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-13 Жмайло Дмитро О.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О.О.* _____
(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.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача і алгоритм**
9	Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники).

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

Program.cs

```
using System.Diagnostics;

namespace lab4PA;

class Program
{
    public static void Main(string[] args)
    {
        const int vertexCount = 150;

        Console.WriteLine("Welcome to my 4th lab! (9 variant)");
        Graph graph = new Graph(new int[vertexCount, vertexCount]);
        Console.Write("Is adjacency matrix validate? - ");
        if (graph.IsAdjMatrixValid())
            Console.WriteLine("Yes! :)");
        else
            Console.WriteLine("No :'(");

        graph.ShowMatrix();
        Console.WriteLine("Degrees of graph: ");
        graph.ShowArr(graph.GetDegreesArr());

        Console.WriteLine("Training is started successfully, stand by...");
        Stopwatch sw = Stopwatch.StartNew();

        graph = new ABCAlgorithm(graph).TrainAlgorithm();
        sw.Stop();
        Console.WriteLine($"Estimated time of training:
{sw.ElapsedMilliseconds/1000}s");


        Console.WriteLine("Final colored graph: ");
        graph.ShowArr(graph.GetColorsArr());

        Console.Write($"Is graph colored properly? - ");
        if (graph.IsGraphValidColored())
            Console.WriteLine("Yes! :)");
        else
            Console.WriteLine("No :'(");
    }
      }
```

Graph.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab4PA
{
    internal class Graph
    {
        private int[,] adjMatrix;
        private int[] colorsArr;

        const int noColor = -1;
        const int vertexCount = 150;
        const int minVertexDegree = 1;
        const int maxVertexDegree = 30;

        public Graph(Graph g)
        {
            this.adjMatrix = new int[g.adjMatrix.GetLength(0),
g.adjMatrix.GetLength(1)];
            this.colorsArr = new int[g.colorsArr.Length];

            Array.Copy(g.adjMatrix, this.adjMatrix, g.adjMatrix.Length);
            Array.Copy(g.colorsArr, this.colorsArr, g.colorsArr.Length);
            /*
            this.adjMatrix = g.adjMatrix;
            this.colors = g.colors;
            */
        }

        public Graph(int[,] adjMatrix)
        {
            Random rand = new Random();
            this.adjMatrix = adjMatrix;
            this.colorsArr = new int[adjMatrix.GetLength(0)];
            Array.Fill(this.colorsArr, noColor);

            for (int currV = 0; currV < vertexCount; ++currV)
            {
                int finalVertexDegree = Math.Min(rand.Next(minVertexDegree,
maxVertexDegree + 1)
                    - GetVertexDegree(currV), vertexCount - currV - 1);
                for (int newConnections = 0; newConnections < finalVertexDegree;
++newConnections)
                {
                    bool isConnectedAlready = true;
                    for (int tryCount = 0, newVertex = rand.Next(currV+1,
vertexCount);
                        isConnectedAlready && tryCount < vertexCount; ++tryCount,
newVertex=rand.Next(currV+1,vertexCount))
                    {
                        if (this.adjMatrix[currV,newVertex] == 0 &&
GetVertexDegree(newVertex) < maxVertexDegree)
                        {
                            isConnectedAlready = false;
                            this.adjMatrix[currV, newVertex] = 1;
                            this.adjMatrix[newVertex, currV] = 1;
                        }
                    }
                }
            }
```

```
}

public bool IsAdjMatrixValid()
{
    for (int vertex = 0; vertex < adjMatrix.GetLength(0); vertex++)
    {
        if (GetVertexDegree(vertex) > maxVertexDegree)
        {
            return false;
        }
    }
    return true;
}

public bool IsGraphValidColored()
{
    for (int i = 0; i < colorsArr.Length; i++)
    {
        if (colorsArr[i] == noColor)
        {
            return false;
        }
    }
    if (IsColoringValid())
    {
        return true;
    }
    return false;
}

public int[] GetColorsArr()
{
    return colorsArr;
}

public int[] GetDegreesArr()
{
    int[] degrees = new int[adjMatrix.GetLength(0)];
    for (int i = 0; i < degrees.Length; ++i)
    {
        degrees[i] = GetVertexDegree(i);
    }
    return degrees;
}

public int GetVertexDegree(int vertex)
{
    int sum = 0;
    for (int i = 0; i < adjMatrix.GetLength(0); i++)
    {
        sum += adjMatrix[vertex, i];
    }
    return sum;
}

public int[] GetConnectedVertexes(int vertex)
{
    int[] connectedVertexes = new int[GetVertexDegree(vertex)];
    for (int i = 0, k = -1; i < adjMatrix.GetLength(0); ++i)
    {
        if (adjMatrix[vertex, i] == 1)
        {
            connectedVertexes[++k] = i;
        }
```

