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

**Катедра 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-11 Лесів В. І.* _____
(шифр, прізвище, ім'я, по батькові)

**Перевірив** *Головченко М.Н.* _____
(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** |
|---|---|
| 1 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 2 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах). |
| 3 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 4 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |

| | |
|---|---|
| 5 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах). |
| 6 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 7 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 8 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах). |
| 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). |
| 10 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 11 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ |

| | |
|---|---|
| | = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах). |
| 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). |
| 13 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 14 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 15 | Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники). |
| 16 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 17 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (15 з них дикі, |

| | |
|---|---|
| | обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 18 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники). |
| 19 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 21 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники). |
| 22 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 23 | Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (15 з них елітні, |

| | |
|---|---|
| | подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 24 | Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники). |
| 25 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 26 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах). |
| 27 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 28 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 29 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах). |

| | |
|---|---|
| 30 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 31 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 32 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах). |
| 33 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 34 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 35 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах). |

# 3 ВИКОНАННЯ

## 3.1 Програмна реалізація алгоритму

### 3.1.1 Вихідний код

main.py

```
from algorithm import *
import matplotlib.pyplot as matplot
from graph_color import *

if __name__ == '__main__':
    graph = Graph(300, 1, 50)
    graph.firstColor(int(input("Введіть кількість початкових розфарбувань: ")))
    last = graph.beesAlgorithm(60)

    g = graphVisualisation()
    last = g.numToCol(last)
    g.addEdges(graph.adjacencyList)
    g.visualize(last, 300)

    matplot.plot(graph.X, graph.Y)
    matplot.xlabel('Кількість ітерацій')
    matplot.ylabel('Кількість кольорів в найкращому розфарбуванні')
    matplot.show()
```

graphColor.py

```
import networkx as nx
import matplotlib.pyplot as plt


class graphVisualisation:
    def __init__(self):
        self.visual = []

    def addEdges(self, adjacency):
        for i in range(len(adjacency)):
            for j in adjacency[i]:
                self.visual.append([i, j])

    def numToCol(self, list):
        dict = {1: "red", 2: "orange", 3: "yellow", 4: "green", 5: "cyan", 6:
"blue", 7: "violet", 8: "gold",
                9: "limegreen", 10: "darkorange", 11: "pink", 12: "magenta", 13:
"grey", 14: "black"}
        for i in dict:
            list = repl(list, i, dict[i])
        return list

    def visualize(self, color,k):
        g = nx.Graph()
        g.add_nodes_from([i for i in range(k)])
        g.add_edges_from(self.visual)
        nx.draw_networkx(g, node_color=color,with_labels=True)
        plt.show()


def repl(list, item, to):
    return [to if it == item else it for it in list]
```

