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

«Проєктування алгоритмів»

«Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1»

Варіант 3

Виконав студент ІП-14 Гайдучек Максим Андрійович

(шифр, прізвище, ім'я, по батькові)

Перевірив Соколовський Владислав Володимирович

( прізвище, ім'я, по батькові)

Київ 2022

**Лабораторна робота №4**

**Тема роботи:** Проектування і аналіз алгоритмів для вирішення NP-складних задач.

**Мета роботи:** вивчити основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою.

**Варіант:** 3

**Завдання**: Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).

**Хід роботи**

1. Код алгоритму

package lab.algorithms;  
  
import lab.entities.Node;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Random;  
import java.util.stream.Collectors;  
  
public class AbcAlgorithm {  
  
 private static final int *MAX\_ITERATIONS* = 10000;  
 private static final int *EXPLORER\_BEES* = 2;  
 private static final int *TOTAL\_BEES* = 30;  
  
 public static void paint(List<Node> nodes) {  
 List<Node> nodesCopy = *copyGraphNodes*(nodes), resultNodes;  
 Colors colors = new Colors();  
 *calcChromaticNumber*(nodesCopy, colors);  
 resultNodes = *copyGraphNodesWithColors*(nodesCopy);  
 int bestChromaticNumber = colors.getChromaticNumber();  
  
 for (int i = 0; i < *MAX\_ITERATIONS*; i += 1) {  
 System.*out*.printf("Iteration: %d, Best chromatic number: %d, Last chromatic number: %d\n",  
 i, bestChromaticNumber, colors.getChromaticNumber());  
  
 nodesCopy = *copyGraphNodes*(nodes);  
 colors = new Colors();  
 *calcChromaticNumber*(nodesCopy, colors);  
 if (colors.getChromaticNumber() >= bestChromaticNumber) {  
 continue;  
 }  
 resultNodes = *copyGraphNodesWithColors*(nodesCopy);  
 bestChromaticNumber = colors.getChromaticNumber();  
 }  
 nodes.clear();  
 nodes.addAll(resultNodes);  
  
 System.*out*.println("\nFinal iteration");  
 System.*out*.printf("Iteration: %d, Best chromatic number: %d, Last chromatic number: %d\n",  
 *MAX\_ITERATIONS*, bestChromaticNumber, colors.getChromaticNumber());  
 }  
  
 private static List<Node> copyGraphNodes(List<Node> nodes) {  
 List<Node> newNodes = nodes.stream().map(node -> new Node(node.getId())).toList();  
 for (Node newNode : newNodes) {  
 newNode.setNodes(  
 nodes.get(newNode.getId()).getNodes().stream()  
 .map(node -> newNodes.get(node.getId()))  
 .collect(Collectors.*toSet*())  
 );  
 }  
 return newNodes;  
 }  
  
 private static List<Node> copyGraphNodesWithColors(List<Node> nodes) {  
 return *copyGraphNodes*(nodes).stream()  
 .peek(node -> node.setColor(nodes.get(node.getId()).getColor()))  
 .toList();  
 }  
  
 private static void calcChromaticNumber(List<Node> nodes, Colors colors) {  
 List<Node> availableNodes = new ArrayList<>(nodes);  
 while (nodes.stream().anyMatch(node -> !node.isPained())) {  
 List<Node> selectedNodes = *selectNodes*(availableNodes);  
 *paintSelectedNodes*(selectedNodes, colors);  
 }  
 }  
  
 private static List<Node> selectNodes(List<Node> nodes) {  
 List<Node> selectedNodes = new ArrayList<>();  
 Random random = new Random();  
 for (int i = 0; i < *EXPLORER\_BEES* && !nodes.isEmpty(); i++) {  
 int index = random.nextInt(nodes.size());  
 Node node = nodes.get(index);  
 selectedNodes.add(node);  
 nodes.remove(index);  
 }  
 return selectedNodes;  
 }  
  
 private static void paintSelectedNodes(List<Node> selectedNodes, Colors colors) {  
 int[] degrees = selectedNodes.stream()  
 .mapToInt(Node::getDegree)  
 .toArray();  
 int totalDegrees = Arrays.*stream*(degrees).sum();  
 double[] nectarValues = Arrays.*stream*(degrees)  
 .mapToDouble(degree -> (double) degree / totalDegrees)  
 .toArray();  
 int onlookerBeesCount = *TOTAL\_BEES* - *EXPLORER\_BEES*;  
 int[] beesCounts = new int[nectarValues.length];  
 for (int i = 0; i < nectarValues.length; i++) {  
 beesCounts[i] = (int) (onlookerBeesCount \* nectarValues[i]);  
 onlookerBeesCount -= beesCounts[i];  
 }  
 for (int i = 0; i < selectedNodes.size(); i++) {  
 Node node = selectedNodes.get(i);  
 List<Node> neighbourNodes = new ArrayList<>(node.getNodes());  
 for (int j = 0; j < neighbourNodes.size() && j < beesCounts[i] - 1; j++) {  
 *paintNode*(neighbourNodes.get(j), colors);  
 }  
 *paintNode*(node, colors);  
 }  
 }  
  
 private static void paintNode(Node node, Colors colors) {  
 if (node.isPained()) {  
 return;  
 }  
 if (colors.hasNoColors()) {  
 node.setColor(colors.increaseColors());  
 return;  
 }  
 for (int i = 0; i <= colors.getColor(); i++) {  
 final int color = i;  
 if (node.getNodes().stream().noneMatch(neighbourNodes -> neighbourNodes.getColor() == color)) {  
 node.setColor(color);  
 return;  
 }  
 }  
 node.setColor(colors.increaseColors());  
 }  
}

