**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №1**

**по дисциплине «Основы искусственного интеллекта»**

Тема: Муравьиный алгоритм

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9302 |  | Ширнин К.В. |
|  |  | Квитко Д.В. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2022

**Содержание**

[Цель работы 3](#_Toc120489151)

[Особенности реализации алгоритма 4](#_Toc120489152)

[UML-диаграмма 5](#_Toc120489153)

[Ход работы 6](#_Toc120489154)

[Пример работы программы 7](#_Toc120489155)

[Вывод 9](#_Toc120489156)

[Список используемой литературы 9](#_Toc120489157)

[Листинг 9](#_Toc120489158)

# Цель работы

Изучить принцип работы муравьиного алгоритма. Написать ПО с пользовательским графическим интерфейсом, находящее кратчайший маршрут среди всех узлов данного графа, с использованием муравьиного алгоритма

**Теория**

Вероятность посещения муравьем смежной точки определяется формулой:

,

где – коэффициент привлекательности феромона,

- коэффициент привлекательности кратчайшего пути,

- количество феромона на пути к вершине,

– обратная величина пути к вершине,

– любая смежная доступная вершина,

i, j – номера вершин, которые соединяет ребро.

Испарение старого феромона с заданным коэффициентом определяется по формуле:

,

где – количество феромона на дорожке,

t – номер итерации,

i, j – номера вершин, которые соединяет ребро,

p – коэффициент испарения феромона.

Количество феромона, которое добавляется на каждый пройдённый муравьями маршрут, рассчитывается по следующей формуле:

Где Q – заданный пользователем коэффициент,

– пройденный муравьем путь.

# Особенности реализации алгоритма

Для реализации муравьиного алгоритма ПО было разделено на две составляющие, а именно на GUI и сам программный код алгоритма. Программный комплекс был написан в рамках правил ООП и без особых трудностей может использоваться как отдельный виджет в других ПО или как библиотека.

Ход алгоритма делится на несколько этапов – он делится на итерации. После каждой итерации производится обновление количества феромона на дорожках – рёбрах графа – с учетом скорости их испарения и других заданных пользователем параметров. Одна итерация, в свою очередь, делится на проход пачки муравьев. Их количество задается пользователем. Каждый муравей из пачки может пойти в абсолютно любую другую смежную вершину, даже если на этой дорожке большое количество феромона. Это происходит из-за того, что выбор муравья определяется случайной составляющей. С учетом удаленности вершины и рассыпанного по пути к ней феромона муравью более привлекательна та или иная вершина, но даже у самой непривлекательной точки будет иметься вероятность посещения её муравьем. Это позволяет изменить даже, самый лучший маршрут на другой и найти еще более оптимальный путь. После того как все муравьи прошли все вершины - производится обновление количества феромона на дорожках. Сначала испаряется старый феромон с заданным коэффициентом. После, ко всем пройденным маршрутам муравьями на данной итерации добавляется некоторое количество феромона.