algorithm.py

```
from random import randint, shuffle


class Graph:
    def __init__(self, nodesNumber, minPow, maxPow):
        self.V = nodesNumber  # Кількість вершин графа
        self.adjacencyList = [[] for i in range(self.V)]
        self.powers = [0 for i in range(self.V)]  # Степені вершин графа
        self.colourings = []  # Початкові розфарбування графа
        self.X = []
        self.Y = []
        # Генеруємо ребра так, щоб степінь відповідала умовам
        for i in range(self.V):
            degree = randint(minPow, maxPow)
            if self.powers[i] < degree:
                for j in range(self.powers[i], degree):
                    try:
                        connect = randint(i + 1, self.V - 1)
                    except ValueError:
                        continue
                    while self.powers[connect] == maxPow or connect == i or
connect in self.adjacencyList[i]:
                        connect = randint(0, self.V - 1)
                    self.adjacencyList[i].append(connect)
                    self.adjacencyList[connect].append(i)
                    self.powers[i] += 1
                    self.powers[connect] += 1

    def firstColor(self, num):
        for i in range(num):
            current = [0 for i in range(self.V)]
            order = list(range(self.V))
            shuffle(order)
            for verge in order:
                neighbourColor = [current[neighbour] for neighbour in
self.adjacencyList[verge]]
                curColor = 1
                while curColor in neighbourColor:
                    curColor += 1
                current[verge] = curColor
            self.colourings.append(current)

    def beesAlgorithm(self, bees):  # 5 розвідників, bees-5 фуражирів
        queue, vergeQueue, neighboursQueue = list(range(self.V)), [], []
        queue.sort(key=lambda x: self.powers[x])
        queue = queue[::-1]  # Черга з вершин з пріоритетом за спаданням степеня
вершини
        for verge in queue:
            for neighbour in self.adjacencyList[verge]:
                vergeQueue.append(verge)
                neighboursQueue.append(neighbour)
        vergeQueues = [vergeQueue.copy() for i in range(len(self.colourings))]
        neighboursQueues = [neighboursQueue.copy() for i in
range(len(self.colourings))]
        for i in range(1000):  # До 1000 ітерацій
            usedColors = [max(i) for i in self.colourings]  # оцінка корисності
ділянок
            if i == 0 or (i + 1) % 20 == 0:
                self.X.append(i)
```

```
                self.Y.append(min(usedColors))
            chosenList = []
            while len(chosenList) != 5:
                randd = self.colourings[randint(0, len(self.colourings) - 1)]
                if randd not in chosenList:
                    chosenList.append(randd)
            mostPersp = self.colourings[usedColors.index(min([max(l) for l in
chosenList]))]
            random = self.colourings[randint(0, len(self.colourings) - 1)]
            while random == mostPersp:
                random = self.colourings[randint(0, len(self.colourings) - 1)]
            for forager in range(max(self.powers)):
                if not vergeQueues[self.colourings.index(mostPersp)]:
                    vergeQueues[self.colourings.index(mostPersp)] =
vergeQueue.copy()
                    neighboursQueues[self.colourings.index(mostPersp)] =
neighboursQueue.copy()
                self.search(mostPersp,
vergeQueues[self.colourings.index(mostPersp)][0],

neighboursQueues[self.colourings.index(mostPersp)][0])
                vergeQueues[self.colourings.index(mostPersp)] = vergeQueues[

self.colourings.index(mostPersp)][1:]
                neighboursQueues[self.colourings.index(mostPersp)] =
neighboursQueues[self.colourings.index(
                    mostPersp)][1:]
            for forager in range(bees - 5 - max(self.powers)):
                if not vergeQueues[self.colourings.index(random)]:
                    vergeQueues[self.colourings.index(random)] =
vergeQueue.copy()
                    neighboursQueues[self.colourings.index(random)] =
neighboursQueue.copy()
                self.search(random,
vergeQueues[self.colourings.index(random)][0],
                            neighboursQueues[self.colourings.index(random)][0])
                vergeQueues[self.colourings.index(random)] =
vergeQueues[self.colourings.index(random)][1:]
                neighboursQueues[self.colourings.index(random)] =
neighboursQueues[self.colourings.index(random)][1:]
        return self.colourings[usedColors.index(min(usedColors))]

    def search(self, colouring, verge, neighbour):  # пошук перестановок
кольорів
        colouring[verge], colouring[neighbour] = colouring[neighbour],
colouring[verge]
        noConflicts = 1
        for conflict in self.adjacencyList[verge]:
            if colouring[verge] == colouring[conflict]:
                noConflicts = 0
                break
        for conflict in self.adjacencyList[neighbour]:
            if colouring[neighbour] == colouring[conflict]:
                noConflicts = 0
                break
        if noConflicts:
            if colouring[verge] > colouring[neighbour]:
                self.reduce(colouring, verge)
                self.reduce(colouring, neighbour)
            else:
                self.reduce(colouring, neighbour)
                self.reduce(colouring, verge)
        else:
            colouring[verge], colouring[neighbour] = colouring[neighbour],
```

```
colouring[verge]

    def reduce(self, colouring, verge):
        curColor = colouring[verge]
        for changeColor in range(1, colouring[verge]):
            colouring[verge] = changeColor
            noConflict = 1
            for neighbour in self.adjacencyList[verge]:
                if colouring[verge] == colouring[neighbour]:
                    noConflict = 0
                    colouring[verge] = curColor
                    break
            if noConflict:
                break
```

