Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 4 з дисципліни
«Проектування алгоритмів»

“Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1”

Виконав(ла) ІП-14 Ковалик Назар Миколайович
(шифр, прізвище, ім’я, по батькові)

Перевірив _____
(прізвище, ім’я, по батькові)

Київ 2022

ЗМІСТ

1. МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою.

2. ЗАВДАННЯ

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

№	**Задача і алгоритм**
8	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0{,}3$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах).

3. ВИКОНАННЯ

1. Програмна реалізація алгоритму

1. Вихідний код

`MainWindow.xaml`

```
<Window x:Class="pa4.Presentation.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:pa4.Presentation"
        mc:Ignorable="d"
        Title="MainWindow" Height="800" Width="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="181*"/>
            <ColumnDefinition Width="619*"/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="0">
            <TextBlock>a</TextBlock>
            <TextBox Text ="3" Name="aT"></TextBox>

            <TextBlock>b</TextBlock>
            <TextBox Text="2" Name="bT"></TextBox>

            <TextBlock>p</TextBlock>
            <TextBox Text="0.3" Name="pT"></TextBox>

            <TextBlock>ants count</TextBlock>
            <TextBox Name="antsT" Text="25"></TextBox>

            <TextBlock>points count</TextBlock>
            <TextBox Name="pointsT" Text="100"></TextBox>
            <Button
                Name="generateB"
                Click="generateB_Click"
              >
                Generate
            </Button>

            <Button Click="Button_Click"
                    Name="startB"
                    IsEnabled="False">
                Start
            </Button>
            <Button Click="Button_Click_1"
                    Name="stopB"
                    IsEnabled="False">Stop</Button>
            <TextBlock Name="output" Height="155"
```

MainWindow.cs

```
using pa4.Algorithm;
using pa4.Algorithm.Settings;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Shapes;
using System.Windows.Threading;
```

```
namespace pa4.Presentation;

/// <summary>
/// Interaction logic for MainWindow.xaml
/// </summary>
public partial class MainWindow : Window
{
    private readonly ConfigurationValidation _validation = new();
    private readonly DispatcherTimer _drawTimer = new();
    private readonly DispatcherTimer _stepTimer = new();
    private Search _search;
    public MainWindow()
    {
        InitializeComponent();
        _stepTimer.Tick += _stepTimer_Tick;
        _stepTimer.Interval = TimeSpan.FromMilliseconds(10);

        _drawTimer.Tick += _drawTimer_Tick;
        _drawTimer.Interval = TimeSpan.FromMilliseconds(100);
    }

    private void _drawTimer_Tick(object? sender, EventArgs e)
    {
        ReDrawCadr();
    }

    private void _stepTimer_Tick(object? sender, EventArgs e)
    {
        _search.UpdateRoades();
    }

    void ReDrawCadr()
    {
        canvas.Children.Clear();
        for (int i = 0; i < _search.Points.Count; i++)
        {
            for (int j = 0; j < _search.Points.Count; ++j)
            {
                if (i == j && _search.Pheramones[i, j] <= 0.0001)
                {
                    continue;
                }

                var line = GetGraphLine(i, j);

                canvas.Children.Add(line);
            }
        }

        for (int i = 0; i < _search.Points.Count; i++)
        {
            var eclipse = GetGraphPoint();

            Canvas.SetTop(eclipse, _search.Points[i].Y - 5);
            Canvas.SetLeft(eclipse, _search.Points[i].X - 5);
            canvas.Children.Add(eclipse);
        }
    }

    private static Ellipse GetGraphPoint()
    {
        return new()
        {
            Fill = new SolidColorBrush(Colors.Green),
            Width = 10,
            Height = 10
        };
    }

    private Line GetGraphLine(int i, int j)
```

```
{
    return new()
    {
        X1 = _search.Points[i].X,
        Y1 = _search.Points[i].Y,
        X2 = _search.Points[j].X,
        Y2 = _search.Points[j].Y,
        Opacity = 2 * Math.Atan(_search.Pheramones[i, j]) / Math.PI,
        Stroke = new SolidColorBrush(Colors.Black),
        StrokeThickness = 1,
    };
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    generateB.IsEnabled = false;
    _stepTimer.Start();
    _drawTimer.Start();
}

private (double, double, double, int, int) ParseInitialValues(out bool success)
{
    try
    {
        var a = double.Parse(aT.Text.Trim().Replace('.', ','));
        var b = double.Parse(bT.Text.Trim().Replace('.', ','));
        var p = double.Parse(pT.Text.Trim().Replace('.', ','));
        var pointsCount = int.Parse(pointsT.Text);
        var antsCount = int.Parse(antsT.Text);
        success = true;
        return (a, b, p, pointsCount, antsCount);
    }
    catch (Exception)
    {
        success = false;
        return default;
    }
}

private void generateB_Click(object sender, RoutedEventArgs e)
{
    var (a, b, p, points, ants) = ParseInitialValues(out var success);
    if (!success)
    {
        output.Text = "Values are not in a correct format!";
        return;
    }

    var conf = new AlgorithmConfiguration(a, b, p, points, ants);
    var validationResult = _validation.Validate(conf);
    if (validationResult.IsValid)
    {
        _search = new(new RandomService(), conf);
        ReDrawCadr();
        generateB.IsEnabled = false;
        startB.IsEnabled = true;
        stopB.IsEnabled = true;
        output.Text = "";
    }
    else
    {
        output.Text = string.Join("\n", validationResult.Errors.Select(x => x.ErrorMessage));
    }
}

private void Button_Click_1(object sender, RoutedEventArgs e)
{
    generateB.IsEnabled = true;
    _drawTimer.Stop();
    _stepTimer.Stop();
}
```

```
}
```

