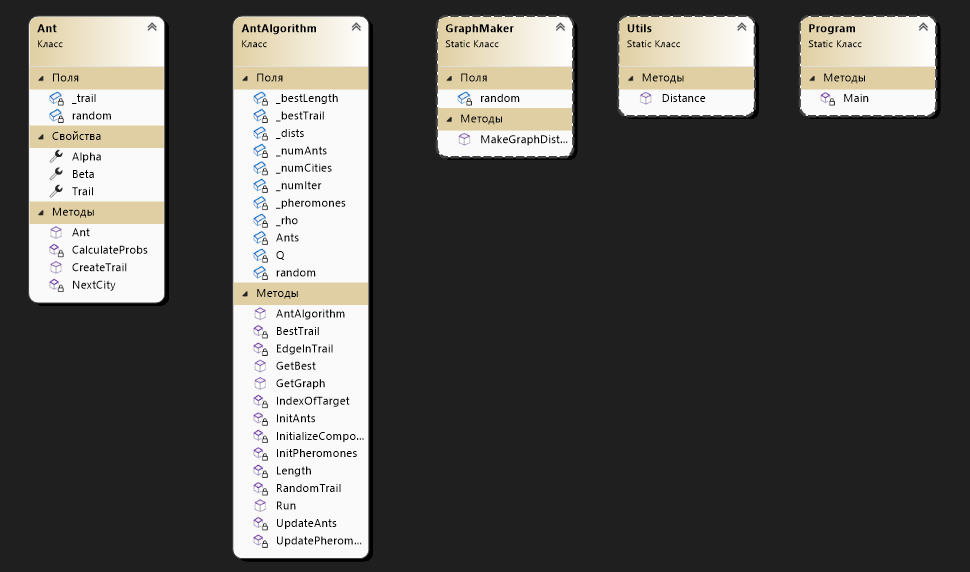


Рис. 3.1 – Диаграмма классов

### Описание класса Ant

Класс, реализующий логику действий муравья

Таблица 3.1.1.1 — Описание полей и свойств класса Ant



Таблица 3.1.1.2 — Описание методов класса Ant



### Описание класса AntAlgorithm

Класс, реализующий работу муравьиного алгоритма

Таблица 3.1.2.1 — Описание полей и свойств класса AntAlgorithm



Таблица 3.1.2.2 — Описание методов AntAlgorithm



### Описание класса GraphMaker

Класс, который генерирует произвольный граф заданного размера

Таблица 3.1.3.1 — Описание полей и свойств класса GraphMaker



Таблица 3.1.3.2 — Описание методов класса GraphMaker



### Описание класса Utils

Класс, который служит для возвращения расстояния между двумя городами

Таблица 3.1.4.1 — Описание методов класса Utils



# Пример работы программы

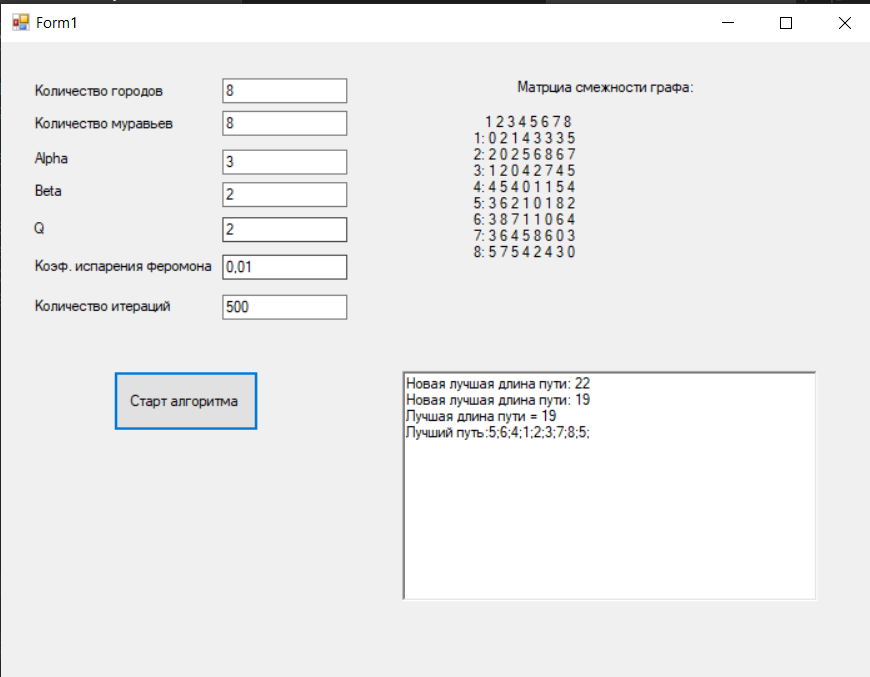


Рис. 4.1 — Пример программы

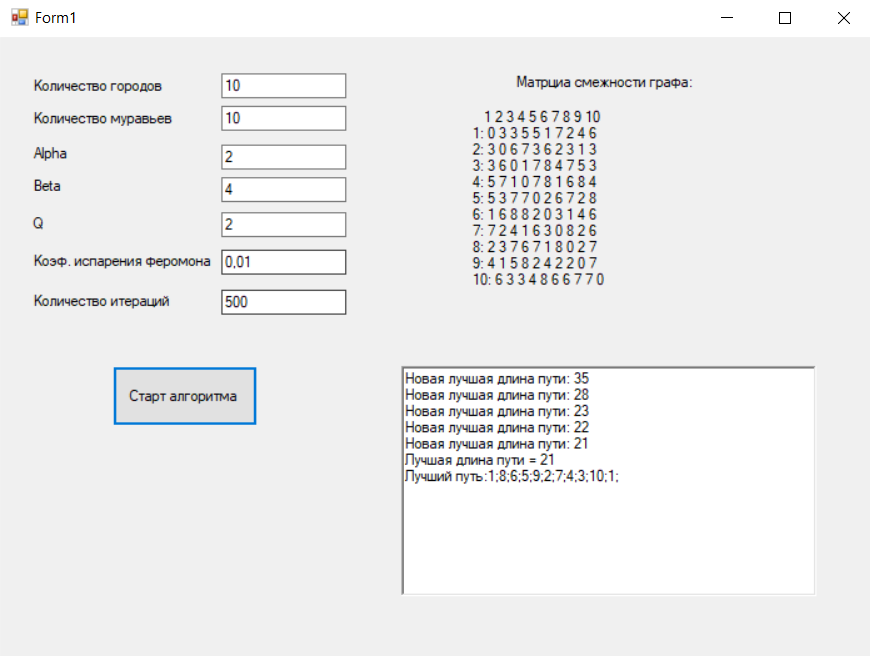


Рис. 4.2 — Пример программы

# Вывод

В ходе выполнения лабораторной работы были приобретены и закреплены знания, а также получены практические навыки работы с простейшими нейронными сетями. Был реализован алгоритм решения задачи коммивояжера — муравьиный алгоритм.

