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 Кисельов Микита* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(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**
12	Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники).

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

GCP_ABC_ABC.java:

```
public class GCP_ABC_ABC {
    public static void main(String[] args) {
        Graph graph = new Graph(new
int[constants.VERTEX_COUNT][constants.VERTEX_COUNT]);
        System.out.printf("Adjacency matrix is validate - %b\n",
graph.validateAdjMatrix());
        graph.printAdjMatrix();

        System.out.println("Degrees of graph: ");
        graph.printArrayByUnits(graph.getDegrees());

        System.out.println("Start solving by ABC(ABC) algorithm...");
        long start = System.currentTimeMillis();
        graph = new ABCAlgorithm(graph).train();
        System.out.printf("Estimated time: %d seconds...\n",
(System.currentTimeMillis() - start) / 1000);

        System.out.println("Final colored graph:");
        graph.printArrayByUnits(graph.getColors());
        System.out.printf("Was graph colored valid? - %b",
graph.isAllVerticesValidColored());
    }
}
```

ABCAlgorithm.java:

```
import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.stream.IntStream;

public class ABCAlgorithm {
    private final Graph initialGraph;
    private Graph graph;
    private ArrayList<Integer> availableVertices;
    private final int[] palette;
    private final ArrayList<Integer> usedColors;

    public int calculateChromaticNumber() {
        while (!graph.isAllVerticesValidColored()) {
            ArrayList<Integer> selectedVertices = sendEmployedBees();
            sendOnlookerBees(selectedVertices);
        }
        return usedColors.size();
    }

    public ABCAlgorithm(Graph initialGraph) {
        this.initialGraph = initialGraph;
        graph = new Graph(initialGraph);
        availableVertices = Graph.getVertexArray();
        palette = IntStream.rangeClosed(0,
```

```
constants.MAX_VERTEX_DEGREE).toArray();
        usedColors = new ArrayList<>();
    }

    public void resetAlgorithm() {
        usedColors.clear();
        availableVertices = Graph.getVertexArray();
        graph = new Graph(initialGraph);
    }

    private @NotNull ArrayList<Integer> sendEmployedBees() {
        ArrayList<Integer> selectedVertices = new ArrayList<>(List.of(0));
        for (int employedBee = 0; employedBee < constants.EXPLORER_BEES_COUNT;
++employedBee) {
            int randomSelectedVertex = availableVertices.get(new
Random().nextInt(availableVertices.size()));
            availableVertices.remove((Object)randomSelectedVertex);
            selectedVertices.add(randomSelectedVertex);
        }
        return selectedVertices;
    }

    private void sendOnlookerBees(@NotNull ArrayList<Integer> selectedVertices)