Search.cs

```
using pa4.Algorithm.Settings;
namespace pa4.Algorithm;

public class Search
{
    private readonly List<Point> _points = new();
    private readonly double[,] _distances;
    private readonly double[,] _pheramones;
    private readonly double Lmin = 0;
    private readonly AlgorithmConfiguration _configuration;
    private readonly IRandom _random;

    public Search(IRandom random, AlgorithmConfiguration configuration)
    {
        _random = random;
        _configuration = configuration;
        _points = GeneratePoints();
        _distances = new double[_configuration.PointsCount, _configuration.PointsCount];
        _pheramones = new double[_configuration.PointsCount, _configuration.PointsCount];
        GenerateInitialPheramones();
        Lmin = GetLMin();
    }

    public List<Point> Points => _points;
    public double[,] Pheramones => _pheramones;

    private List<Point> GeneratePoints()
    {
        var points = new List<Point>();
        for (var i = 0; i < _configuration.PointsCount; i++)
        {
            var pos = _random.GetRandomPoint();
            points.Add(pos);
        }

        return points;
    }

    private void GenerateInitialPheramones()
    {
        for (var i = 0; i < _points.Count; i++)
        {
            for (var j = 0; j < _configuration.PointsCount; j++)
            {
                if (i == j)
                {
                    _distances[i, j] = 0;
                    _pheramones[i, j] = 0d;
                    continue;
                }

                var dx = _points[i].X - _points[j].X;
                var dy = _points[i].Y - _points[j].Y;
                var dist = Math.Sqrt(dx * dx + dy * dy);
                _distances[i, j] = dist;
                _pheramones[i, j] = 0.1;
            }
        }
    }

    private double GetLMin()
    {
        double lmin = 0;
        var now = 0;
        for (var i = 0; i < _configuration.PointsCount - 1; i++)
```

```
    {
        var minDist = int.MaxValue;
        var minIdx = 0;
        for (var j = 1; j < _configuration.PointsCount; j++)
        {
            if (_points[j].LMin != 0)
            {
                continue;
            }

            var dx = _points[now].X - _points[j].X;
            var dy = _points[now].Y - _points[j].Y;
            var dist = dx * dx + dy * dy;
            if (dist < minDist)
            {
                minDist = dist;
                minIdx = j;
            }
        }

        lmin += Math.Sqrt(minDist);
        now = minIdx;
        _points[now].LMin = Lmin;
    }

    return lmin;
}

public void UpdateRoades(int iterationsToStop)
{
    for (int i = 0; i < iterationsToStop; i++)
    {
        UpdateRoades();
    }
}

public void UpdateRoades()
{
    var pheramoneList = new double[_configuration.PointsCount, _configuration.PointsCount];
    for (var i = 0; i < _configuration.AntsCount; i++)
    {
        var startPos = _random.Next(0, _configuration.PointsCount);
        var curPos = startPos;
        var visited = new bool[_configuration.PointsCount];
        visited[startPos] = true;

        for (var k = 1; k <= _configuration.PointsCount; k++)
        {
            var newPos = 0;
            if (k < _configuration.PointsCount)
            {
                var chances = GetPheramonesChances(curPos, visited);
                var chancesSum = chances.Sum();
                var chanceRanges = GetChancesRanges(chances, chancesSum);
                var random = _random.NextDouble();
                newPos = chanceRanges
                        .TakeWhile(x => newPos < _configuration.PointsCount && random > x)
                        .Count() - 1;
            }
            else
            {
                newPos = startPos;
            }

            var add = Lmin / _distances[curPos, newPos];
            pheramoneList[curPos, newPos] += add;
            pheramoneList[newPos, curPos] += add;
            visited[newPos] = true;
            curPos = newPos;
        }
    }
```

```
            UpdatePheramones(pheramoneList);
        }

        private void UpdatePheramones(double[,] pheramoneList)
        {
            for (var i = 0; i < _configuration.PointsCount; i++)
            {
                for (var j = 0; j < _configuration.PointsCount; j++)
                {
                    if (i == j)
                    {
                        continue;
                    }

                    _pheramones[i, j] = _pheramones[i, j] * (1 - _configuration.P) + pheramoneList[i, j];
                }
            }
        }

        private List<double> GetChancesRanges(List<double> chances, double chancesSum)
        {
            var chanceRanges = new List<double> { 0 };
            for (var j = 1; j < _configuration.PointsCount; j++)
            {
                chances[j] /= chancesSum;
                chanceRanges.Add(chanceRanges[j - 1] + chances[j - 1]);
            }

            return chanceRanges;
        }

        private List<double> GetPheramonesChances(int curPos, bool[] visited)
        {
            var chances = new List<double>();
            for (var j = 0; j < _configuration.PointsCount; j++)
            {
                if (visited[j])
                {
                    chances.Add(0);
                    continue;
                }

                if (_pheramones[curPos, j] == 0 || _distances[curPos, j] == 0)
                {
                    chances.Add(0);
                    continue;
                }

                 var chance = Math.Pow(_pheramones[curPos, j], _configuration.A)
                    * (1 / Math.Pow(_distances[curPos, j], _configuration.B));
                chances.Add(chance);
            }

            return chances;
        }
    }
```