```
            }
            return connectedVertexes;
        }

        public bool TryToColorIfValid(int vertex, int newColor)
        {
            int oldColor = colorsArr[vertex];
            colorsArr[vertex] = newColor;
            bool isValid = IsColoringValid();
            if (!isValid)
            {
                colorsArr[vertex] = oldColor;
            }
            return isValid;
        }

        private bool IsColoringValid()
        {
            for (int row = 0; row < adjMatrix.GetLength(0); ++row)
            {
                for (int col = 0; col < adjMatrix.GetLength(1); ++col)
                {
                    if (adjMatrix[row, col] == 1 && colorsArr[row] != noColor &&
colorsArr[row] == colorsArr[col])
                    {
                        return false;
                    }
                }
            }
            return true;
        }


        public void ShowMatrix()
        {
            for (int i = 0; i < adjMatrix.GetLength(0); i++)
            {
                for (int j = 0; j < adjMatrix.GetLength(1); j++)
                {
                    Console.Write(adjMatrix[i, j] + " ");
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }

        public void ShowArr(int[] arr)
        {
            const int maxRowLength = 10;
            for (int i = 0; i < arr.Length; i++)
            {
                Console.Write(arr[i] + "\t");
                if (i + 1 % maxRowLength == 0)
                {
                    Console.WriteLine();
                }
            }
            Console.WriteLine();
            Console.WriteLine();
        }

        public static List<int> GetVertexList()
        {
            List<int> list = new List<int>();
            for (int i = 0; i < vertexCount; i++)
```