{
        int[] selectedVerticesDegrees = new int[selectedVertices.size()];
        for (int i = 0; i < selectedVerticesDegrees.length; ++i)
            selectedVerticesDegrees[i] =
graph.getVertexDegree(selectedVertices.get(i));
        int[] onlookerBeesSplit = getOnlookerBeesSplit(selectedVerticesDegrees);
        for (int i = 0; i < selectedVertices.size(); ++i) {
            int onlookerBeesCountForVertex = onlookerBeesSplit[i];
            int[] connectedVertices =
graph.getConnectedVertexes(selectedVertices.get(i));
            colorConnectedVertices(connectedVertices,
onlookerBeesCountForVertex);
            colorVertex(selectedVertices.get(i));
        }
    }

    private int @NotNull [] getOnlookerBeesSplit(int[] selectedVerticesDegrees)
{
        double[] nectarValues = getNectarValues(selectedVerticesDegrees);
        int onlookerBeesCount = constants.TOTAL_BEES_COUNT -
constants.EXPLORER_BEES_COUNT;
        int[] res = new int[nectarValues.length];
        for (int i = 0; i < nectarValues.length; ++i)
            onlookerBeesCount -= res[i] = (int)(onlookerBeesCount *
nectarValues[i]);
        return res;
    }

    private double @NotNull [] getNectarValues(int[] selectedVerticesDegrees) {
        double[] res = new double[selectedVerticesDegrees.length];
        for (int i = 0, totalDegrees =
IntStream.of(selectedVerticesDegrees).sum(); i < selectedVerticesDegrees.length;
++i)
            res[i] = (double)selectedVerticesDegrees[i] / totalDegrees;
        return res;
    }

    private void colorConnectedVertices(int @NotNull [] connectedVertices, int
onlookerBeesCount) {
        for (int i = 0; i < connectedVertices.length; ++i)
            if (i < onlookerBeesCount - 1) colorVertex(connectedVertices[i]);
```

```
    }

    private void colorVertex(int vertex) {
        ArrayList<Integer> availableColors = new ArrayList<>(usedColors);
        boolean isColoredSuccessfully = false;
        while (!isColoredSuccessfully) {
            if (availableColors.isEmpty()) {
                int newColor = palette[usedColors.size()];
                usedColors.add(newColor);
                graph.tryToColorAndCheckIsValid(vertex, newColor);
                break;
            }
            int color = availableColors.get(new
Random().nextInt(availableColors.size()));
            availableColors.remove((Object)color);
            isColoredSuccessfully = graph.tryToColorAndCheckIsValid(vertex,
color);
        }
    }

    public Graph train() {
        Graph resGraph = new Graph(graph);
        int bestCN = calculateChromaticNumber();
        System.out.println("Init colored graph:");
        System.out.printf("The new best solution of the graph found on %4d
iteration - old: %3d, new: %3d:, estimated time - %2d seconds\n",
                0, constants.MAX_VERTEX_DEGREE + 1, bestCN, 0);