1. Приклади роботи алгоритму

На рисунках 2.1.1 і 2.1.2 показані приклади роботи програми.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2.1.1 – Лог ітерацій

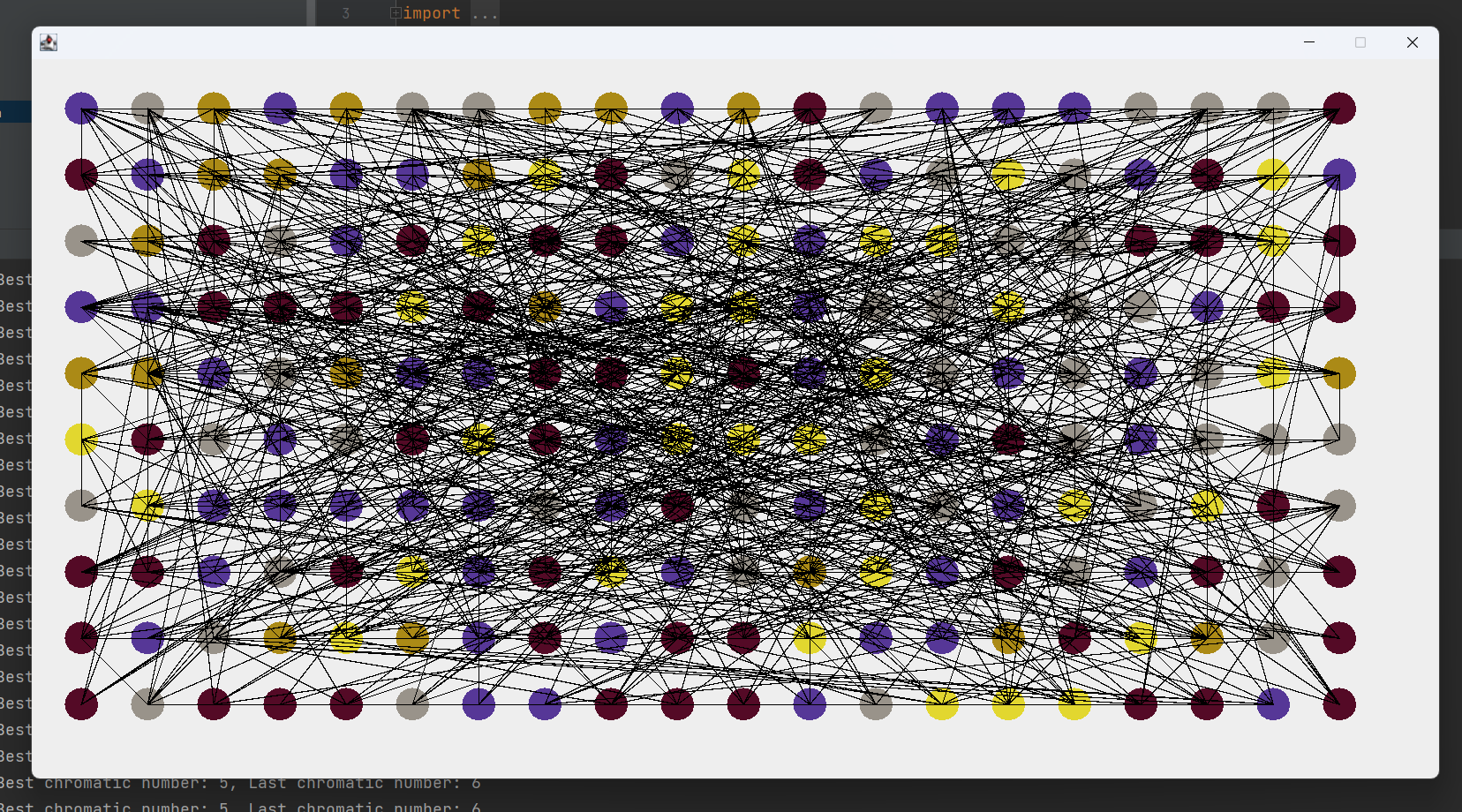


Рисунок 2.1.2 – Виведення графу

1. Тестування алгоритму

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

Таблиця 3.1 - значення цільової функції зі збільшенням кількості ітерацій

|  |  |
| --- | --- |
| Кількість ітерацій | Хроматичне число |
| 1 | 6 |
| 5 | 6 |
| 10 | 5 |
| 100 | 5 |
| 500 | 5 |
| 1000 | 5 |
| 5000 | 5 |
| 10000 | 5 |

На рисунку 3.2 наведений графік, який показує якість отриманого розв'язку.

Рисунок 3.1 - Графік залежності розв'язку від числа ітерацій

**Висновок**: під час виконання даної лабораторної роботи було розв’язано задачу розмальовки графу та розроблено бджолиний алгоритм АВС з 30 бджолами та 2 розвідниками. Було протестовано код та досліджено залежність розв'язку від числа ітерацій. Оскільки алгоритм має елемент випадковості, то чим більше ітерацій проведемо, тим більша ймовірність на знаходження правильного результату.