```
            {
                list.Add(i);
            }
            return list;
        }
    }
}
```

ABCAlgorithm

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab4PA
{
    internal class ABCAlgorithm
    {
        private Graph initialGraph;
        private Graph graph;
        private List<int> availableVertices;
        private int[] paletteArr;
        private List<int> usedColorsList;

        const int explorerBeesCount = 3;
        const int totalBeesCount = 25;
        const int maxVertexDegree = 30;
        const int iterationsPerStep = 20;
        const int maxIterationsCount = 1000;

        public ABCAlgorithm(Graph initialGraph)
        {
            this.initialGraph = initialGraph;
            graph = new Graph(initialGraph);
            availableVertices = Graph.GetVertexList();
            paletteArr = new int[maxVertexDegree + 1];
            for (int i = 0; i < maxVertexDegree + 1; i++)
            {
                paletteArr[i] = i;
            }
            usedColorsList = new List<int>();
        }

        public Graph TrainAlgorithm()
        {
            Graph resultGraph = new Graph(graph);
            int bestChromNumber = CalculateChromNumber();
            Console.WriteLine("Init colored graph: ");
            graph.ShowArr(graph.GetColorsArr());
            Console.WriteLine($"The new best solution of the graph found on 0
iteration, old = {maxVertexDegree + 1}," +
            $" new = {bestChromNumber}, estimated time = 0 s");
            resetAlgorithm();

            for (int iterations = 0; iterations < maxIterationsCount;)
            {
                Stopwatch sw = Stopwatch.StartNew();
                for (int k = 1; k < iterationsPerStep + 1; k++, resetAlgorithm())
                {
                    int newChromNumber = CalculateChromNumber();
                    if (newChromNumber < bestChromNumber)
```

```
                    {
                        Console.WriteLine($"The new best solution of the graph found
on {iterations + k} " +
                        $"iteration, old = {bestChromNumber}," + $" new =
{bestChromNumber = newChromNumber}, " +
                        $"estimated time = {sw.ElapsedMilliseconds / 1000} s");
                        resultGraph = new Graph(graph);
                    }
                }
                Console.WriteLine($"Iteration {iterations += iterationsPerStep}, best
result = " +
                    $"{bestChromNumber}, estimated time = {sw.ElapsedMilliseconds /
1000} s");
            }
            Console.WriteLine("Initial colors of graph are (-1 - no color): ");
            graph.ShowArr(graph.GetColorsArr());
            return resultGraph;
        }

        public void resetAlgorithm()
        {
            usedColorsList.Clear();
            availableVertices = Graph.GetVertexList();
            graph = new Graph(initialGraph);
        }

        public int CalculateChromNumber()
        {
            while (!graph.IsGraphValidColored())
            {
                List<int> selectedVertices = SendEmployeedBees();
                SendOnlookerBees(selectedVertices);
            }
            return usedColorsList.Count();
        }

        private List<int> SendEmployeedBees()
        {
            List<int> selectedVertices = new List<int>();
            for (int employeedBee = 0; employeedBee < explorerBeesCount;
++employeedBee)
            {
                if (availableVertices.Count == 0)
                {
                    continue;
                }
                int index = new Random().Next(availableVertices.Count);
                int randomSelectedVertex = availableVertices[index];
                availableVertices.RemoveAt(index);
                selectedVertices.Add(randomSelectedVertex);
            }
            return selectedVertices;
        }

        private void SendOnlookerBees(List<int> selectedVertices)
        {
            int[] selectedVerticesDegrees = new int[selectedVertices.Count];
            for (int i = 0; i < selectedVerticesDegrees.Length; ++i)
            {
                selectedVerticesDegrees[i] =
graph.GetVertexDegree(selectedVertices[i]);
            }
            int[] onlookerBeesSplit = GetOnlookersBeesSplit(selectedVerticesDegrees);
            for (int i = 0; i < selectedVertices.Count; ++i)
            {
```

```
                int onlookerBeesCountForVertex = onlookerBeesSplit[i];
                int[] connectedVertices =
graph.GetConnectedVertexes(selectedVertices[i]);
                ColorConnectedVertex(connectedVertices, onlookerBeesCountForVertex);
                ColorVertex(selectedVertices[i]);
            }
        }
        private int[] GetOnlookersBeesSplit(int[] selectedVerticesDegrees)
        {
            double[] nectarValues = GetNectarValues(selectedVerticesDegrees);
            int onlookerBeesCount = totalBeesCount - explorerBeesCount;
            int[] res = new int[nectarValues.Length];
            for (int i = 0; i < nectarValues.Length; ++i)
            {
                res[i] = (int)(onlookerBeesCount * nectarValues[i]);
                onlookerBeesCount -= res[i];
            }
            return res;
        }

        private double[] GetNectarValues(int[] selectedVerticesDegrees)
        {
            double[] res = new double[selectedVerticesDegrees.Length];
            for (int i = 0, totalDegrees = selectedVerticesDegrees.Sum(); i <
selectedVerticesDegrees.Length; ++i)
            {
                res[i] = (double)selectedVerticesDegrees[i] / totalDegrees;
            }
            return res;
        }
        private void ColorConnectedVertex(int[] connectedVertices, int
onlookerBeesCount)
        {
            for (int i = 0; i < connectedVertices.Length; ++i)
            {
                if (i < onlookerBeesCount - 1)
                {
                    ColorVertex(connectedVertices[i]);
                }
            }
        }
        private void ColorVertex(int vertex)
        {
            List<int> availableColorsList = new List<int>(usedColorsList);
            bool isColoredSuccessfully = false;
            while (!isColoredSuccessfully)
            {
                if (availableColorsList.Count == 0)
                {
                    int newColor = paletteArr[usedColorsList.Count];
                    usedColorsList.Add(newColor);
                    graph.TryToColorIfValid(vertex, newColor);
                    break;
                }
                int index = new Random().Next(availableColorsList.Count);
                int color = availableColorsList[index];
                availableColorsList.RemoveAt(index);
                isColoredSuccessfully = graph.TryToColorIfValid(vertex, color);
            }
        }


    }
}
```

3.1.2 Приклади роботи

На рисунках 3.1, 3.2 і 3.3 показані приклади роботи програми.