# UML-диаграмма

На рисунке 1 представлена UML-диаграмма структуры программы

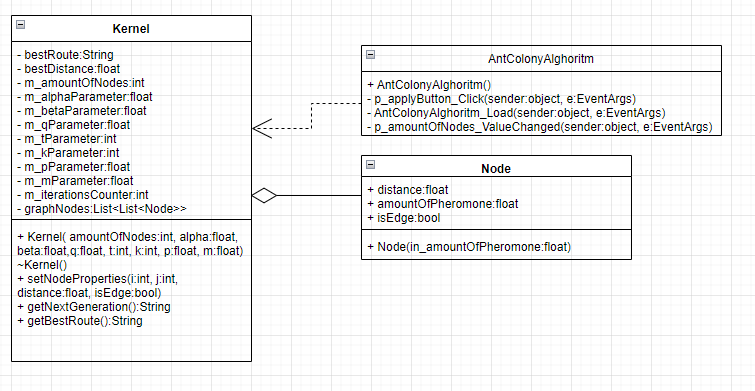


Рис. 1. UML-диаграмма

Таблица 1. Методы и их описание

|  |  |
| --- | --- |
| Метод | Описание |
| public Kernel(int amountOfNodes,float alpha,float beta,float q,int t,int k,float p,float m) | Конструктор класса Kernel |
| ~Kernel() | Деструктор класса Kernel |
| public void setNodeProperties(int i, int j, float distance, bool isEdge) | Устанавливает параметры Node’а |
| public String getNextGeneration() | Вызывает срабатывание одного такта алгоритма |
| public String getBestRoute() | Возвращает лучший маршрут |
| public Node(float in\_amountOfPheromone) | Конструктор класса Node |
| private void p\_applyButton\_Click(object sender, EventArgs e) | Запускает работу алгоритма при нажатии пользователем кнопки |
| private void AntColonyAlghoritm\_Load(object sender, EventArgs e) | Вызывается при загрузки формы запускает алгоритм построения динамических ячеек |
| private void p\_amountOfNodes\_ValueChanged(object sender, EventArgs e) | Алгоритм построения динамических ячеек |

# Ход работы

Для выполнения данной лабораторной работы был выбран язык C#.

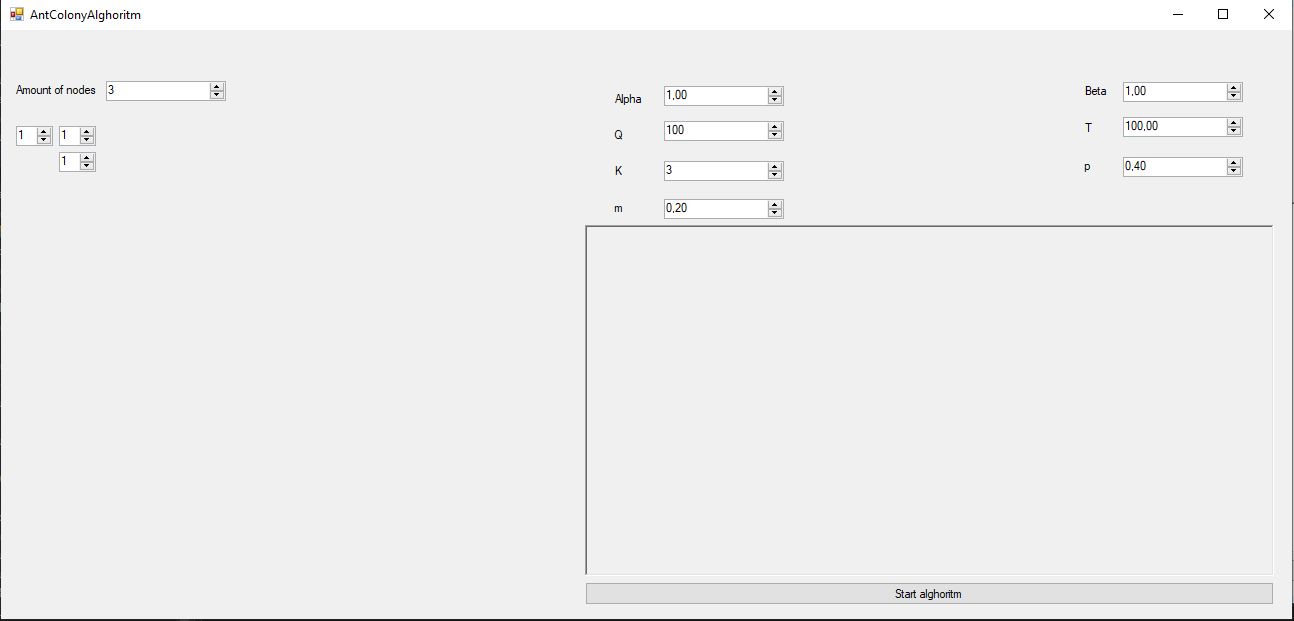


Рис.2. Главное окно приложения

При выполнении работы, были взяты основные используемые параметры:

* Альфа – коэффициент привлекательности феромона на дорожке
* Бета – коэффициент привлекательности более короткой дорожки
* Q – Константа, определяющая сколько феромона оставит за собой муравей
* T – Количество итераций алгоритма
* K – Количество муравьев (кол-во итераций без обновления количества феромонов на дорожках)
* p – Скорость испарения феромона с дорожек
* m – Начальное количество феромона на дорожка

Для задачи графа была реализована динамическая составляющая пользовательского интерфейса. Пользователю достаточно выставить в поле amount of nodes количество вершин в графе, после чего программа сама отобразит матрицу смежности нужного размера. Она представлена в виде треугольника, т.к. в муравьином алгоритме будет производиться расчет на неориентированном графе (что указывает на симметричность матрицу относительно главной диагонали). Это позволит сэкономить время при вводе данных, а также в два раза сократит расходы на выделяемую память, так как хранение внутри программы производится тем же самым образом.

При выполнении работы также были учтены крайние значения, вводимые пользователем в программу.

# Пример работы программы

Пример работы программы представлен на рисунках ниже.

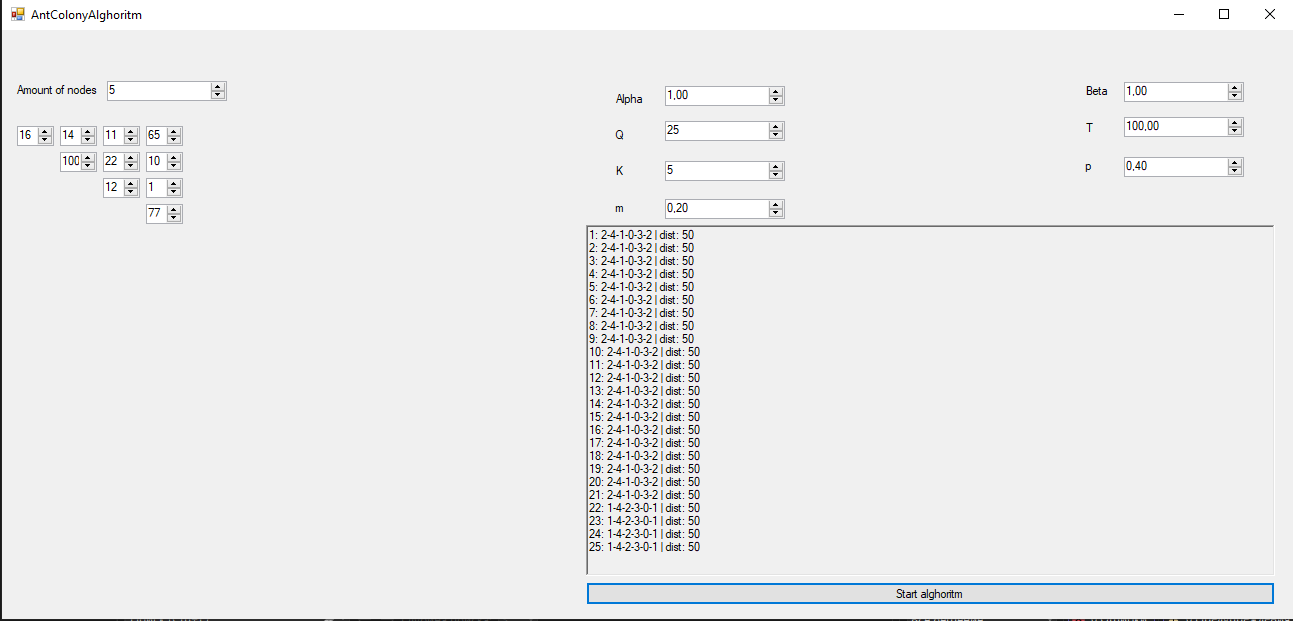


Рис. 3. Пример работы программы для 5 вершин

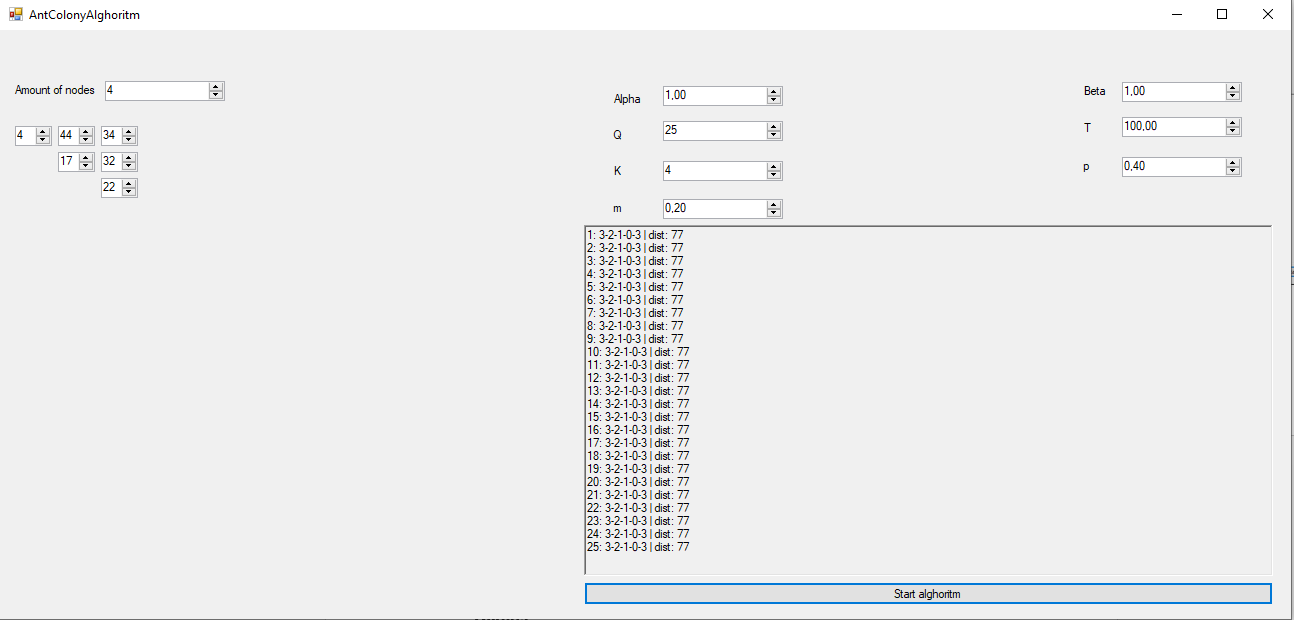


Рис.4. Пример работы программы для 4 вершин

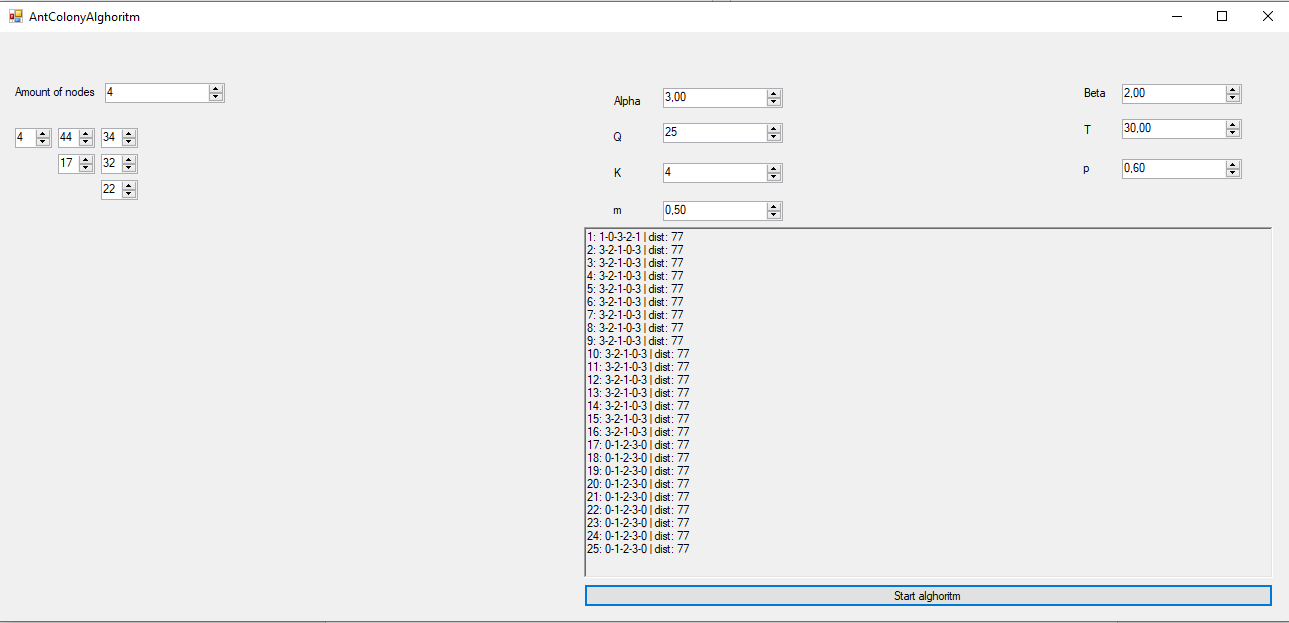


Рис.5. Пример работы программы для 4 вершин

# Вывод

В ходе выполнения работы стало ясно, что муравьиный алгоритм очень сильно зависим от заданных констант и параметров. Абсолютно к каждой новой задаваемой матрице смежности требуется подбирать свои индивидуальные параметры. Если подобрать конфигурацию неверно – алгоритм может проложить сильно неверный путь или зайти в тупик, что не имеет никакой практической ценности для решения поставленных задач. Алгоритм был успешно реализован на языке C#. В лабораторной работе были достигнуты все цели.

# Список используемой литературы

1. Муравьиные алгоритмы // Хабр URL: https://habr.com/ru/post/105302/ (дата обращения: 21.09.2022).

