**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-12 Кушнір Ганна Вікторівна*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 14](#_Toc51260922)

[3.2 Тестування алгоритму 15](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 15](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 15](#_Toc51260925)

[Висновок 17](#_Toc51260926)

[Критерії оцінювання 18](#_Toc51260927)

# Мета лабораторної роботи

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

# 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 М = 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 М = 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 М = 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 М = 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 М = 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 М = 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 М = 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

Файл «main.py»

from creation import \*

from algorithm import \*

if \_\_name\_\_ == "\_\_main\_\_":

    path = 'file\_lab4.txt'

    f = CreateGraph()

    f.create\_and\_save\_to\_file(path)

    algorithm = GraphColoring(path)

    algorithm.bee\_algorithm()

Файл «creation.py»

import numpy as np

class CreateGraph:

    def \_\_init\_\_(self, num: int = 100, min\_pow: int = 1, max\_pow: int = 20):

        self.num = num

        self.min\_pow = min\_pow

        self.max\_pow = max\_pow

    def create\_and\_save\_to\_file(self, path: str):

        self.\_generate\_graph()

        with open(path, 'w') as f:

            f.write(str(self.num) + '\n')

            for edge in self.edges:

                f.write(str(edge[0]) + ' ' + str(edge[1]) + '\n')

    def \_generate\_graph(self):

        self.edges = []

        counts = [0 for i in range(self.num)]

        for vertex in range(self.num):

            num\_of\_neighbors = np.random.randint(1, self.max\_pow)

            if counts[vertex] + num\_of\_neighbors > self.max\_pow:

                num\_of\_neighbors = self.max\_pow - counts[vertex]

            counts[vertex] += num\_of\_neighbors

            i = 0

            neighbors = []

            while i < num\_of\_neighbors:

                neighbor = np.random.randint(0, self.num)

                if vertex != neighbor and ([vertex, neighbor] not in self.edges) and ([neighbor, vertex] not in self.edges):

                    if counts[neighbor] < self.max\_pow:

                        neighbors.append(neighbor)

                        counts[neighbor] += 1

                        i += 1

            for neighbor in neighbors:

                self.edges.append([vertex, neighbor])

Файл «algorithm.py»

import numpy as np

import networkx as nx

import matplotlib.pyplot as plt

"""

Задача розфарбовування графу (100 вершин, степінь вершини не більше 20,

але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них

3 розвідники).

"""

class GraphColoring:

    def \_\_init\_\_(self, path: str, iterations: int = 1000):

        self.scout\_bees = 3

        self.foragers = 30

        self.iterations = iterations

        self.edges = []

        with open(path, 'r') as f:

            lines = f.readlines()

            self.num = int(lines[0])

            lines = lines[1:]

            self.edges = [list(map(int, line.split())) for line in lines]

        self.counts = self.\_count\_occurence\_of\_vertices\_and\_sort()

    def \_count\_occurence\_of\_vertices\_and\_sort(self):

        vertices\_in\_edges = [start for start, end in self.edges] + [end for start, end in self.edges]

        counts = [0 for i in range(self.num)]

        for vertex in set(vertices\_in\_edges):

            counts[vertex] = [vertex, vertices\_in\_edges.count(vertex)]

        for i in range(len(counts)):

            for j in range(i + 1, len(counts)):

                if counts[i][1] < counts[j][1]:

                    temp = counts[j]

                    counts[j] = counts[i]

                    counts[i] = temp

        return counts

    def \_get\_neighbors(self, vertex: int):

        neighbors = []

        for start, end in self.edges:

            if start == vertex:

                neighbors.append(end)

            if end == vertex:

                neighbors.append(start)

        return set(neighbors)

    def \_get\_nth\_vertex\_with\_highest\_multiplicity(self, n: int):

        return self.counts[n % self.num][0]

    def \_get\_available\_color(self, vertex: int, num\_of\_colors: int, vertex\_colors: list[int]):

        neighbors = self.\_get\_neighbors(vertex)

        available\_colors = [color for color in range(num\_of\_colors)]

        if vertex\_colors[vertex] in available\_colors:

            available\_colors.remove(vertex\_colors[vertex])

        for neighbor in neighbors:

            if vertex\_colors[neighbor] in available\_colors:

                available\_colors.remove(vertex\_colors[neighbor])

        if len(available\_colors) == 0:

            return -1

        return available\_colors[0]

    def \_is\_color\_available(self, vertex: int, color: int, vertex\_colors: list[int]):

        neighbors = self.\_get\_neighbors(vertex)

        for neighbor in neighbors:

            if vertex\_colors[neighbor] == color:

                return False

        return True

    def greedy\_algorithm(self):

        self.vertex\_colors = [-1 for i in range(self.num)]

        curr\_color = 0

        while -1 in self.vertex\_colors:

            for vertex in range(self.num):

                if self.vertex\_colors[vertex] == -1:

                    if self.\_is\_color\_available(vertex, curr\_color, self.vertex\_colors):