### 3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

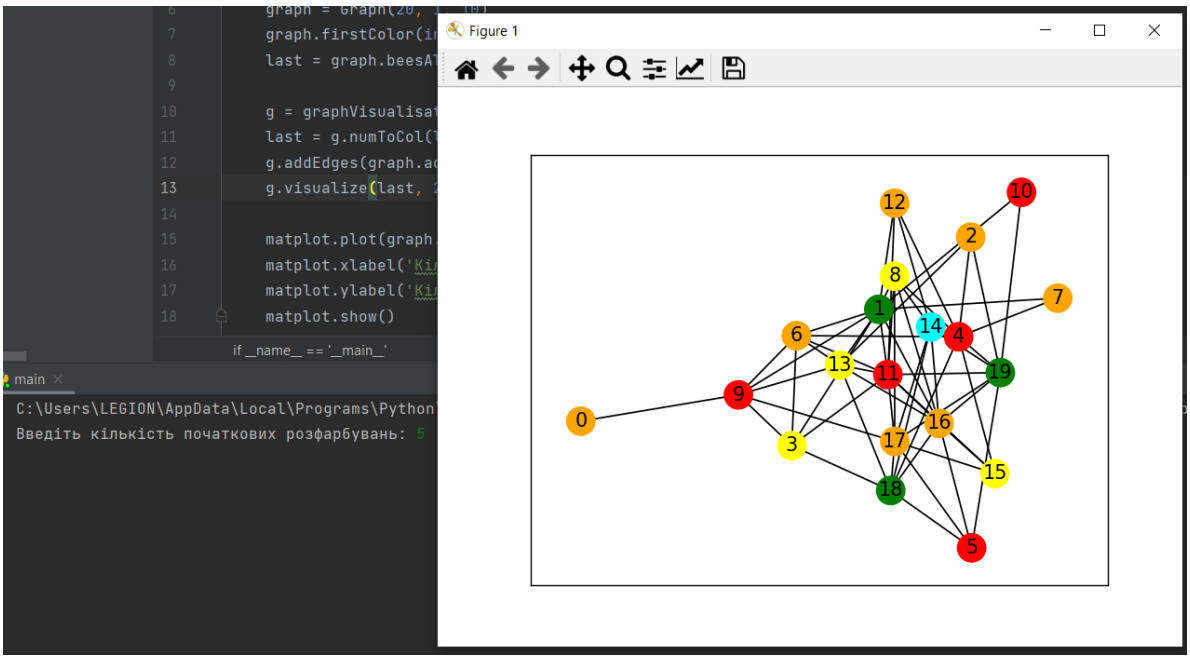


Рисунок 3.1 – Візуалізація розфарбування графу на 20 вершин з 5 початковими розфарбуваннями.

