**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](#_Toc121424506)

[UML-диаграмма 4](#_Toc121424507)

[Описание классов 4](#_Toc121424508)

[Ход работы 7](#_Toc121424509)

[Пример работы программы 8](#_Toc121424510)

[Вывод 10](#_Toc121424511)

[Список используемой литературы 10](#_Toc121424512)

[Приложение 11](#_Toc121424513)

# 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**

Муравьиный алгоритм один из эффективных полиномиальных 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.

В реальном мире муравьи (первоначально) ходят в случайном порядке и после нахождении продовольствия возвращаются в свою колонию, прокладывая феромонами тропы. Если другие муравьи находят такие тропы, они, вероятнее всего, пойдут по ним. Вместо того, чтобы отслеживать цепочку, они укрепляют её при возвращении, если в конечном итоге находят источник питания. Со временем феромонная тропа начинает испаряться, тем самым уменьшая свою привлекательную силу. Чем больше времени требуется для прохождения пути до цели и обратно, тем сильнее испарится феромонная тропа. На коротком пути, для сравнения, прохождение будет более быстрым, и, как следствие, плотность феромонов остаётся высокой. Испарение феромонов также имеет функцию избежания стремления к локально-оптимальному решению. Если бы феромоны не испарялись, то путь, выбранный первым, был бы самым привлекательным. В этом случае, исследования пространственных решений были бы ограниченными. Таким образом, когда один муравей находит (например, короткий) путь от колонии до источника пищи, другие муравьи, скорее всего пойдут по этому пути, и положительные отзывы в конечном итоге приводят всех муравьёв к одному, кратчайшему, пути.

Задача коммивояжёра одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз — в таком случае выбор осуществляется среди гамильтоновых циклов.

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,

– номера вершин, которые соединяет ребро.

Испарение старого феромона с заданным коэффициентом определяется по формуле:

,

где – количество феромона на дорожке,

– номер итерации,

– номера вершин, которые соединяет ребро,

– коэффициент испарения феромона.

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

Где – заданный пользователем коэффициент,

– пройденный муравьем путь.

# UML-диаграмма

На рисунке 1 представлена UML-диаграмма структуры программы

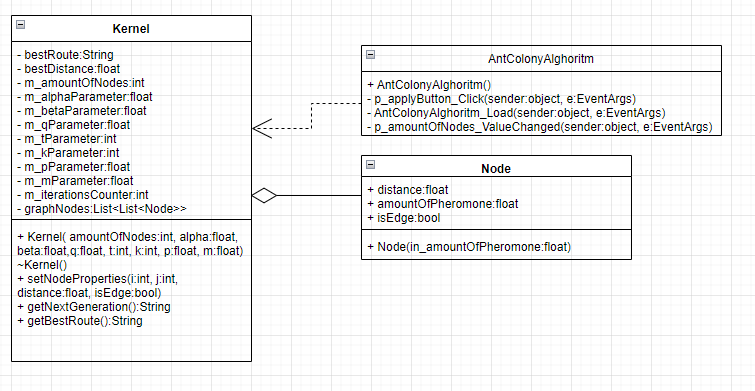


Рис. 1. UML-диаграмма

# Описание классов

Таблица 1 Описание методов класса Kernel

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Kernel | - | public | int amountOfNodes,float alpha,float beta,float q,int t,int k,float p,float m | - | Конструктор |
| setNodeProperties | void | public | int i, int j, float distance, bool isEdge | - | Устанавливает параметры Node’а |
| getNextGeneration() | String | public | - | result | Вызывает срабатывание одного такта алгоритма |
| getBestRoute() | String | public | - | bestRoute | Возвращает лучший маршрут |

Таблица 2 Описание полей и свойств класса Kernel

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| bestRoute | String | private | Хранит вершины лучшего маршрута |
| bestDistance | float | private | Хранит размер лучшей дистанции |
| m\_iterationsCounter | int | private | Количество итераций |
| m\_amountOfNodes | int | private | Количество вершин |
| m\_alphaParameter | float | private | Коэффициент привлекательности феромона на дорожке |
| m\_betaParameter | float | private | Коэффициент привлекательности более короткой дорожки |
| m\_qParameter | float | private | Сколько феромона оставит за собой муравей |
| m\_tParameter | int | private | Количество итераций алгоритма |
| m\_kParameter | int | private | Количество муравьев |
| m\_pParameter | float | private | Скорость испарения феромона с дорожек |
| m\_mParameter | float | private | Начальное количество феромона на дорожка |
| graphNodes | List<List<Node>> | private | Треугольная матрица смежности |

