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-12 Кушнір Ганна Вікторівна* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 10

3.1 Програмна реалізація алгоритму 10

3.1.1 Вихідний код 10

3.1.2 Приклади роботи 14

3.2 Тестування алгоритму 15

3.2.1 Значення цільової функції зі збільшенням кількості ітерацій . 15

3.2.2 Графіки залежності розв'язку від числа ітерацій 15

ВИСНОВОК 17

КРИТЕРІЇ ОЦІНЮВАННЯ 18

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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, 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

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»

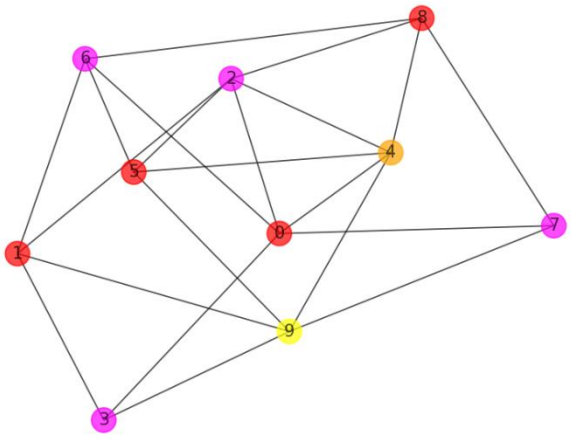
```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

"""
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)
```

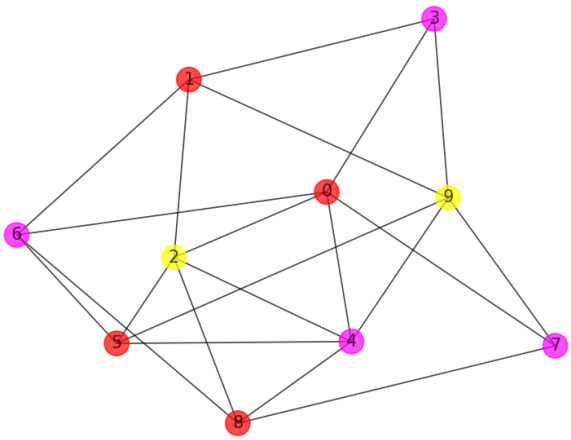
```
            if end == vertex:
                neighbors.append(start)
        return set(neighbors)

    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()
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.






```
RESULT OF GREEDY ALGORITHM
Number of colors: 4
Colors of vertices: [0, 0, 1, 1, 2, 0, 1, 1, 0, 3]
Iteration: 20
Number of colors: 3
Colors of vertices: [0, 0, 3, 1, 1, 0, 1, 1, 0, 3]
Iteration: 40
Number of colors: 3
Colors of vertices: [0, 0, 3, 1, 1, 0, 1, 1, 0, 3]
Iteration: 980
Number of colors: 3
Colors of vertices: [0, 0, 3, 1, 1, 0, 1, 1, 0, 3]
Iteration: 1000
Number of colors: 3
Colors of vertices: [0, 0, 3, 1, 1, 0, 1, 1, 0, 3]
RESULT OF BEES ALGORITHM
Number of colors: 3
Colors of vertices: [0, 0, 3, 1, 1, 0, 1, 1, 0, 3]
```

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