# Список литературы

1. Муравьиные алгоритмы. [Электронный ресурс] Режим доступа: https://habr.com/ru/articles/105302/ (дата обращения 10.10.2023).

2. Управление знаниями в распределенной информационной среде Сост.: А.В. Горячев, Н.Е. Новакова. СПБ.: Изд-во СПБГЭТУ “ЛЭТИ”, 2009. 32

### Приложение А

using System;

namespace AntColony

{

public class Ant

{

private int[] \_trail;

public int[] Trail => \_trail;

public static int Alpha { get; set; }

public static int Beta { get; set; }

Random random = new Random();

public Ant(int[] trail)

{

\_trail = trail;

}

public void CreateTrail(int start, double[][] pheromones, int[][] dists)

{

\_trail = new int[\_trail.Length];

bool[] visited = new bool[\_trail.Length];

\_trail[0] = start;

visited[start] = true;

for (int i = 0; i <= \_trail.Length - 2; i++)

{

int cityX = \_trail[i];

int next = NextCity(cityX, visited,pheromones,dists);

\_trail[i + 1] = next;

visited[next] = true;

}

}

private int NextCity(int cityX, bool[] visited, double[][] pheromones, int[][] dists)

{

double[] probs = CalculateProbs(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;

throw new Exception("Failure to return valid city in NextCity");

}

private double[] CalculateProbs(int cityX, bool[] visited, double[][] pheromones, int[][] dists)

{

double[] taueta = new double[\_trail.Length];