Point.cs

```
namespace pa4.Algorithm;

public class Point
{
    public Point(int x, int y, double lMin)
    {
        X = x;
        Y = y;
        LMin = lMin;
    }

    public int X { get; set; }
```

```
    public int Y { get; set; }
    public double LMin { get; set; }

    public override string ToString()
    {
        return X + " " + Y + "\n";
    }
}
```

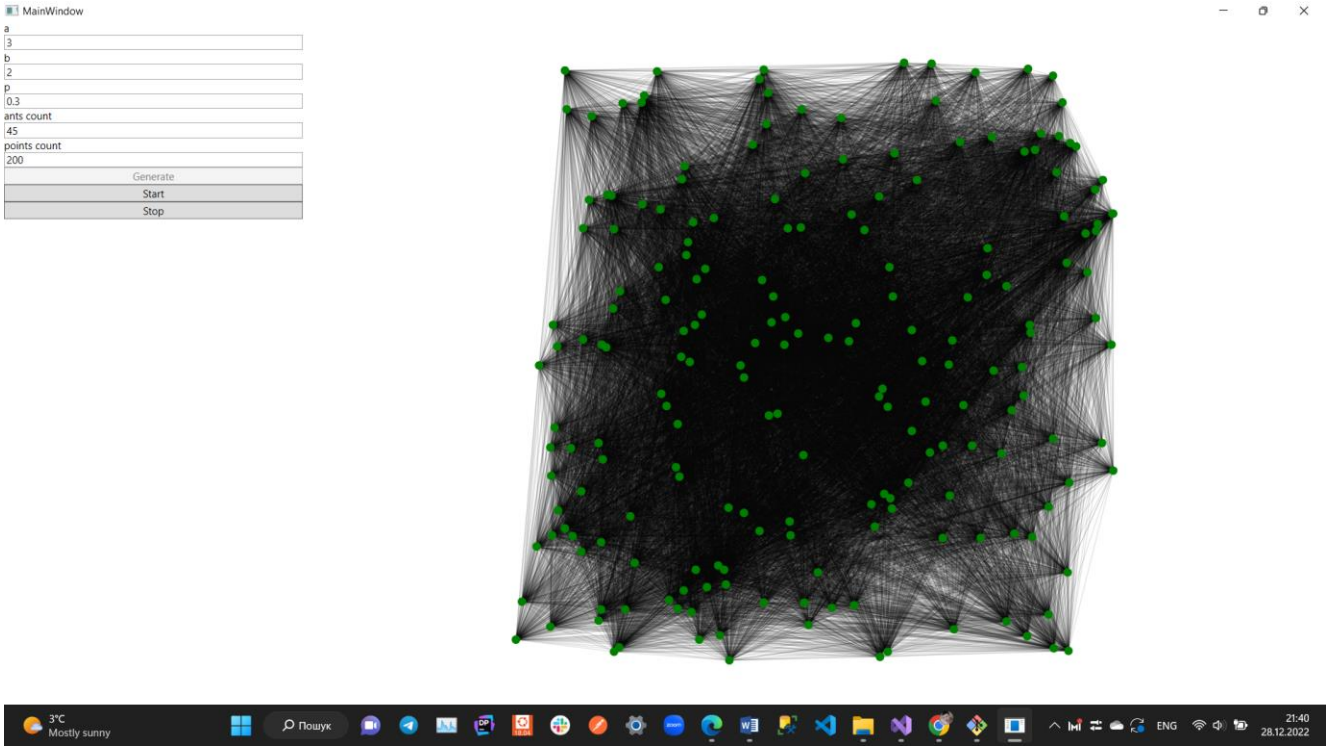
DoubleEquiltyCommparer.cs

```
using System.Diagnostics.CodeAnalysis;

namespace pa4.Algorithm;

public class DoubleEqualityComparer : IEqualityComparer<double>
{
    const double EPSILON = 0.00000001;
    public bool Equals(double x, double y)
    {
        var diff = Math.Abs(x - y);
        if (diff <= EPSILON)
        {
            return true;
        }

        return false;
    }

    public int GetHashCode([DisallowNull] double obj)
    {
        return obj.GetHashCode();
    }
}using FluentValidation;
using pa4.Algorithm.Settings;

namespace pa4;

public class ConfigurationValidation : AbstractValidator<AlgorithmConfiguration>
{
    public ConfigurationValidation()
    {
        RuleFor(x => x.AntsCount)
            .GreaterThan(0)
            .LessThanOrEqualTo(50)
            .WithMessage($"Ants count must be in the range: from 0 to 50");

        RuleFor(x => x.PointsCount)
            .GreaterThan(0)
            .LessThanOrEqualTo(200)
            .WithMessage($"Points count must be in the range: from 0 to 200");

        RuleFor(x => x.P)
            .GreaterThanOrEqualTo(0)
            .LessThan(5)
            .WithMessage($"Ro must be in the range: from 0 to 5");

        RuleFor(x => x.B)
           .GreaterThanOrEqualTo(0)
           .LessThan(5)
           .WithMessage($"Beta must be in the range: from 0 to 5");

        RuleFor(x => x.A)
           .GreaterThanOrEqualTo(0)
           .LessThan(5)
           .WithMessage($"Alpha must be in the range: from 0 to 5");
    }
}namespace pa4.Algorithm.Settings;

public static class DefaultOptions