# Листинг

AntColonyAlghoritm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace AntColonyAlgorithm

{

public partial class AntColonyAlghoritm : Form

{

public AntColonyAlghoritm()

{

InitializeComponent();

}

int x, x1, y, y1;

List<List<NumericUpDown>> listOfGraphNodes = new List<List<NumericUpDown>>();

private void p\_applyButton\_Click(object sender, EventArgs e)

{

p\_resultBrowser.Clear();

Kernel kernel = new Kernel((int)p\_amountOfNodes.Value,(float) p\_alphaParameter.Value, (float)p\_betaParameter.Value,

(float)p\_qParameter.Value, (int)p\_tParameter.Value, (int)p\_kParameter.Value, (float)p\_pParameter.Value,

(float)p\_mParameter.Value);

for (int i = 1; i < p\_amountOfNodes.Value; ++i)

{

for (int j = 0; j < p\_amountOfNodes.Value - 1; ++j)

{

if (i > j)

{

kernel.setNodeProperties(i, j, (float)listOfGraphNodes[i - 1][j].Value, true);

}

}

}

for (int i = 0; i < p\_qParameter.Value; ++i)

{

p\_resultBrowser.Text += (i + 1) + ": " + kernel.getNextGeneration();

p\_resultBrowser.AppendText(Environment.NewLine);

}

//p\_resultBrowser.Text += "Best route: " + kernel.getBestRoute();