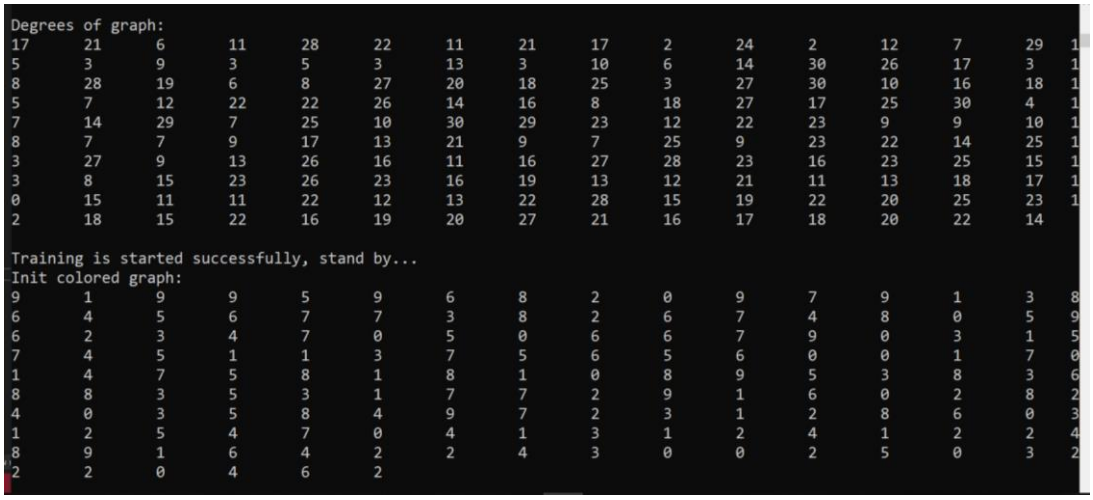

Рисунок 3.1 – Початок тренування алгоритму

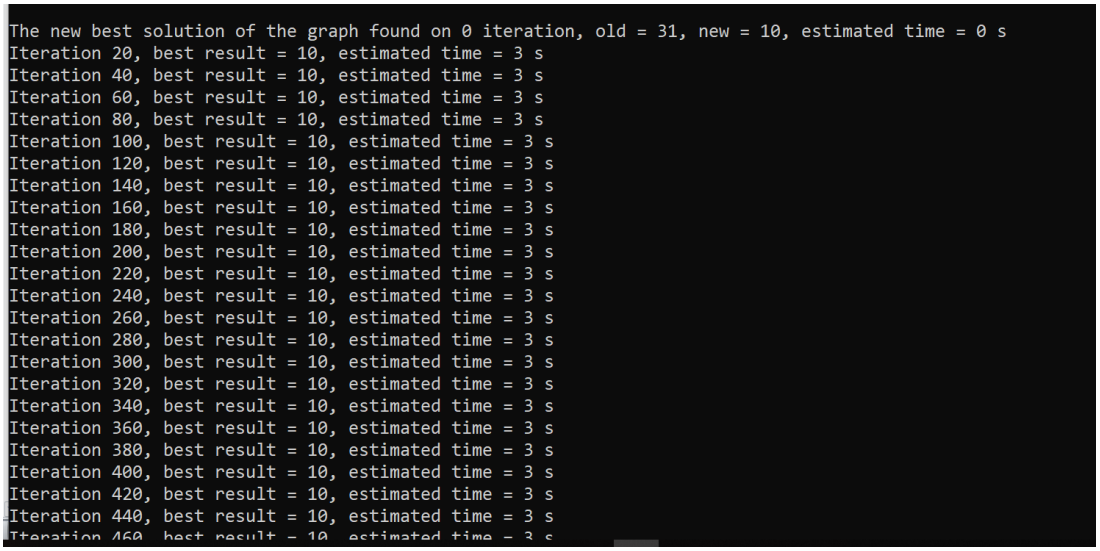

Рисунок 3.2 – Процес тренування алгоритму

```
Initial colors of graph are (-1 - no color):
-1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -
1       -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1      -1

Estimated time of training: 9s
Final colored graph:
9       7       5       4       6       10      0       3       3       7       2       6       0       8       9       5
9       9       6       10      7       7       0       4       3       5       2       0       8       0       1       9
2       6       9       3       0       8       0       5       6       3       7       4       7       9       5       4
0       7       1       5       6       4       5       1       5       6       7       1       9       0       5       1
3       8       9       8       8       0       1       1       5       8       8       1       2       8       1       2
1       6       6       4       7       2       1       6       4       2       2       10      5       2       3       4
1       8       2       5       6       6       7       6       7       0       8       8       2       5       6       5
1       4       4       8       0       5       8       9       6       3       4       3       2       0       1       3
4       0       5       4       6       2       3       7       2       2       8       3       5       4       1       5
2       3       9       4       3       10

Is graph colored properly? - Yes! :)
```