{
    public const double A = 3;
    public const double B = 2;
    public const double P = 0.3;
    public const int AntsCount = 2;
    public const int PointsCount = 3;
}namespace pa4.Algorithm.Settings;

public class AlgorithmConfiguration
{
    public AlgorithmConfiguration(double a, double b, double p, int pointsCount, int antsCount)
    {
```

```
        A = a;
        B = b;
        P = p;
        PointsCount = pointsCount;
        AntsCount = antsCount;
    }

    public AlgorithmConfiguration(int pointsCount, int antsCount)
    {
        AntsCount = antsCount;
        PointsCount = pointsCount;
        A = DefaultOptions.A;
        B = DefaultOptions.B;
        P = DefaultOptions.P;
    }

    public AlgorithmConfiguration()
    {
        A = DefaultOptions.A;
        B = DefaultOptions.B;
        P = DefaultOptions.P;
        PointsCount = DefaultOptions.PointsCount;
        AntsCount = DefaultOptions.AntsCount;
    }

    public double A { get; }
    public double B { get; }
    public double P { get; }
    public int PointsCount { get; }
    public int AntsCount { get; }
}           namespace pa4.Algorithm;

public class RandomService : IRandom
{
    private readonly Random _random = new();
    public Point GetRandomPoint() => new(_random.Next(0, 700), _random.Next(0, 700), 0);

    public int Next(int min, int max) => _random.Next(min, max);

    public double NextDouble() => _random.NextDouble();
}namespace pa4.Algorithm;

public interface IRandom
{
    int Next(int min, int max);
    Point GetRandomPoint();
    double NextDouble();
}namespace pa4.Algorithm.Extentions;

public static class DoubleArrayExtentions
{
    public static List<double> ToList(this double[,] arr)
    {
        var list = new List<double>();
        for (int i = 0; i < arr.GetLength(0); i++)
        {
            for (int j = 0; j < arr.GetLength(1); j++)
            {
                list.Add(arr[i, j]);
            }
        }

        return list;
    }
}
```