```
RESULT OF GREEDY ALGORITHM
Number of colors: 10
Colors of vertices: [0, 0, 1, 0, 1, 0, 1, 1, 0, 2, 3, 3, 1, 0, 0, 2, 2, 1, 1, 4, 5, 0, 1, 1, 2, 2, 3, 3, 3, 3, 6, 5, 2, 0, 5, 0, 4, 0, 0, 3, 4, 0, 2, 0, 1,
 1, 5, 3, 4, 2, 0, 4, 1, 3, 2, 2, 3, 3, 5, 4, 4, 5, 5, 5, 6, 6, 7, 7, 0, 3, 4, 5, 4, 4, 6, 3, 6, 5, 5, 7, 3, 6, 8, 7, 7, 7, 2, 1, 8, 5, 4, 3, 2, 6, 7, 4, 4
, 6, 8, 9]
Iteration: 20
Number of colors: 10
Colors of vertices: [0, 3, 1, 0, 2, 0, 1, 1, 0, 2, 3, 3, 1, 0, 0, 7, 3, 1, 9, 9, 0, 0, 1, 9, 2, 0, 3, 3, 3, 7, 6, 5, 2, 0, 6, 0, 4, 0, 8, 3, 9, 0, 9, 0, 5,
 8, 2, 0, 6, 2, 0, 2, 2, 3, 2, 7, 2, 3, 8, 9, 1, 5, 5, 5, 6, 5, 3, 4, 0, 8, 7, 2, 4, 5, 8, 3, 3, 5, 5, 1, 5, 5, 6, 1, 7, 9, 2, 1, 6, 6, 4, 4, 3, 1, 6, 8, 2
, 6, 4, 1]
Iteration: 40
Number of colors: 10
Colors of vertices: [0, 3, 1, 0, 8, 0, 1, 1, 0, 2, 3, 7, 1, 0, 0, 3, 3, 1, 5, 4, 0, 0, 1, 3, 7, 0, 3, 3, 3, 8, 2, 5, 2, 0, 2, 0, 4, 0, 3, 3, 6, 0, 1, 0, 5,
 8, 0, 0, 9, 2, 0, 2, 2, 3, 2, 7, 2, 4, 8, 4, 1, 5, 5, 5, 3, 5, 3, 2, 0, 8, 8, 2, 4, 5, 6, 4, 3, 5, 5, 1, 5, 5, 6, 1, 7, 9, 7, 1, 4, 6, 4, 4, 3, 8, 6, 8, 2
, 6, 4, 1]
Iteration: 60
Number of colors: 10
Colors of vertices: [0, 3, 1, 0, 2, 5, 1, 1, 0, 2, 4, 7, 1, 0, 0, 3, 3, 7, 5, 5, 6, 0, 1, 4, 3, 0, 2, 5, 3, 6, 2, 8, 8, 0, 9, 0, 4, 0, 3, 3, 4, 0, 0, 0, 5,
 3, 0, 0, 5, 2, 0, 7, 2, 3, 3, 9, 6, 3, 1, 4, 1, 5, 5, 2, 3, 5, 5, 7, 5, 1, 9, 2, 7, 5, 4, 0, 4, 8, 5, 1, 6, 5, 2, 7, 9, 0, 6, 3, 9, 4, 2, 4, 3, 1, 1, 7, 2
, 4, 2, 1]
Iteration: 80
Number of colors: 10