                        self.vertex\_colors[vertex] = curr\_color

            curr\_color += 1

        self.draw\_graph('RESULT OF GREEDY ALGORITHM')

        return curr\_color

    def \_reduce\_num\_of\_colors(self, vertex: int, num\_of\_colors: int):

            neighbors = self.\_get\_neighbors(vertex)

            for neighbor in neighbors:

                temp = self.vertex\_colors.copy()

                temp[vertex], temp[neighbor] = temp[neighbor], temp[vertex]

                if self.\_is\_color\_available(neighbor, temp[neighbor], temp) and self.\_is\_color\_available(vertex, temp[vertex], temp):

                    new\_color = self.\_get\_available\_color(neighbor, num\_of\_colors, temp)

                    if new\_color != -1:

                        temp[neighbor] = new\_color

                        self.vertex\_colors = temp.copy()

    def bee\_algorithm(self):

        num\_of\_colors = self.greedy\_algorithm()

        for k in range(self.iterations):

            vertex = self.\_get\_nth\_vertex\_with\_highest\_multiplicity(k)

            lst = [vertex]

            current = 0

            ancestor = -1

            flag = 1

            while len(lst) < self.foragers:

                vertex = lst[current]

                neighbors = self.\_get\_neighbors(vertex)

                for neighbor in neighbors:

                    if neighbor != ancestor:

                        lst.append(neighbor)

                    if flag == 0:

                        lst.append(np.random.randint(0, self.num + 1))

                        flag = 1

                ancestor = vertex

                current += 1

                flag = 0

            for vertex in lst:

                self.\_reduce\_num\_of\_colors(vertex, num\_of\_colors)

            if k % 20 == 19:

                print('Iteration: ' + str(k + 1))

                print('Number of colors:', len(set(self.vertex\_colors)))

                print('Colors of vertices:', self.vertex\_colors)

        self.draw\_graph('RESULT OF BEES ALGORITHM')

    def draw\_graph(self, header: str):

        print(header)

        print('Number of colors:', len(set(self.vertex\_colors)))

        print('Colors of vertices:', self.vertex\_colors)

        graph = nx.Graph()

        for u, v in self.edges:

            graph.add\_edge(u, v)

        pos = nx.spring\_layout(graph)

        colors = ['red', 'magenta', 'orange', 'yellow', 'green', 'cyan', 'blue', 'purple', 'pink', 'brown', 'grey', 'black']