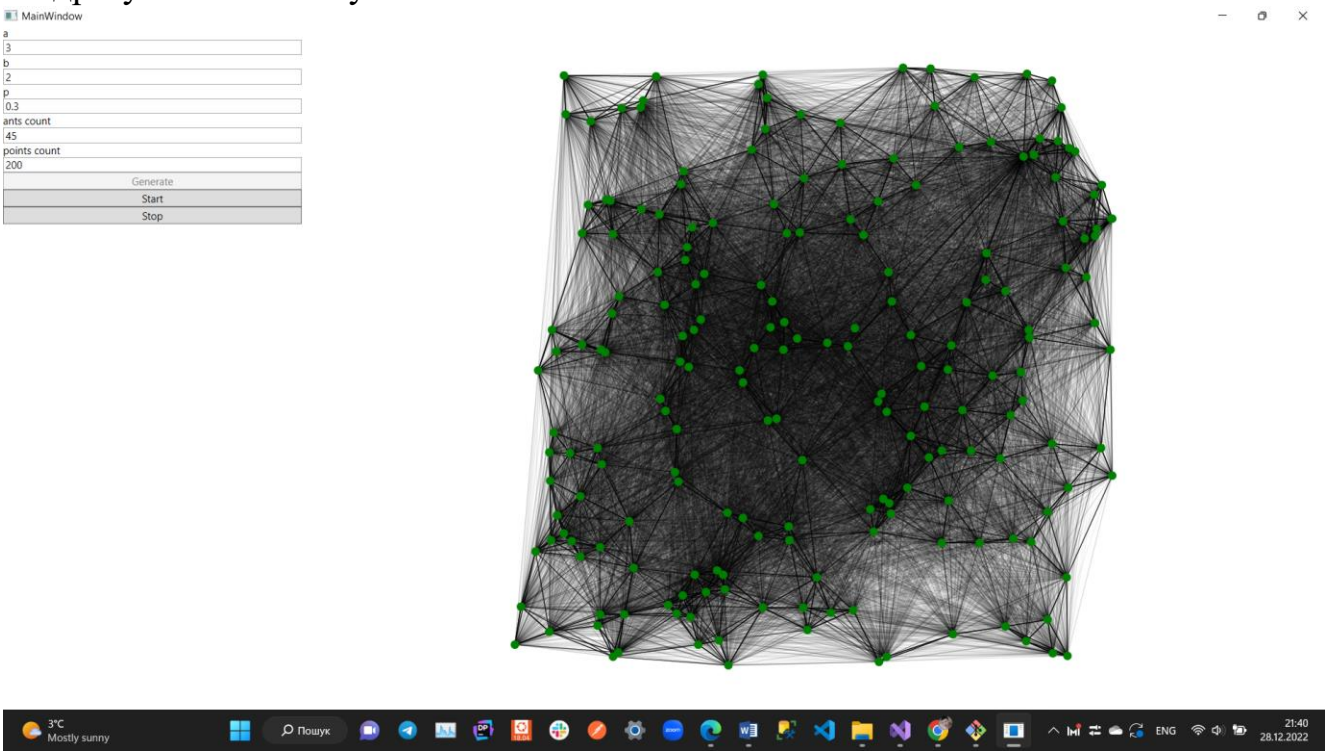
2. Виконання

До початку:

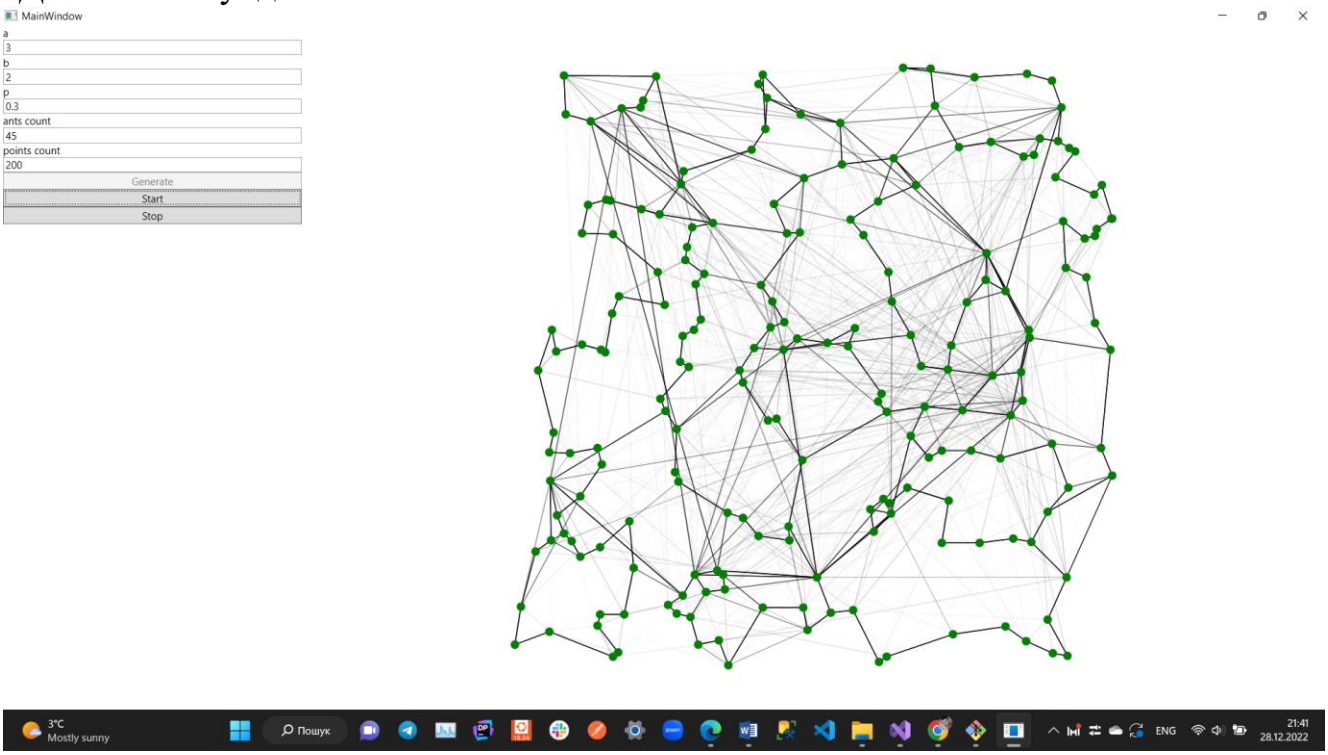

MainWindow
a
3
b
2
p
0.3
ants count
45
points count
200
Generate
Start
Stop

3°C
Mostly sunny
Пошук
ENG
21:40
28.12.2022

Відразу після початку:



Декілька секунд пізніше:

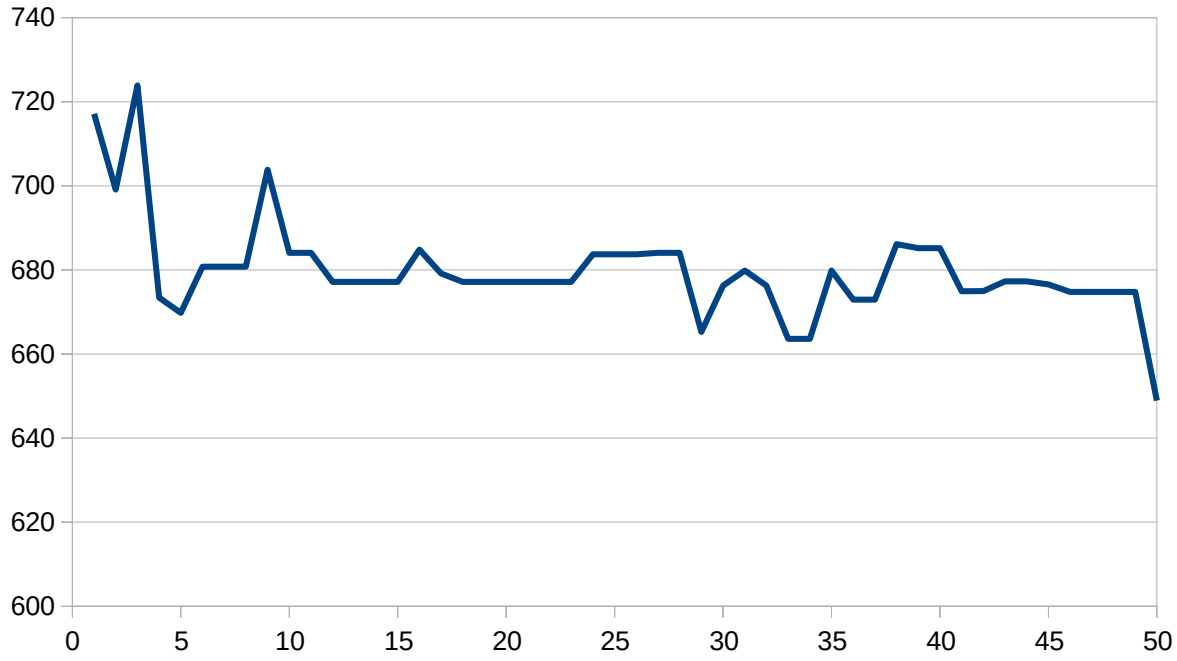


2. Тестування алгоритмів
    1. Значення цільової функції із збільшенням кількості ітерацій

Кількість ітерацій	Довжина шляху
20	717,12
40	699,12
60	723,89
80	673,4
100	669,77
120	680,77
140	680,77
160	680,77
180	703,83
200	684,07
220	684,07
240	677,15
260	677,15
280	677,15
300	677,15
320	684,82
340	679,16
360	677,15
380	677,15
400	677,15
420	677,15
440	677,15
460	677,15
480	683,72
500	683,72
520	683,72
540	684,07
560	684,07
580	665,27
600	676,24
620	679,85
640	676,24
660	663,59
680	663,59
700	679,85
720	672,93
740	672,93
760	686,12
780	685,17
800	685,17
820	674,94
840	674,98
860	677,3
880	677,3
900	676,55
920	674,77

940	674,77
960	674,77
980	674,77
1000	648,92

2. Графіки залежності розв’язку від числа ітерацій



4. ВИСНОВОК

При виконанні лабораторної роботи було ознайомлено з різними метаевристичними алгоритмами, а також створена реалізація на мові програмування c# алгоритму комівояжера на основі метаевристичного мурашиного алгоритму, за допомогою якої було проведено дослідження роботи цього алгоритму. Результати було записано в таблицю в пункті 3.2.1 та на основі цих даних побудовано графік в пункті 3.2.2.

5. КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 23.10.2022 включно максимальний бал дорівнює – 5. Після 23.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 10%;
- програмна реалізація алгоритму – 60%;
- дослідження алгоритмів – 25%;
- висновок – 5%.