        graph.printArrayByUnits(graph.getColors());
        resetAlgorithm();
        for (int iteration = 0; iteration < constants.ITERATIONS_COUNT;) {
            long start = System.currentTimeMillis();
            for (int k = 0; k < constants.ITERATIONS_PER_STEP; ++k,
resetAlgorithm()) {
                int newCN = calculateChromaticNumber();
                if (newCN < bestCN) {
                    System.out.printf("New best solution of the graph found on
%4d iteration, old: %3d, new: %3d, estimated time - %2d seconds\n",
                            iteration + k, bestCN, bestCN = newCN,
(System.currentTimeMillis() - start) / 1000);
                    graph.printArrayByUnits(graph.getColors());
                    resGraph = new Graph(graph);
                }
            }
            System.out.printf("On iteration %4d best result is %3d, estimated
time - %2d seconds\n",
                    iteration += constants.ITERATIONS_PER_STEP, bestCN,
(System.currentTimeMillis() - start) / 1000);
        }
        return resGraph;
    }
}
```

Graph.java:

```
import org.jetbrains.annotations.NotNull;

import java.util.*;
import java.util.stream.IntStream;

class Graph {
    private final int[][] adjMatrix;
    private final int[] colors;

    public Graph(@NotNull Graph g) {
```

```
        this.adjMatrix = g.adjMatrix.clone();
        this.colors = g.colors.clone();
    }

    public Graph(int @NotNull [][] adjMatrix) {
        this.adjMatrix = adjMatrix;
        this.colors = new int[adjMatrix.length];
        Arrays.fill(this.colors, constants.NO_COLOR);
        for (int currV = 0; currV < constants.VERTEX_COUNT; ++currV) {
            int finalVertexDegree = Math.min(rand(constants.MIN_VERTEX_DEGREE,
                    constants.MAX_VERTEX_DEGREE + 1) - getVertexDegree(currV),
constants.VERTEX_COUNT - currV - 1);
            for (int newConnection = 0; newConnection < finalVertexDegree;
++newConnection) {
                boolean isConnectedAlready = true;
                for (int tryCount = 0, newVertex = rand(currV + 1,
constants.VERTEX_COUNT);
                     isConnectedAlready && tryCount < constants.VERTEX_COUNT;
                     ++tryCount, newVertex = rand(currV + 1,
constants.VERTEX_COUNT)) {
                    if (this.adjMatrix[currV][newVertex] == 0
                            && getVertexDegree(newVertex) <
constants.MAX_VERTEX_DEGREE) {
                        isConnectedAlready = false;
                        this.adjMatrix[currV][newVertex] = 1;
                        this.adjMatrix[newVertex][currV] = 1;
                    }
                }
            }
        }
    }

    public boolean validateAdjMatrix() {
        for (int vertex = 0; vertex < adjMatrix.length; ++vertex)
            if (getVertexDegree(vertex) > constants.MAX_VERTEX_DEGREE) return
false;