Colors of vertices: [7, 3, 0, 6, 9, 5, 6, 5, 0, 2, 1, 8, 1, 0, 8, 3, 3, 5, 4, 9, 7, 0, 1, 0, 6, 7, 0, 5, 0, 3, 2, 0, 1, 1, 4, 4, 2, 0, 3, 3, 2, 0, 0, 0, 5,
 3, 0, 0, 5, 6, 0, 4, 2, 2, 3, 7, 0, 3, 8, 6, 2, 5, 2, 5, 3, 9, 5, 4, 5, 1, 7, 5, 2, 5, 4, 2, 0, 2, 9, 1, 3, 5, 1, 8, 7, 0, 6, 2, 5, 4, 2, 2, 3, 1, 1, 4, 6
, 3, 9, 1]
Iteration: 100
Number of colors: 9
Colors of vertices: [4, 2, 0, 1, 3, 6, 1, 5, 6, 2, 3, 3, 1, 0, 8, 1, 3, 5, 7, 7, 6, 0, 1, 0, 7, 7, 0, 5, 3, 3, 2, 0, 1, 1, 4, 4, 2, 0, 3, 3, 2, 0, 0, 0, 5,
 1, 0, 0, 6, 6, 0, 0, 2, 3, 3, 6, 0, 3, 4, 6, 2, 5, 2, 5, 4, 1, 5, 4, 5, 3, 5, 5, 7, 5, 4, 2, 0, 5, 5, 1, 3, 5, 4, 1, 6, 0, 6, 2, 6, 7, 2, 2, 3, 1, 1, 4, 0
, 3, 4, 1]
Iteration: 120
Number of colors: 8
Colors of vertices: [4, 2, 0, 1, 3, 6, 1, 5, 6, 2, 3, 3, 7, 0, 1, 1, 3, 5, 5, 7, 6, 0, 1, 0, 6, 7, 0, 5, 3, 3, 2, 0, 1, 1, 4, 4, 2, 0, 3, 3, 2, 0, 0, 0, 5,
 1, 0, 0, 5, 6, 0, 0, 2, 3, 3, 6, 0, 3, 4, 6, 2, 5, 2, 5, 4, 1, 5, 4, 5, 3, 5, 5, 7, 5, 4, 2, 0, 1, 5, 1, 3, 5, 4, 1, 6, 0, 6, 2, 6, 7, 2, 2, 3, 1, 1, 4, 0
, 3, 4, 4]
```

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

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.

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

3.2.2 Графіки залежності розв'язку від числа ітерацій

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

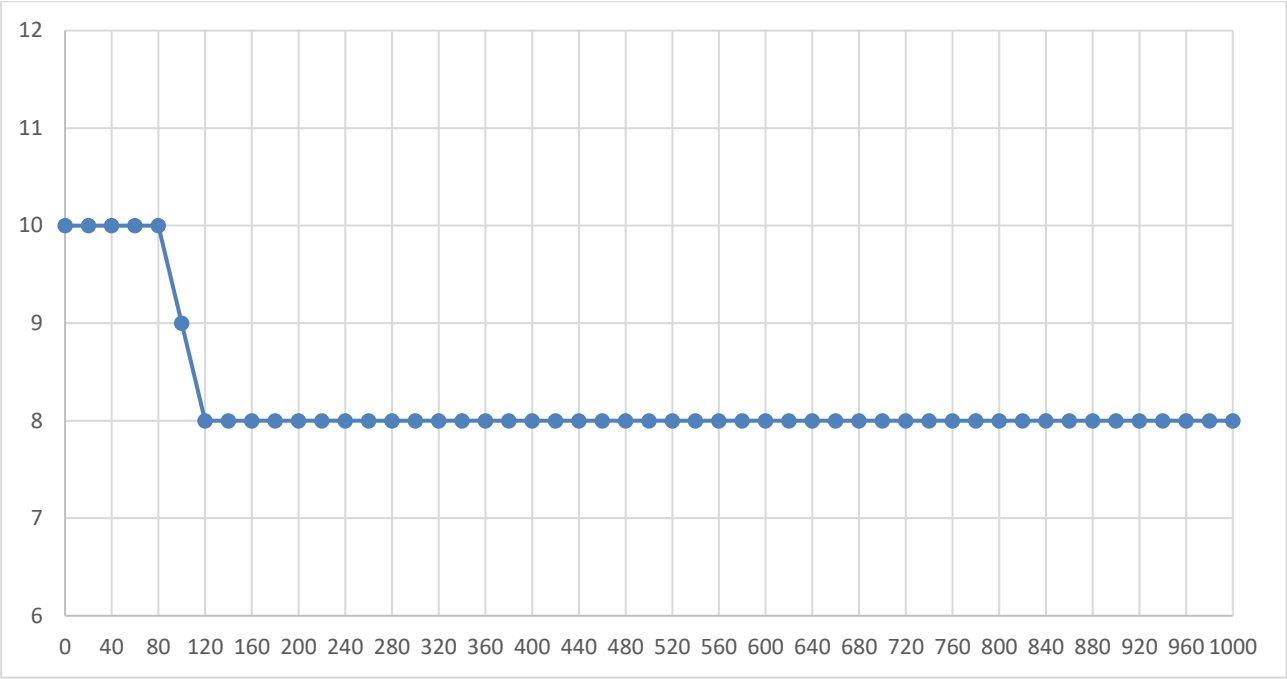


Рисунок 3.4 – Графік залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи було вивчено основні підходи до формалізації метаевристичних 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%.