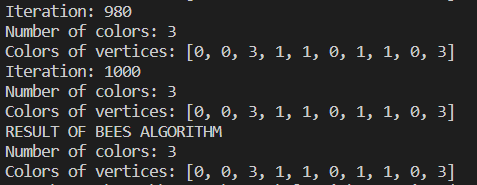
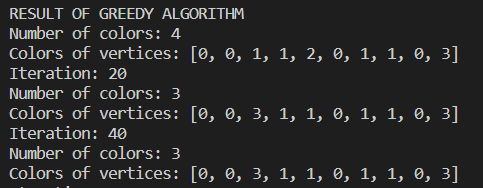
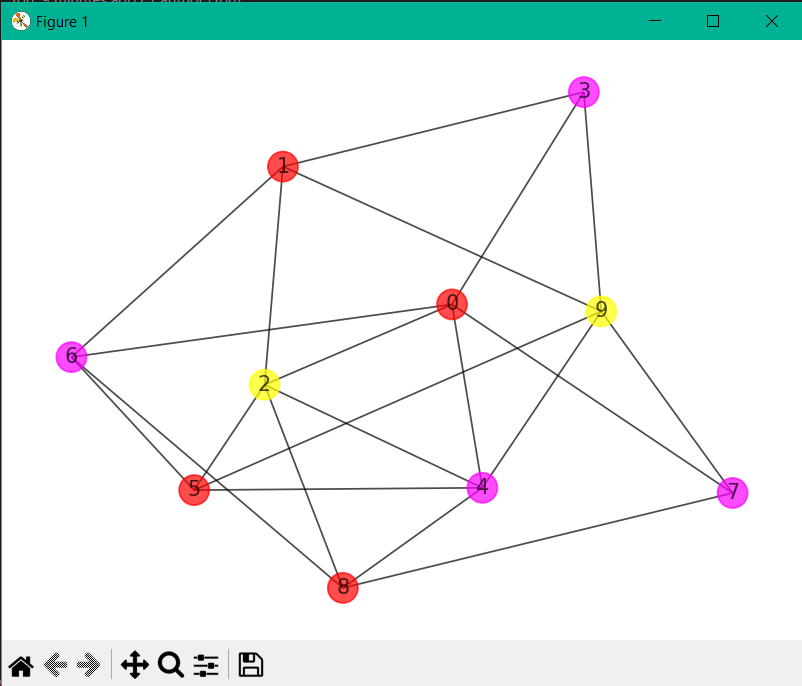
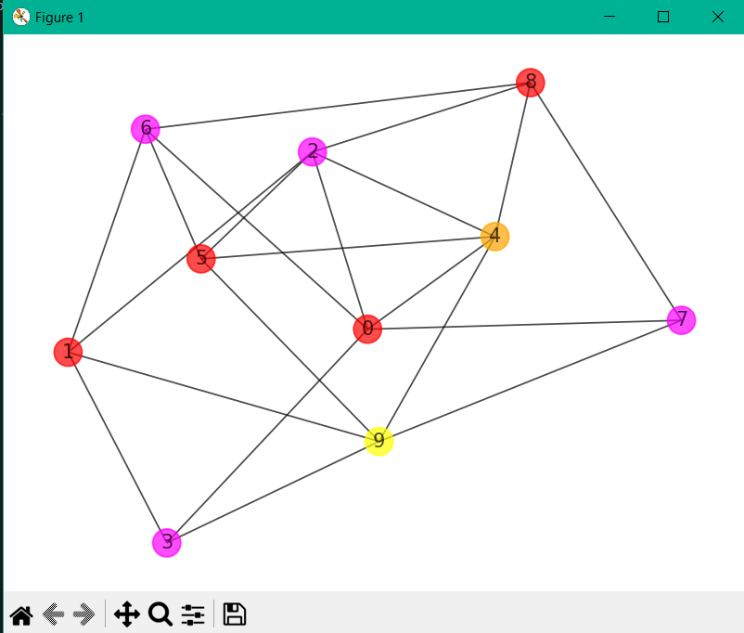
        vertex\_colors\_names = [colors[self.vertex\_colors[vertex]] for vertex in graph.nodes()]

        nx.draw(graph, pos, with\_labels = True, node\_color = vertex\_colors\_names, edge\_color = 'black', alpha = 0.7)

