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 Сімчук Андрій Володимирович* _____
(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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, 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

3.1.1.1 Файл «main.py»

```
from input_file import *
from algorithm import *


def main():
    fill = InputFile(file='files/input.txt')
    fill.create()

    algorithms = Algorithm()
    algorithms.bees()


if __name__ == "__main__":
    main()
```

3.1.1.2 Файл «input_file.py»

```
import numpy as np


class InputFile:
    # Створити довільний вхідний файл для графа
    def __init__(self, n=200, min_pow=1, max_pow=30,
                 file='files/input.txt'):
        # n - кількість вершин у графі, min_pow - мінімальний степінь
          вершини
        # max_pow - максимальний степінь вершини, file - назва вихідного
          файлу
        self.n = n
        self.min_pow = min_pow
        self.max_pow = max_pow
        self.file = file

    def create(self):
        # Створити вхідний файл
        # Створити випадковий список зв'язків з використанням обмежень
        lst = []
        for i in range(self.n):
            num_of_edges = np.random.randint(1, self.max_pow + 1)
            all_vertexes = np.arange(self.n)
            np.random.shuffle(all_vertexes)
            neighbours = all_vertexes[:num_of_edges]

            for neighbor in neighbours:
                lst.append([i, neighbor])

        # Зберегти матрицю у файл
        with open(self.file, 'w') as f:
            f.write(str(self.n) + "\n")
            for key, value in lst:
                f.write(str(key) + " " + str(value) + "\n")
```

3.1.1.3 Файл «algorithm.py»

```
from graph import *
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt


class Algorithm:
    def __init__(self, file="files/input.txt"):
        # Зчитати дані для побудови графа з файлу
        with open(file, 'r') as f:
            lst = f.readlines()
            n = int(lst[0])
            lst = lst[1:]
            lst = [list(map(int, elem.split())) for elem in lst]

        self.n = n
        self.lst = lst

    def get_neighbours(self, vertex):
        # Отримати список сусідів для вершини
        result = []
        for key, val in self.lst:
            if key == vertex:
                result.append(val)

        return result

    def all_neighbours_not_in_same_color(self, vertex, vertex_colors,
                                         current_color):
        # Перевірити, чи всі сусіди мають інші кольори, від вершини
        neighbours = self.get_neighbours(vertex)
        neighbours_not_in_same_color = [vertex_colors[neighbor] !=
        current_color for neighbor in neighbours]

        if sum(neighbours_not_in_same_color) ==
        len(neighbours_not_in_same_color):