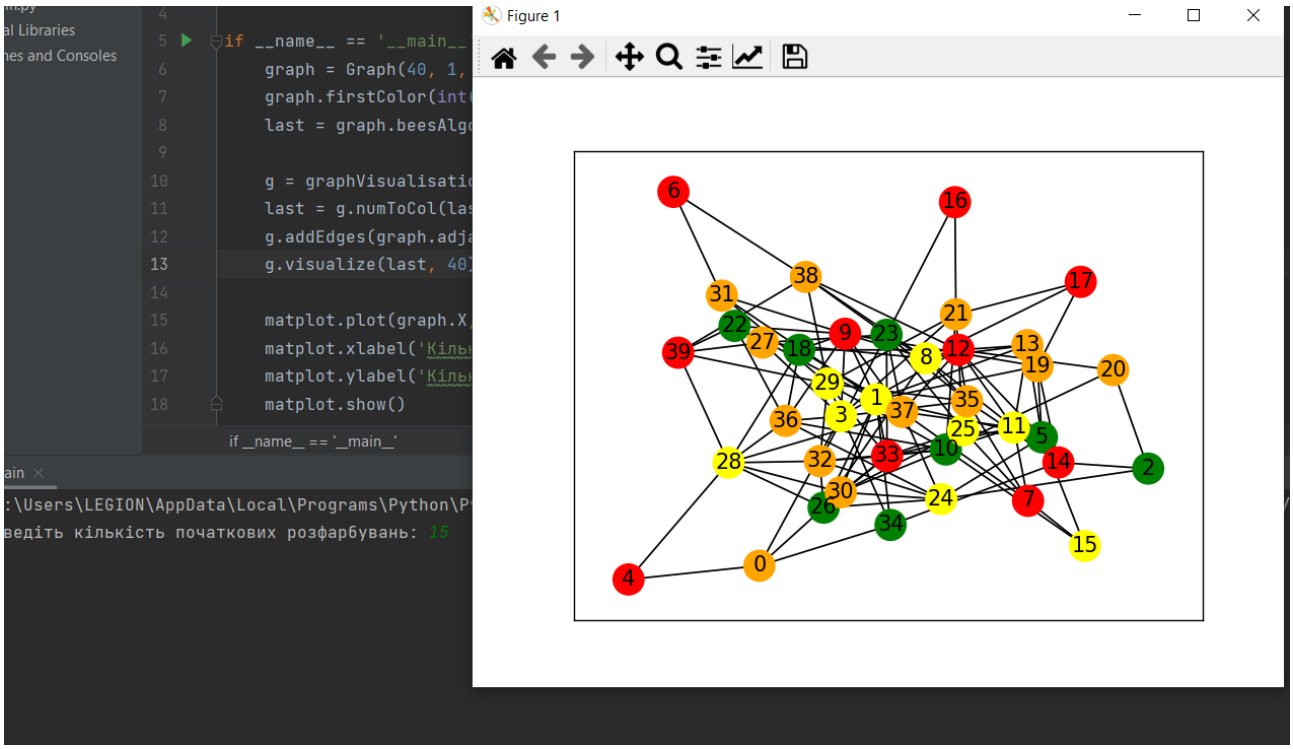


Рисунок 3.2 – Візуалізація розфарбування графу на 40 вершин з 15 початковими розфарбуваннями.

## 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.

Таблиця 3.1

| Ітерація | Значення ЦФ |
|---|---|
| 0 | 14 |
| 20 | 14 |
| 40 | 14 |
| 60 | 14 |
| 80 | 14 |
| 100 | 14 |
| 120 | 14 |
| 140 | 14 |
| 160 | 14 |
| 180 | 14 |
| 200 | 14 |
| 220 | 14 |
| 240 | 14 |
| 260 | 14 |
| 280 | 14 |
| 300 | 14 |
| 320 | 14 |
| 340 | 14 |
| 360 | 13 |
| 380 | 13 |
| 400 | 13 |
| 420 | 13 |
| 440 | 13 |
| 460 | 13 |
| 480 | 13 |
| 500 | 13 |
| 520 | 13 |
| 540 | 13 |
| 560 | 13 |
| 580 | 13 |
| 600 | 13 |
| 620 | 13 |
| 640 | 13 |
| 660 | 13 |

Продовження таблиці 3.1

| | |
|---|---|
| 680 | 13 |
| 700 | 13 |
| 720 | 13 |
| 740 | 13 |
| 760 | 13 |
| 780 | 13 |
| 800 | 13 |
| 820 | 13 |
| 840 | 13 |
| 860 | 13 |
| 880 | 13 |
| 900 | 13 |
| 920 | 13 |
| 940 | 13 |
| 960 | 13 |
| 980 | 13 |
| 1000 | 13 |