        return true;
    }

    public void printAdjMatrix() {
        System.out.println(Arrays.deepToString(adjMatrix));
    }

    public void printArrayByUnits(int @NotNull [] arr) {
        Arrays.stream(IntStream.range(0, arr.length).toArray()).forEach(i ->
System.out.printf("%4d" +
                ((i+1) % constants.UNIT_SIZE > 0 ? "" : "\n"), arr[i]));
        System.out.println();
    }

    public int[] getColors() {
        return colors;
    }

    public int[] getDegrees() {
        int[] degrees = new int[adjMatrix.length];
        for (int i = 0; i < degrees.length; ++i) degrees[i] =
getVertexDegree(i);
        return degrees;
    }

    public static ArrayList<Integer> getVertexArray() {
        return IntStream.range(0,
constants.VERTEX_COUNT).collect(ArrayList::new, List::add, List::addAll);
```

```
    }

    public int getVertexDegree(int vertex) {
        return IntStream.of(adjMatrix[vertex]).sum();
    }

    public int[] getConnectedVertexes(int vertex) {
        int[] connectedVertexes = new int[getVertexDegree(vertex)];
        for (int i = 0, k = -1; i < adjMatrix[vertex].length; ++i)
            if (adjMatrix[vertex][i] == 1) connectedVertexes[++k] = i;
        return connectedVertexes;
    }

    public boolean isAllVerticesValidColored() {
        return IntStream.of(colors).noneMatch(c -> c == constants.NO_COLOR) &&
isColoringValid();
    }

    public boolean tryToColorAndCheckIsValid(int vertex, int newColor) {
        int oldColor = colors[vertex];
        colors[vertex] = newColor;
        boolean isValid = isColoringValid();
        if (!isValid) colors[vertex] = oldColor;
        return isValid;
    }

    private boolean isColoringValid() {
        for (int row = 0; row < adjMatrix.length; ++row)
            for (int col = 0; col < adjMatrix[row].length; ++col)
                if (adjMatrix[row][col] == 1
                        && colors[row] != constants.NO_COLOR
                        && colors[row] == colors[col]) {
                    return false;
                }
        return true;
    }

    private static int rand(int min, int max) {
        return new Random().nextInt(max - min) + min;
    }
}
```

constants.java:

```
public abstract class constants {
    public static final int
            ITERATIONS_COUNT = 1000,
            ITERATIONS_PER_STEP = 20,
            VERTEX_COUNT = 300,
            MIN_VERTEX_DEGREE = 1,
            MAX_VERTEX_DEGREE = 30,
            TOTAL_BEES_COUNT = 60,
            EXPLORER_BEES_COUNT = 5,
            NO_COLOR = -1,
            UNIT_SIZE = 44;
}
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

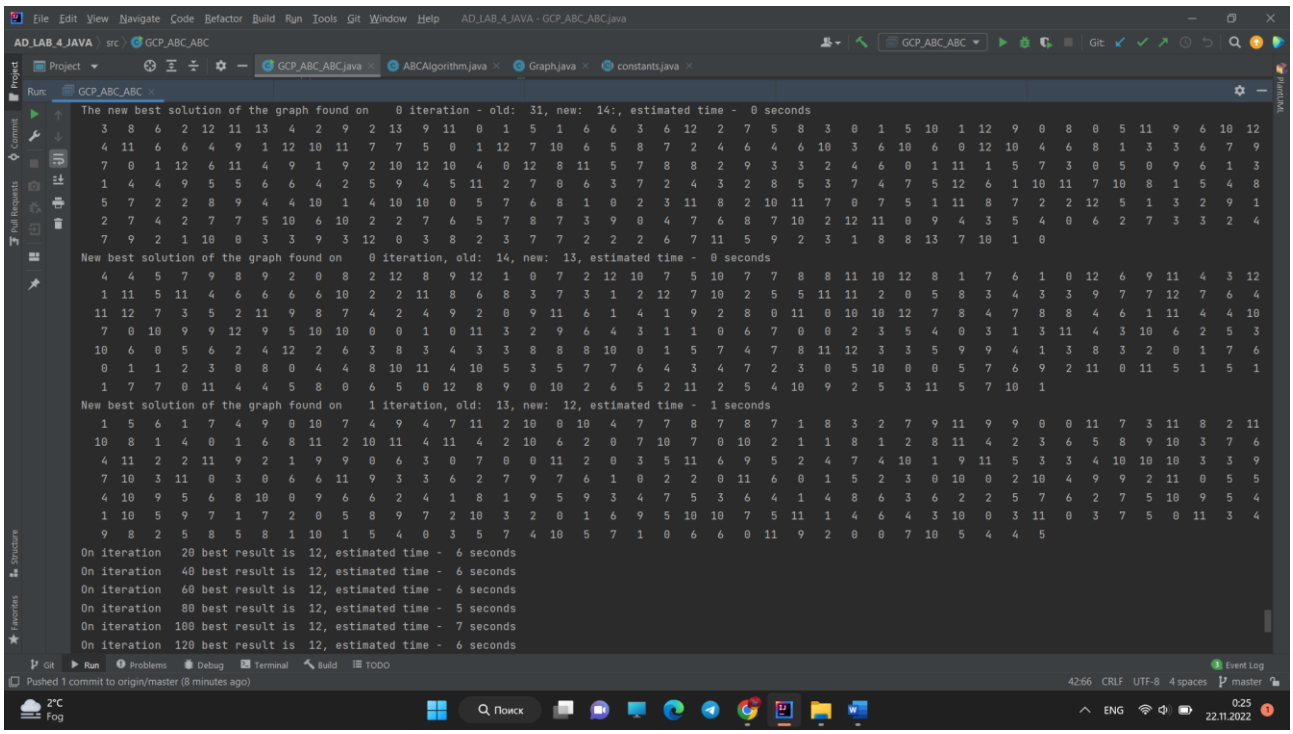

Рисунок 3.1 – процес роботи програми, покращення початкового стану.

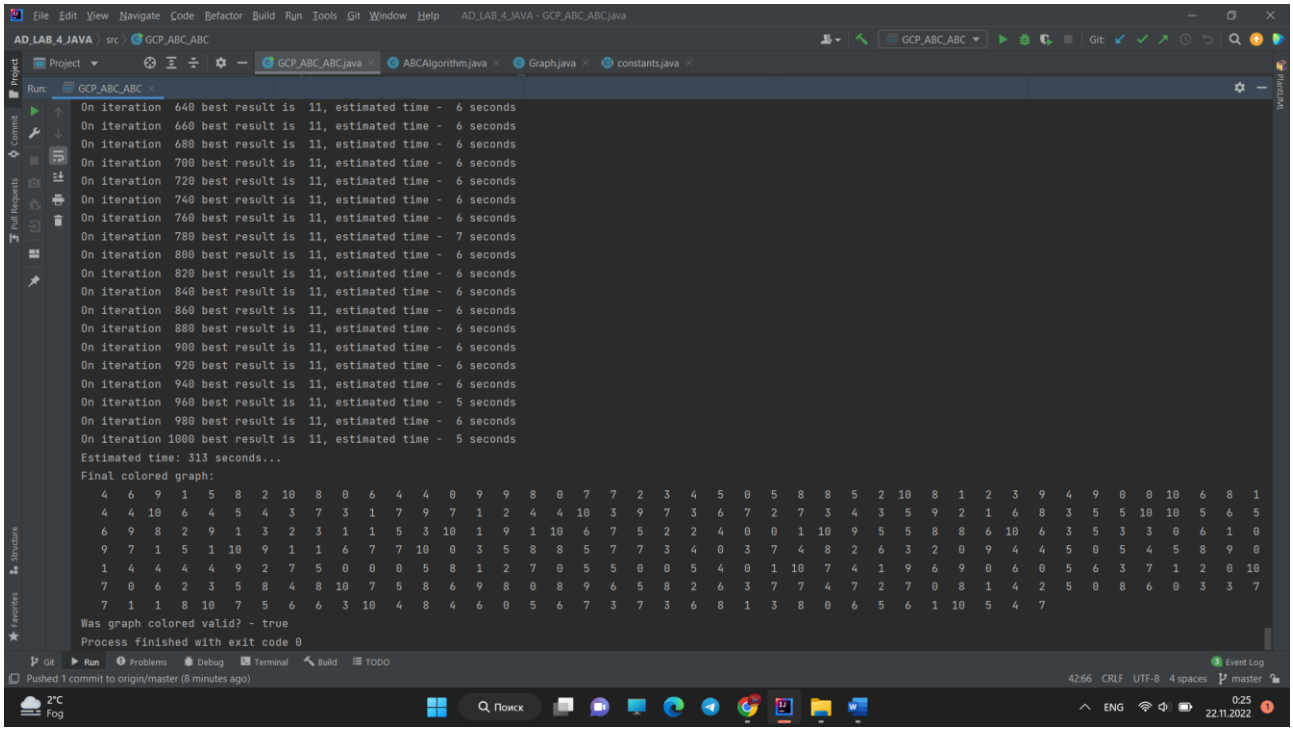

Рисунок 3.2 – результат роботи програми досягнений за 1000 ітерацій.