            return True

        return False

    def max_power_vertex(self):
        # Знайти у графі вершину з максимальною кількістю ребер
        elements = [row[0] for row in self.lst]
        max_val = max(set(elements), key=elements.count)

        return max_val

    def try_to_reduce_num_of_colors(self, vertex, vertex_colors,
                                    num_colors):
        # Спробувати зменшити кількість кольорів для кожного сусіда
          поточної вершини
        neighbours = self.get_neighbours(vertex)

        for neighbor in neighbours:
            temp_colors = vertex_colors.copy()
            # Поміняти колір із сусідом
            temp_colors[vertex], temp_colors[neighbor] = \
                temp_colors[neighbor], temp_colors[vertex]
            # Перевірити, чи можна виконати обмін
            if self.all_neighbours_not_in_same_color(
                   neighbor, temp_colors, temp_colors[neighbor]
```

```
            ):
                # Спробувати зменшити кількість кольорів
                new_color = Graph.get_available_color(
                    self.get_neighbours(neighbor),
                    temp_colors,
                    num_colors,
                    vertex_colors[neighbor]
                )
                # Якщо будь-який альтернативний колір підходить, то
                  перефарбувати
                if new_color != -1:
                    temp_colors[neighbor] = new_color
                    vertex_colors = temp_colors.copy()

    return vertex_colors

# Отримати вхідні дані з файлу та створити неорієнтований граф
def create_graph(self):
    graph = nx.Graph()
    lst = Graph.remove_duplicate_edges(self.lst)

    for row in lst:
        graph.add_edge(row[0], row[1])

    return graph

def show_graph(self, vertex_colors):
    # Показати граф
    print("Vertex colors:", vertex_colors)
    print("Number of colors:", len(set(vertex_colors)))
    graph = self.create_graph()

    Graph.draw_graph(graph, vertex_colors)
    plt.show()

def greedy(self):
    current_color = 0
    # Список кольорів для кожної вершини. Примітка: -1 означає
      відсутність кольору
    vertex_colors = [-1 for _ in range(self.n)]

    while sum([val == -1 for val in vertex_colors]) != 0:
        for vertex in range(self.n):
            if vertex_colors[vertex] == -1:
                if self.all_neighbours_not_in_same_color(
                        vertex, vertex_colors, current_color
                    ):
                    vertex_colors[vertex] = current_color
        current_color += 1

    return vertex_colors, current_color

def bees(self):
    # Застосувати жадібний алгоритм
    vertex_colors, num_colors = self.greedy()

    # Намалювати розфарбований граф, створений за допомогою жадібного
      алгоритму
    self.show_graph(vertex_colors)