### 3.2.2 Графіки залежності розв'язку від числа ітерацій

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

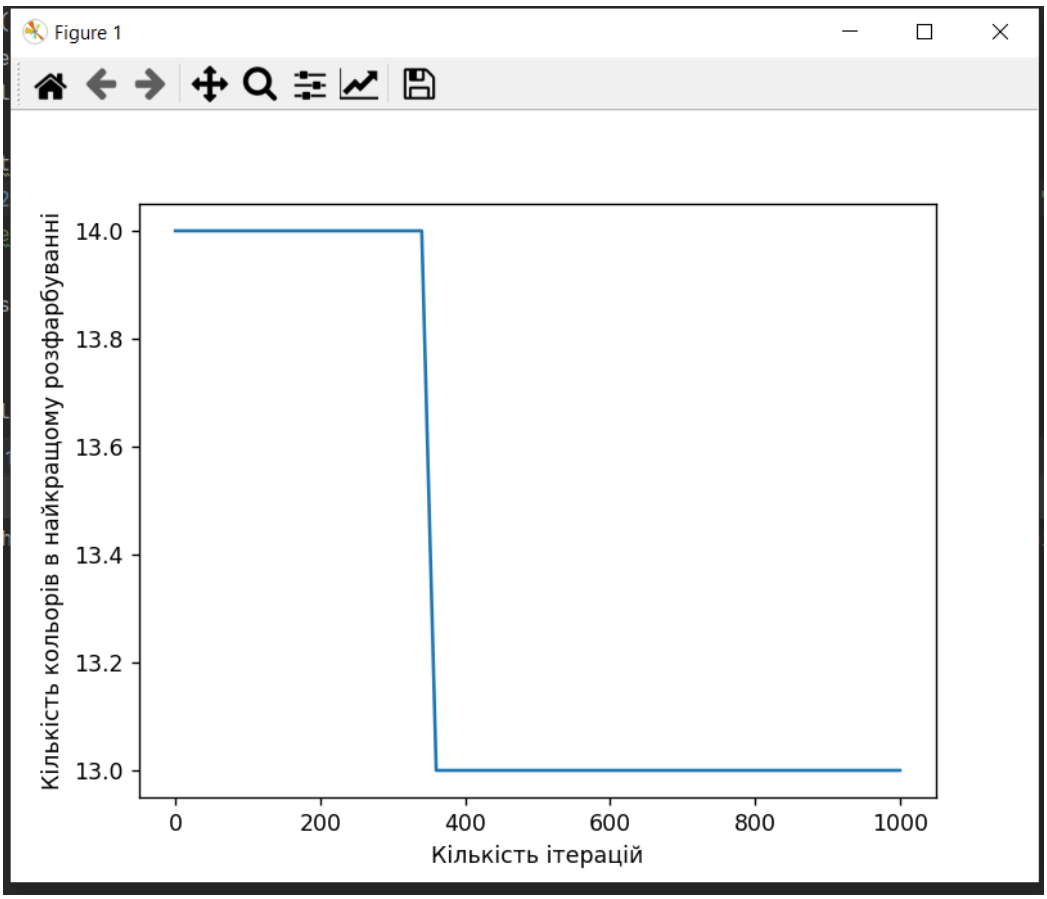


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

# ВИСНОВОК

В рамках даної лабораторної роботи було вивчено основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою. Було досліджено та спроєктовано алгоритм «Класичний бджолиний алгоритм» відповідно до умови варіанту. Ідея бджолиного алгоритму полягає в тому, що всі бджоли на кожному кроці будуть вибирати як елітні ділянки для дослідження, так і ділянки в околиці елітних, що дозволить, по-перше, урізноманітнити популяцію рішень на наступних ітераціях, по-друге, збільшити ймовірність виявлення близьких до оптимальних рішень.

У результаті виконання лабораторної роботи після наведення прикладів, проведеного аналізу та тестування переконуємося, що бджолиний алгоритм дійсно є ефективним для вирішення задачі розфарбування графу, адже з 1000 ітерацій між 340 та 360 ітерацією нашого прикладу кількість кольорів, необхідна для розфарбування, зменшилася на 1 від найкращого початкового.

Використовуючи програмні засоби мови Python, модулі matplotlib і networkx, отримуємо доступну візуалізацію й коректний результат.

# 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%.