Рисунок 3.3 – Результат виконання алгоритму

Тестування алгоритму

3.1.3 Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

Ітерації	**Хром. число**	Ітерації	**Хром. число**	Ітерації	**Хром. число**	Ітерації	**Хром. число**	Ітерації	**Хром. число**
0	**31**	220	**10**	440	**9**	660	**9**	880	**9**
20	**12**	240	**10**	460	**9**	680	**9**	900	**9**
40	**11**	260	**10**	480	**9**	700	**9**	920	**9**
60	**11**	280	**10**	500	**9**	720	**9**	940	**9**
80	**11**	300	**9**	520	**9**	740	**9**	960	**9**
100	**10**	320	**9**	540	**9**	760	**9**	980	**9**
120	**10**	340	**9**	560	**9**	780	**9**	1000	**9**
140	**10**	360	**9**	580	**9**	800	**9**	…	**…**
160	**10**	380	**9**	600	**9**	820	**9**	…	**…**
180	**10**	400	**9**	620	**9**	840	**9**	…	**…**
200	**10**	420	**9**	640	**9**	860	**9**	…	**…**

3.1.4 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

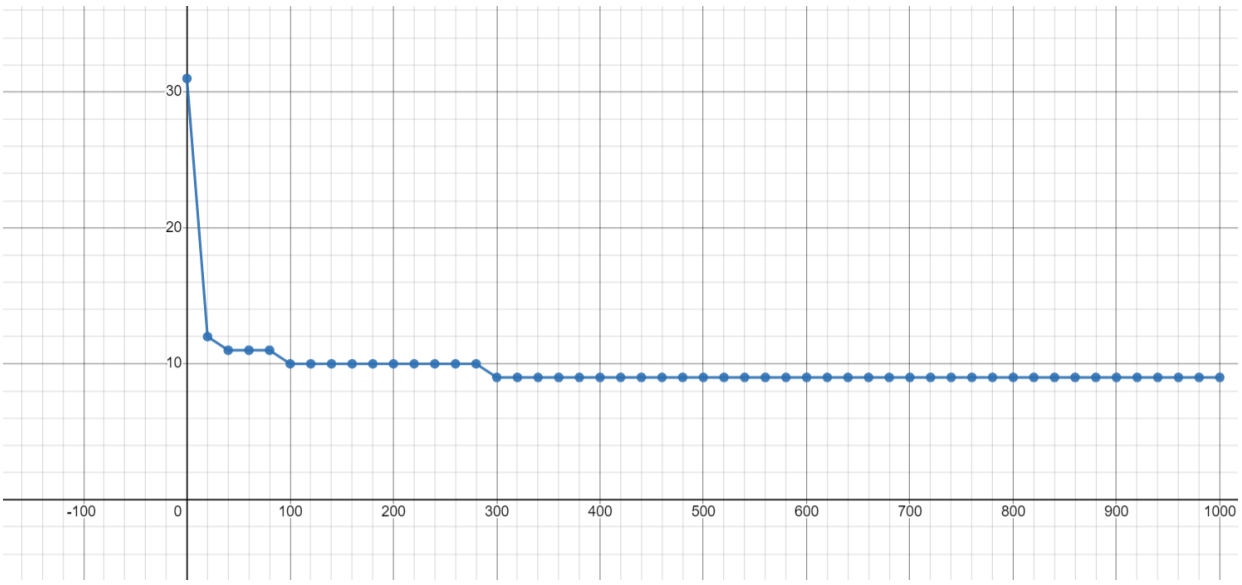

Рисунок 3.3 – Графік залежності хроматичного числа від числа ітерацій (розроблений у застосунку Desmos)

ВИСНОВОК

Виконуючи дану лабораторну роботу, я розв’язував задачу розфарбування графу на 150 вершин, у кожної з яких степінь від 1 до 30 включно. Виконав реалізацію бджолиного алгоритму, використовуючи мову програмування C#. Для розв'язку використав модифікацію бджолиного алгоритму – штучну бджолину колонію. Відповідно до варіанту, використовувалися 25 бджіл, з яких 3 були розвідниками.

Побудував графік залежності хроматичного числа від кількості ітерацій та підмітив, що після 280 ітерацій хроматичне число залишається без змін, і, відповідно, сенсу продовжувати виконання алгоритму після цієї кількості ітерація при заданих параметрах немає.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація алгоритму – 75%;
- тестування алгоритму– 20%;
- висновок – 5%.