3.2 Тестування алгоритму

3.2.1 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**
0	31	260	11	520	11	780	11
20	13	280	11	540	11	800	11
40	12	300	11	560	11	820	11
60	12	320	11	580	11	840	11
80	12	340	11	600	11	860	11
100	12	360	11	620	11	880	11
120	12	380	11	640	11	900	11
140	12	400	11	660	11	920	11
160	12	420	11	680	11	940	11
180	12	440	11	700	11	960	11
200	12	460	11	720	11	980	11
220	11	480	11	740	11	1000	11
240	11	500	11	760	11	—	—

3.2.2 Графік залежності розв'язку від числа ітерацій

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

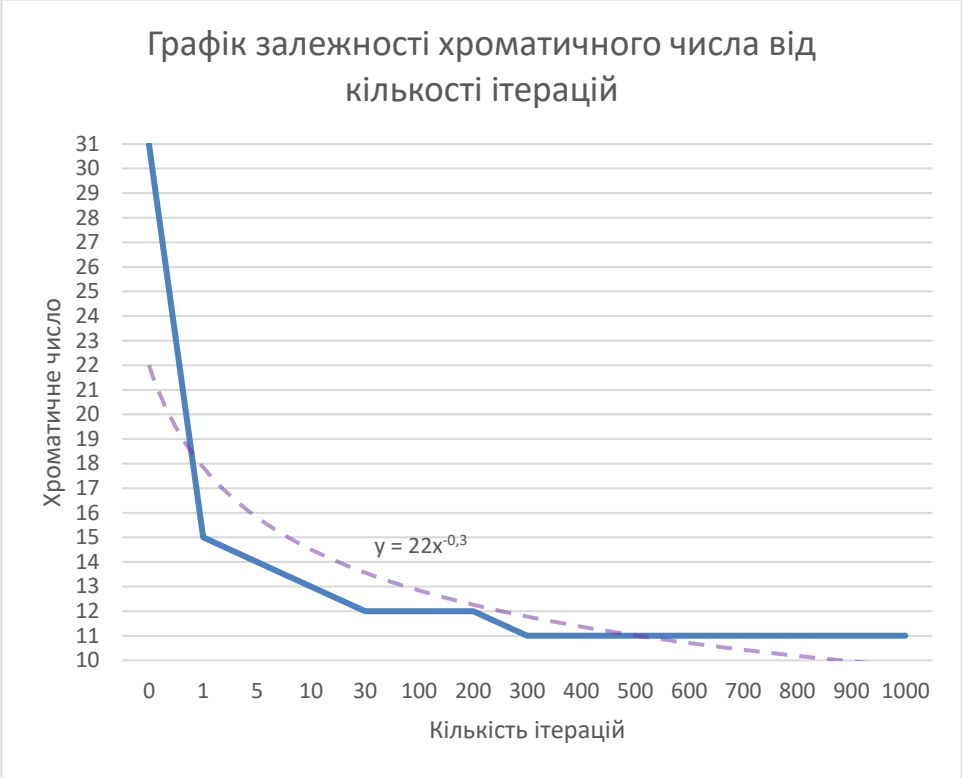

Рисунок 3.3 – Графік залежності хроматичного числа від кількості ітерацій.

ВИСНОВОК

В рамках даної лабораторної роботи Я розв’язав задачу розфарбування графу на 300 вершин, де степінь кожної від 1 до 30 включно, виконавши програмну реалізацію на мові програмування Java, бджолиним алгоритмом, а саме його модифікацією – штучна бджолина колонія з такими параметрами: число бджіл 60 із них 5 розвідники.

Також отримав графік залежності хроматичного числа від кількості ітерацій алгоритму, зафіксувавши на кожній двадцятій ітерації, яких всього 1000, значення цільової функції. З цього графіка видно, що вже після 200 ітерації сенсу продовжувати виконувати алгоритм при таких параметрах немає, бо максимальне хроматичне число після 200 ітерацій, яке вдалося зафіксувати для цього графа – 11.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 27.11.2022 включно максимальний бал дорівнює – 5. Після 27.11.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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