Таблица 3 Описание методов класса Node

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Node | - | public | float in\_amountOfPheromone | - | Конструктор |

Таблица 4 Описание полей и свойств класса Node

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| distance | float | private | Хранит значение дистанции |
| amountOfPheromone | float | private | Количество феромонов |
| isEdge | bool | private | Существует ли вершина |

Таблица 5 Описание методов класса AntColonyAlghoritm

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| p\_applyButton\_Click | void | private | object sender, EventArgs e | - | Запускает работу алгоритма при нажатии пользователем кнопки |
| AntColonyAlghoritm\_Load | void | private | object sender, EventArgs e | - | Вызывается при загрузки формы запускает алгоритм построения динамических ячеек |
| p\_amountOfNodes\_ValueChanged | void | private | object sender, EventArgs e | - | Алгоритм построения динамических ячеек |

Таблица 6 Описание полей и свойств класса AntColonyAlghoritm

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| x | int | private | Начальная координата |
| x1 | int | private | Координата смещения |
| y | int | private | Начальная координата |
| y1 | int | private | Координата смещения |
| listOfGraphNodes | List<List<NumericUpDown>> | private | Список длины ребер |

# Ход работы

Для выполнения данной лабораторной работы был выбран язык C#.

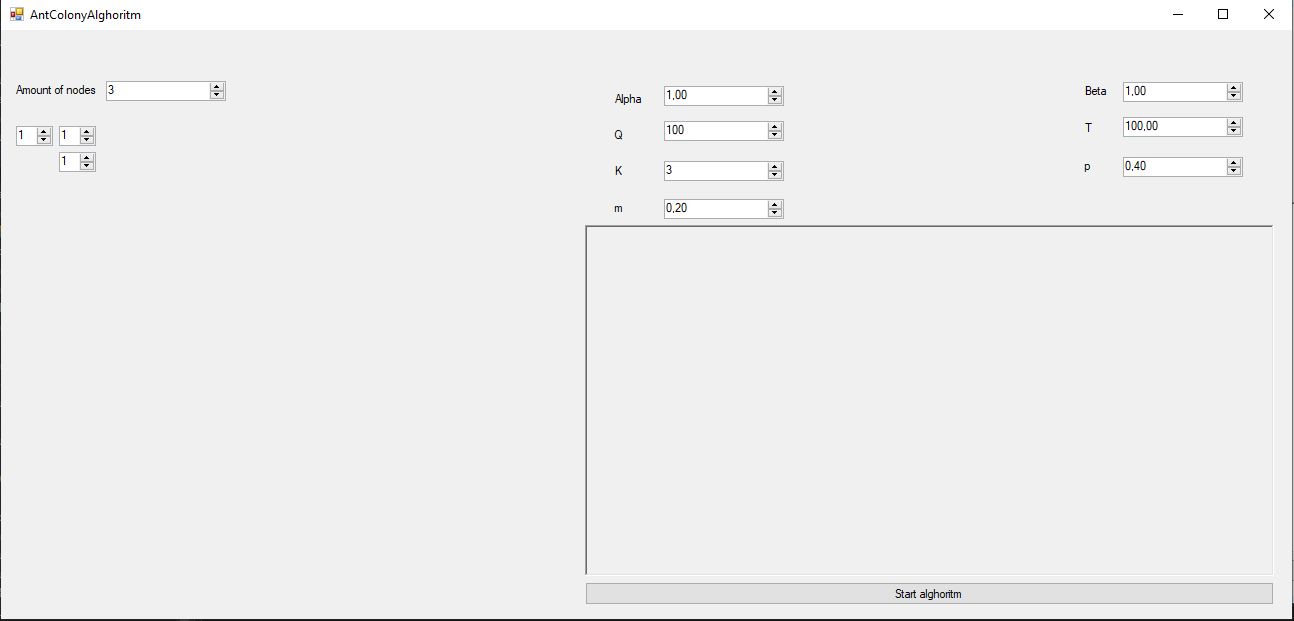


Рис.2. Главное окно приложения

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. В поле Amout of nodes вводится количество вершин графа. В нижних ячейках вводится расстояния от одной вершины к другой. В ячейке альфа коэффициент привлекательности феромона на дорожке, в ячейке бета коэффициент привлекательности более короткой дорожки, в ячейке Q Константа, определяющая сколько феромона оставит за собой муравей, в ячейке T Количество итераций алгоритма, в ячейке K Количество муравьев (кол-во итераций без обновления количества феромонов на дорожках), в ячейке P Скорость испарения феромона с дорожек, в ячейке m Начальное количество феромона на дорожка. При нажатии пользователем кнопки Start alghoritm программа находит наилучший маршрут для каждой итерации, когда алгоритм подходит к концу, программа выводит наилучший маршрут который она нашла за все проходу по алгоритму.

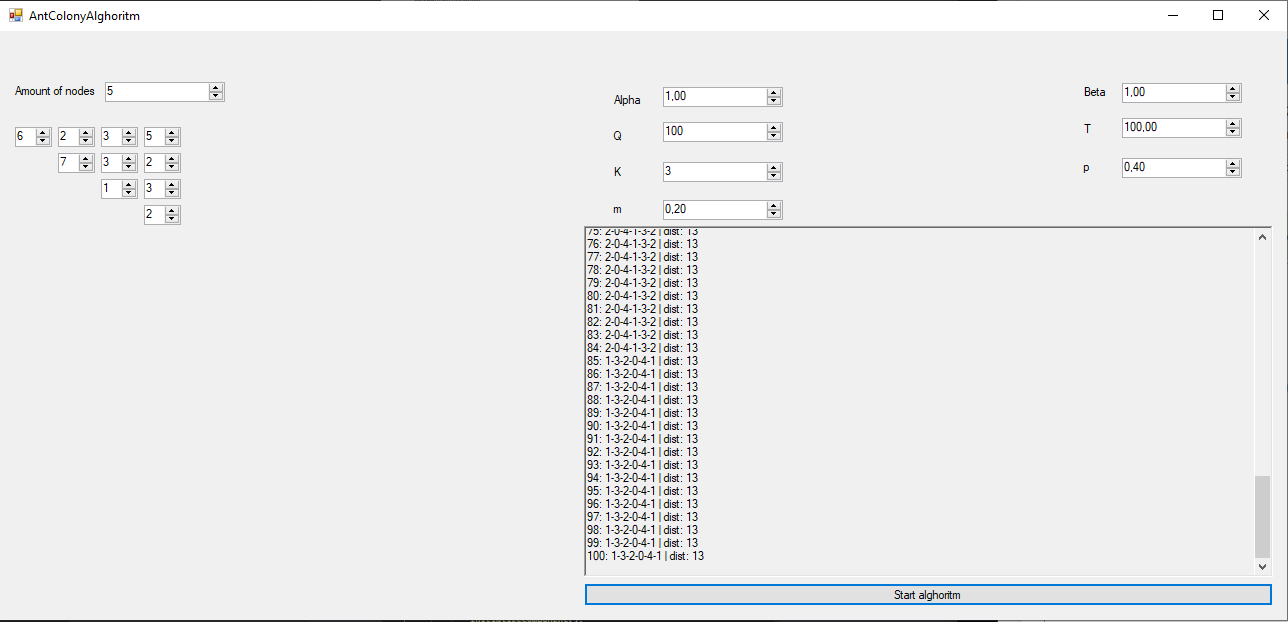


Рис.3.Пример работы программы для 5 вершин

Программа нашла наилучший маршрут, что подтверждает данные со стороннего сайт.