        plt.show()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



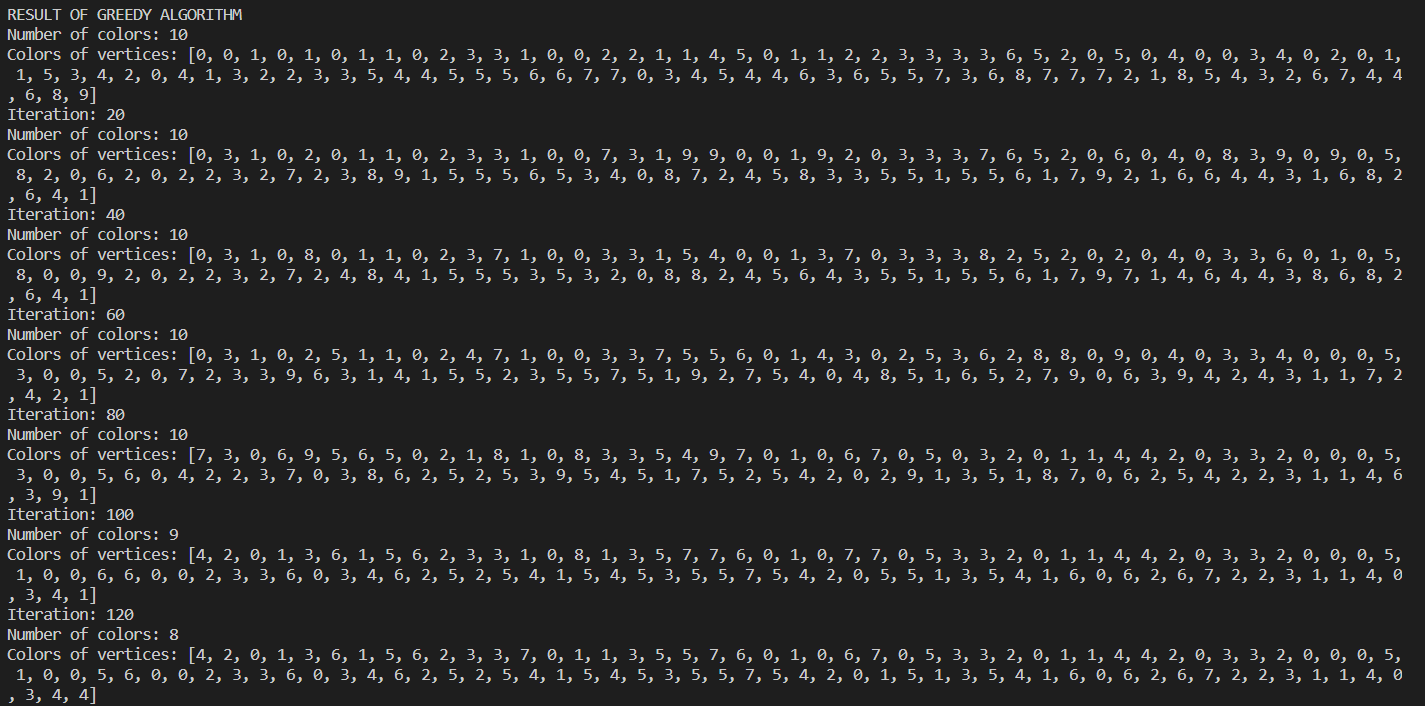
Рисунок 3.1 – Приклад роботи програми для графу з кількістю вершин num = 10 та максимальним степенем вершини max\_pow = 5.

Рисунок 3.3 – Приклад роботи програми для графу, характеристики якого задані умовою лабораторної роботи

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ітерація | Значення цільової функції | Ітерація | Значення цільової функції | Ітерація | Значення цільової функції |
| 0 | 10 | 340 | 8 | 680 | 8 |
| 20 | 10 | 360 | 8 | 700 | 8 |
| 40 | 10 | 380 | 8 | 720 | 8 |
| 60 | 10 | 400 | 8 | 740 | 8 |
| 80 | 10 | 420 | 8 | 760 | 8 |
| 100 | 9 | 440 | 8 | 780 | 8 |
| 120 | 8 | 460 | 8 | 800 | 8 |
| 140 | 8 | 480 | 8 | 820 | 8 |
| 160 | 8 | 500 | 8 | 840 | 8 |
| 180 | 8 | 520 | 8 | 860 | 8 |
| 200 | 8 | 540 | 8 | 880 | 8 |
| 220 | 8 | 560 | 8 | 900 | 8 |
| 240 | 8 | 580 | 8 | 920 | 8 |
| 260 | 8 | 600 | 8 | 940 | 8 |
| 280 | 8 | 620 | 8 | 960 | 8 |
| 300 | 8 | 640 | 8 | 980 | 8 |
| 320 | 8 | 660 | 8 | 1000 | 8 |

### Графіки залежності розв'язку від числа ітерацій

На рисунку 3.4 наведений графік, який показує якість отриманого розв'язку.

Рисунок 3.4 – Графік залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи було вивчено основні підходи до формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою. Було розроблено алгоритм вирішення задачі «розфарбування графу», за основу якого було взято класичний бджолиний алгоритм, і виконано його програмну реалізацію на мові програмування Python.

Після кожних 20 ітерацій алгоритму до 1000 було зафіксовано якість отриманого розв'язку і за отриманими результатами було побудовано графік залежності значення цільової функції від числа ітерацій.

Отже, на графах з невеликою кількістю вершин (близько 10-20) бджолиний алгоритм знаходить найбільш оптимальний розв’язок за 2-3 ітерації. При цьому на графах з досить великою кількістю вершин (більше 100) пошук найоптимальнішого розв’язку може відбуватися протягом 100-120 ітерацій алгоритму.

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%.