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 / Utils.Distance(dists,cityX, i)), Beta);

if (taueta[i] < 0.0001)

taueta[i] = 0.0001;

else if (taueta[i] > (double.MaxValue / (\_trail.Length \* 100)))

taueta[i] = double.MaxValue / (\_trail.Length \* 100);

}

sum += taueta[i];

}

double[] probs = new double[\_trail.Length];

for (int i = 0; i <= probs.Length - 1; i++)

probs[i] = taueta[i] / sum;

return probs;

}

}

}

### Приложение B

using System;

using System.Collections.Generic;

namespace AntColony

{

public class AntAlgorithm

{

private int \_numCities;

private int \_numAnts;

private double \_bestLength, \_rho, Q;

private int[] \_bestTrail;

private int[][] \_dists;

private Ant[] Ants = null;

private double[][] \_pheromones;

private static Random random = new Random(0);

private int \_numIter = 1000;

public AntAlgorithm(int numCities, int numAnts, int alpha, int beta, double Q, double rho, int numIter, int[][] dists = null)

{

Ant.Alpha = alpha;

Ant.Beta = beta;

\_numCities = numCities;

\_numAnts = numAnts;

\_rho = rho;

\_numIter = numIter;

this.Q = Q;

if (dists != null)

\_dists = dists;

else

\_dists = GraphMaker.MakeGraphDistances(\_numCities);

InitializeComponents();

}

private void InitializeComponents()

{

InitAnts();

\_pheromones = InitPheromones();

}

public IEnumerable<double> Run()

{

\_bestTrail = BestTrail();

\_bestLength = Length(\_bestTrail);

int iter = 0;

while (iter < \_numIter)

{

UpdateAnts();

UpdatePheromones();

int[] curr\_bestTrail = BestTrail();

double curr\_bestLength = Length(curr\_bestTrail);

if (curr\_bestLength < \_bestLength)

{

\_bestLength = curr\_bestLength;

\_bestTrail = curr\_bestTrail;

yield return \_bestLength;

}

iter++;

}

}

public (double, int[]) GetBest() => (\_bestLength, \_bestTrail);

public int[][] GetGraph() => \_dists;

private void InitAnts()

{

Ants = new Ant[\_numAnts];

for (int k = 0; k <= \_numAnts - 1; k++)

{

int start = random.Next(0, \_numCities);

Ants[k] = new Ant(RandomTrail(start));

}

}

private int[] RandomTrail(int start)

{

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 int IndexOfTarget(int[] trail, int target)

{

for (int i = 0; i <= trail.Length - 1; i++)

if (trail[i] == target)

return i;

throw new Exception("Target not found in IndexOfTarget");

}

private double Length(int[] trail)

{

double result = 0.0;

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

result += Utils.Distance(\_dists,trail[i], trail[i + 1]);

result += Utils.Distance(\_dists, trail[0], trail[trail.Length - 1]);

return result;

}

private int[] BestTrail()

{

double \_bestLength = Length(Ants[0].Trail);

int idx\_bestLength = 0;

for (int k = 1; k <= Ants.Length - 1; k++)

{

double len = Length(Ants[k].Trail);

if (len < \_bestLength)

{

\_bestLength = len;

idx\_bestLength = k;

}

}

int[] \_bestTrail\_Renamed = new int[\_numCities];

Ants[idx\_bestLength].Trail.CopyTo(\_bestTrail\_Renamed, 0);

return \_bestTrail\_Renamed;

}

private double[][] InitPheromones()

{

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 void UpdateAnts()

{

for (int k = 0; k <= Ants.Length - 1; k++)

{

int start = random.Next(0, \_numCities);

Ants[k].CreateTrail(start, \_pheromones, \_dists);

}

}

private void UpdatePheromones()

{

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].Trail);

// length of ant k trail

double decrease = (1.0 - \_rho) \* \_pheromones[i][j];

double increase = 0.0;

if (EdgeInTrail(i, j, Ants[k].Trail) == 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 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;

}

}

}

}

### Приложение С

using System;

namespace AntColony

{

public static class GraphMaker

{

static Random random = new Random();

public 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;

}

}

}

### Приложение D

namespace AntColony

{

public static class Utils

{

public static double Distance(int[][] dists,int cityX, int cityY) => dists[cityX][cityY];

}

}