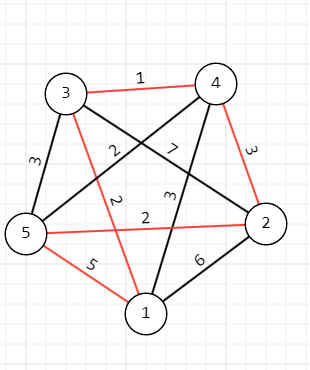


Рис.4.Лучший маршрут вычисленный с помощью стороннего сайта

Еще несколько примеров:

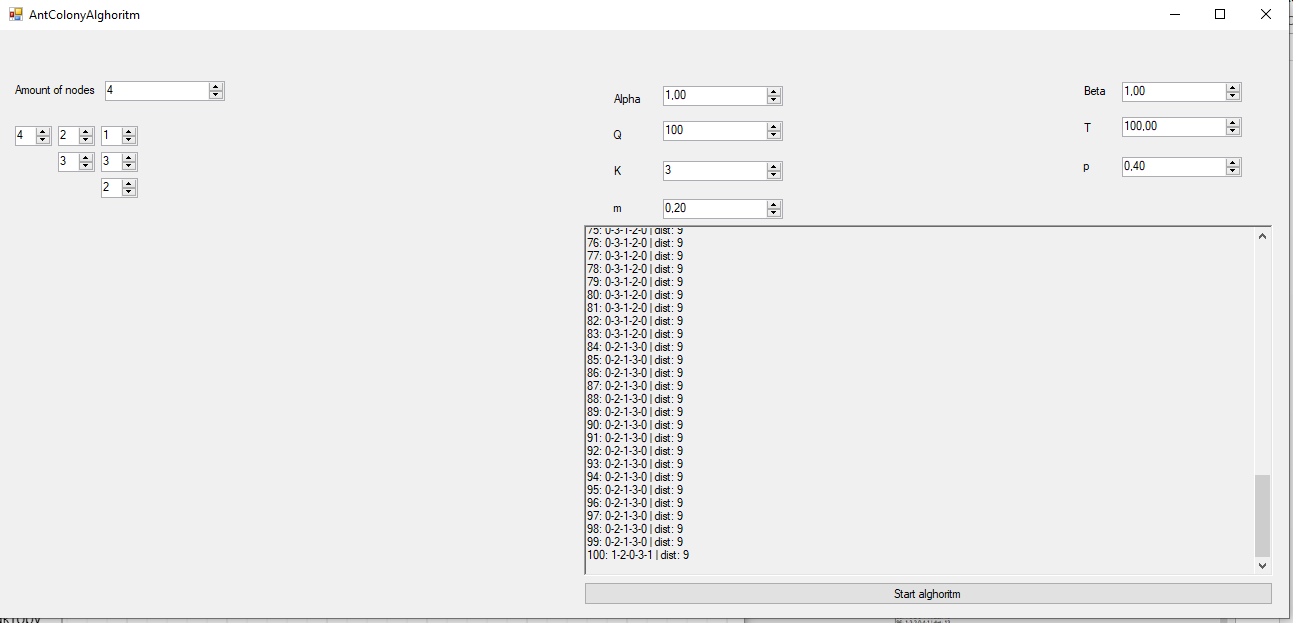


Рис.5.Пример работы программы для 4 вершин

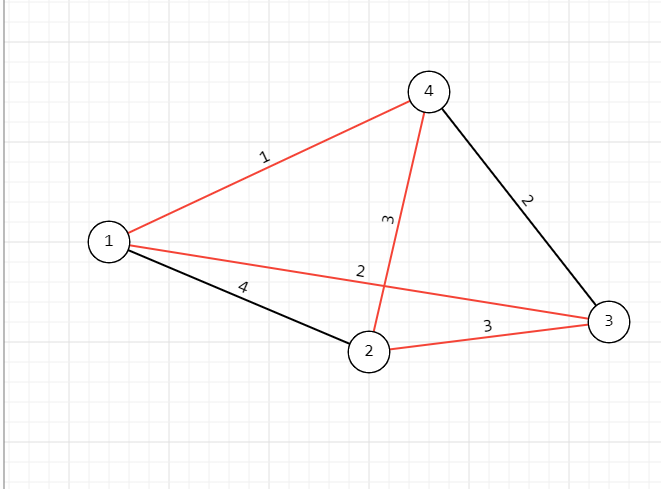


Рис.6.Лучший маршрут вычисленный с помощью стороннего сайта