}

private void AntColonyAlghoritm\_Load(object sender, EventArgs e)

{

x = 96;

y = 15;

p\_amountOfNodes\_ValueChanged(sender, e);

}

private void p\_amountOfNodes\_ValueChanged(object sender, EventArgs e)

{

y1 = y;

for (int i = 0; i < listOfGraphNodes.Count(); i++)

for (int j = 0; j < listOfGraphNodes[i].Count(); j++)

this.Controls.Remove(listOfGraphNodes[i][j]);

listOfGraphNodes.Clear();

for (int i = 1; i < p\_amountOfNodes.Value; i++)

{

x1 = x;

listOfGraphNodes.Add(new List<NumericUpDown>());

for (int j = 0; j < p\_amountOfNodes.Value - 1; j++)

{

if (i > j)

{

listOfGraphNodes[i - 1].Add(new NumericUpDown());

listOfGraphNodes[i - 1][j].Size = new Size(37, 20);

listOfGraphNodes[i - 1][j].Location = new Point(y1, x1);

listOfGraphNodes[i - 1][j].Minimum = 1;

Controls.Add(listOfGraphNodes[i - 1][j]);

x1 += 26;

}

}

y1 += 43;

}

}

}

}

Kernel.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace AntColonyAlgorithm

{

public partial class AntColonyAlghoritm : Form

{

public AntColonyAlghoritm()

{

InitializeComponent();

}

int x, x1, y, y1;

List<List<NumericUpDown>> listOfGraphNodes = new List<List<NumericUpDown>>();

private void p\_applyButton\_Click(object sender, EventArgs e)

{

p\_resultBrowser.Clear();

Kernel kernel = new Kernel((int)p\_amountOfNodes.Value,(float) p\_alphaParameter.Value, (float)p\_betaParameter.Value,

(float)p\_qParameter.Value, (int)p\_tParameter.Value, (int)p\_kParameter.Value, (float)p\_pParameter.Value,

(float)p\_mParameter.Value);

for (int i = 1; i < p\_amountOfNodes.Value; ++i)

{

for (int j = 0; j < p\_amountOfNodes.Value - 1; ++j)

{

if (i > j)

{

kernel.setNodeProperties(i, j, (float)listOfGraphNodes[i - 1][j].Value, true);

}

}

}

for (int i = 0; i < p\_qParameter.Value; ++i)

{

p\_resultBrowser.Text += (i + 1) + ": " + kernel.getNextGeneration();

p\_resultBrowser.AppendText(Environment.NewLine);

}

//p\_resultBrowser.Text += "Best route: " + kernel.getBestRoute();

}

private void AntColonyAlghoritm\_Load(object sender, EventArgs e)

{

x = 96;

y = 15;

p\_amountOfNodes\_ValueChanged(sender, e);

}

private void p\_amountOfNodes\_ValueChanged(object sender, EventArgs e)

{

y1 = y;

for (int i = 0; i < listOfGraphNodes.Count(); i++)

for (int j = 0; j < listOfGraphNodes[i].Count(); j++)

this.Controls.Remove(listOfGraphNodes[i][j]);

listOfGraphNodes.Clear();

for (int i = 1; i < p\_amountOfNodes.Value; i++)

{

x1 = x;

listOfGraphNodes.Add(new List<NumericUpDown>());

for (int j = 0; j < p\_amountOfNodes.Value - 1; j++)

{

if (i > j)

{

listOfGraphNodes[i - 1].Add(new NumericUpDown());

listOfGraphNodes[i - 1][j].Size = new Size(37, 20);

listOfGraphNodes[i - 1][j].Location = new Point(y1, x1);

listOfGraphNodes[i - 1][j].Minimum = 1;

Controls.Add(listOfGraphNodes[i - 1][j]);

x1 += 26;

}

}

y1 += 43;

}

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace AntColonyAlgorithm

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new AntColonyAlghoritm());

}

}

}

# 