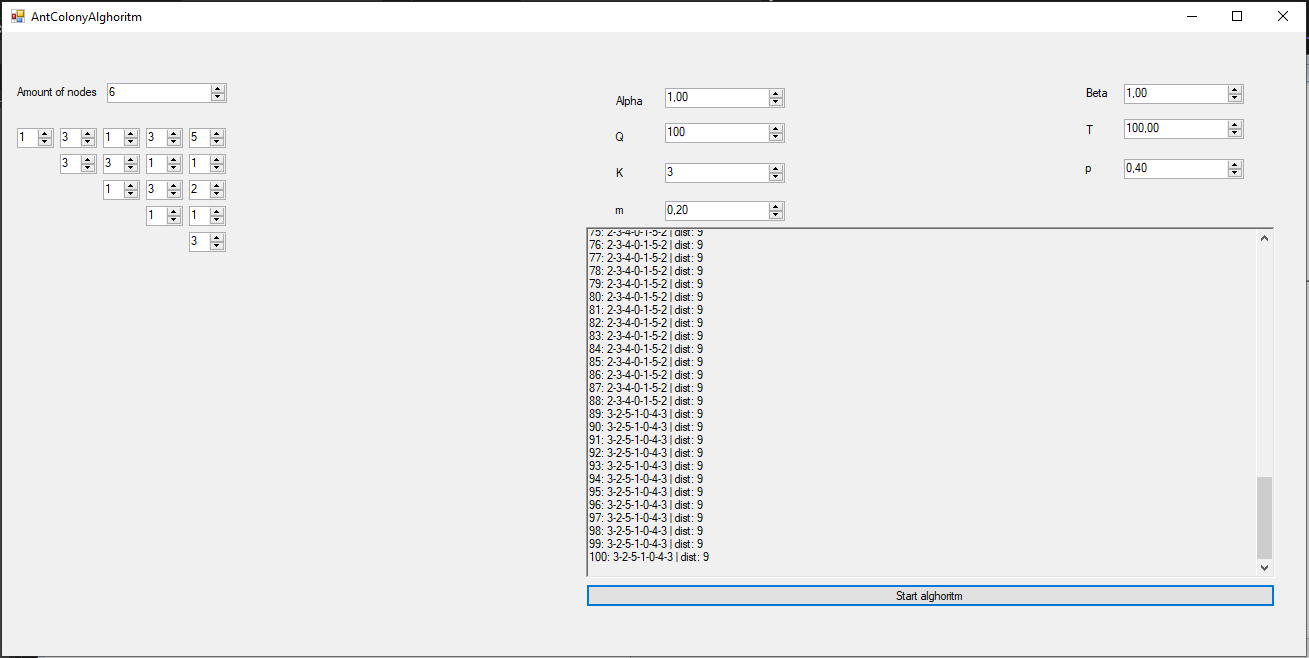


Рис.7. Пример работы программы для 4 вершин

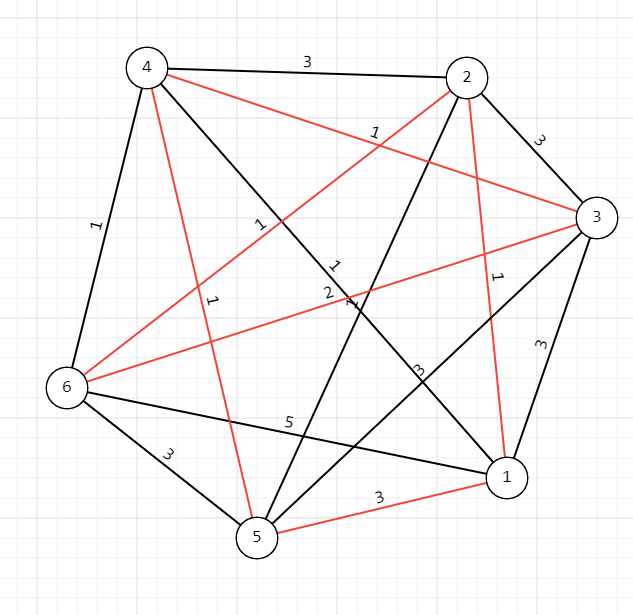


Рис.8. Лучший маршрут вычисленный с помощью стороннего сайта

# 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).

# Приложение

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());

}

}

}

# 