    # Знайти вершину з максимальним степенем (стартова вершина)
    vertex = self.max_power_vertex()
    # Список вершин для обробки
    nxt = [vertex]
```

```
        counter = 0
        parent = -1
        randomness = 1
        mutation = randomness

        while len(nxt) < 40:
            vertex = nxt[counter]
            neighbours = self.get_neighbours(vertex)
            for neighbor in neighbours:
                if neighbor != parent:
                    nxt.append(neighbor)
                if mutation == 0:
                    nxt.append(np.random.randint(0, self.n+1))
                    mutation = randomness
            parent = vertex
            counter += 1
            mutation -= 1

        for vertex in nxt:
            vertex_colors = self.try_to_reduce_num_of_colors(vertex,
                                        vertex_colors, num_colors)

        # Намалювати розфарбований граф, створений за допомогою класичного
          бджолиного алгоритму
        self.show_graph(vertex_colors)
```

3.1.1.4 Файл «graph.py»

```
 import networkx as nx


class Graph:
    def __init__(self):
        self.x = 0

    @staticmethod
    def get_available_color(neighbours, vertex_colors, num_colors,
                            old_color):
        # Отримати перший доступний колір, якого немає у сусідів
        available_colors = [color for color in range(num_colors)]

        for neighbor in neighbours:
            color = vertex_colors[neighbor]
            if color in available_colors:
                available_colors.remove(color)

        if old_color in available_colors:
            available_colors.remove(old_color)

        if len(available_colors) != 0:
            return available_colors[0]
        return -1

    @staticmethod
    def remove_duplicate_edges(lst):
        # Видаляти повтори, наприклад 1-3, 3-1, або 1-1
        new_list = []
        for key, val in lst:
            if key != val:
                if [key, val] and [val, key] not in new_list:
                    new_list.append([key, val])
        return new_list
```

```
@staticmethod
def draw_graph(graph, vertex_colors):
    # Намалювати граф
    pos = nx.spring_layout(graph)
    colors = ['red', 'blue', 'yellow', 'purple', 'orange', 'black',
              'green', 'grey', 'purple', 'pink', 'brown']
    values = [colors[vertex_colors[node]] for node in graph.nodes()]

    nx.draw(graph, pos, with_labels=True, node_color=values,
            edge_color='black', width=1, alpha=0.7)
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

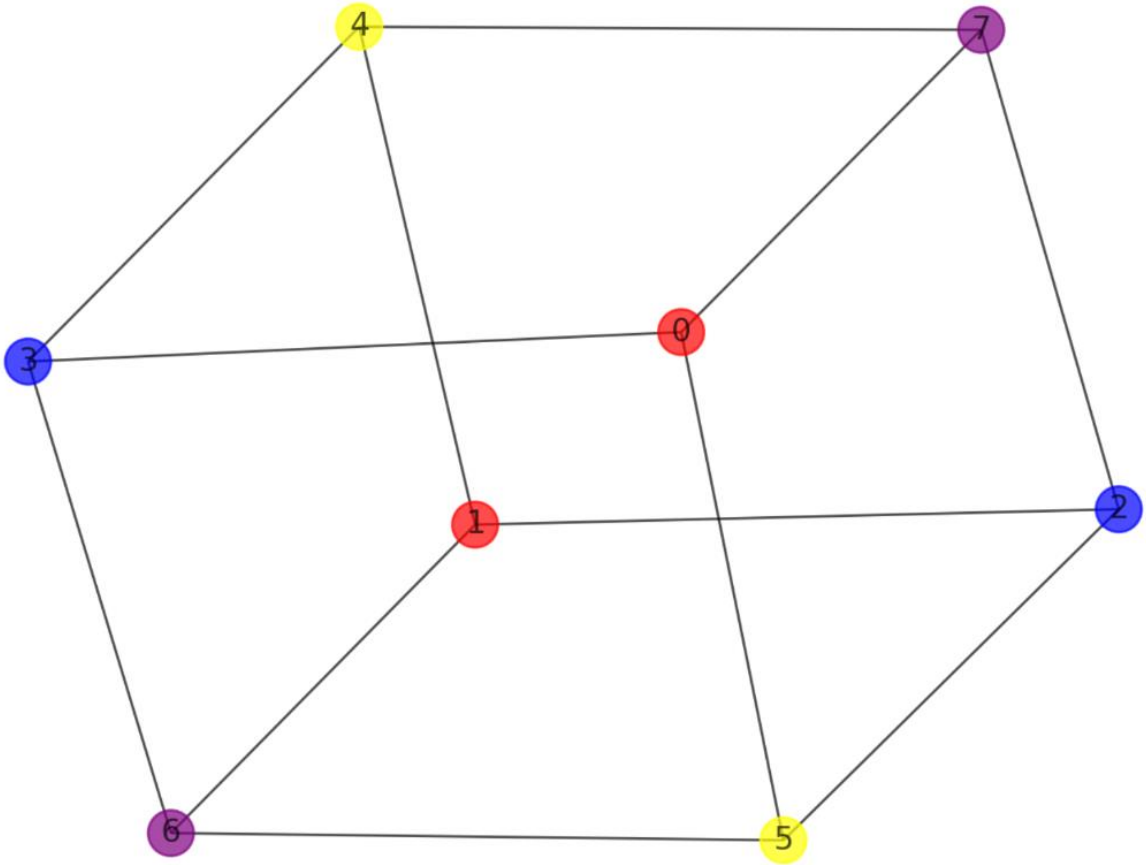

Рисунок 3.1 – Початковий граф, розфарбований жадібним алгоритмом

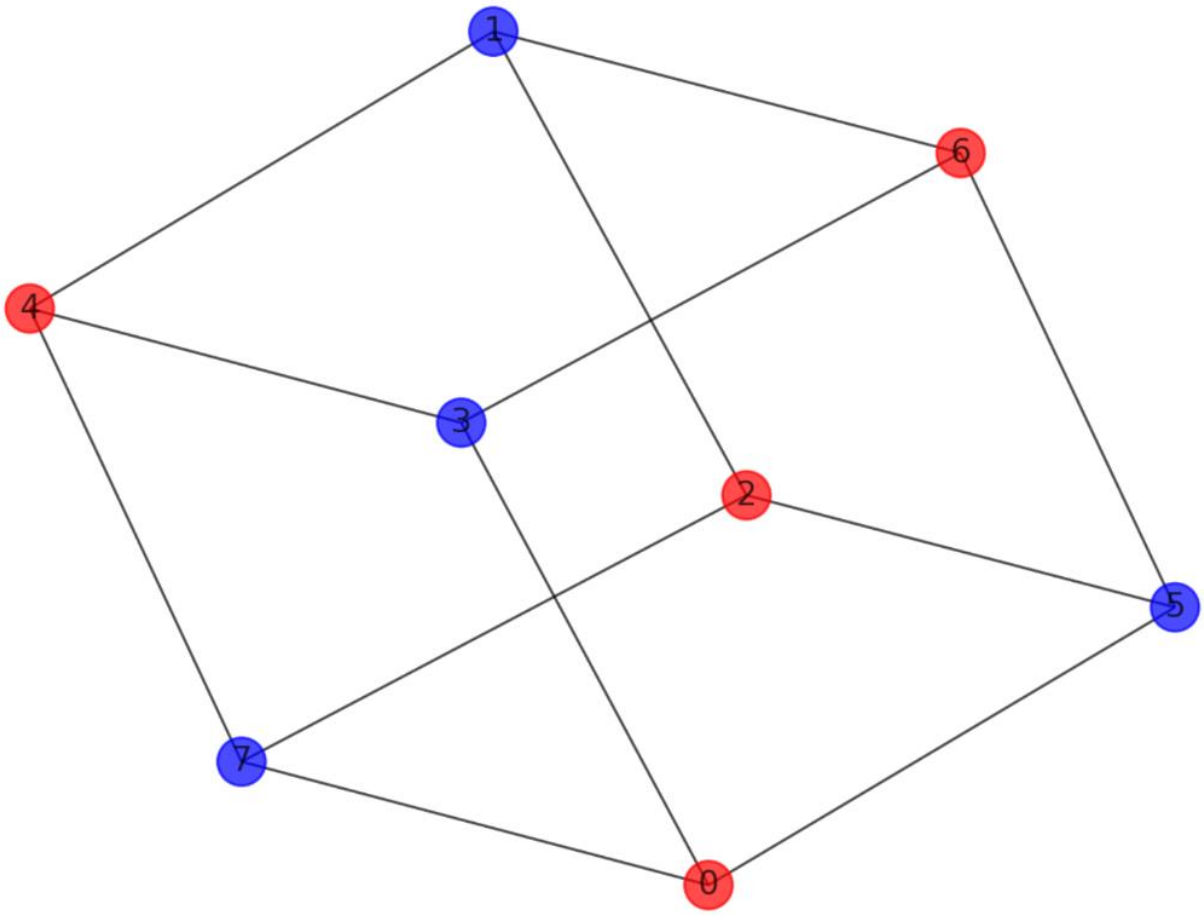

Рисунок 3.2 – Граф, після роботи Класичного бджолиного алгоритму

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	Ітерація	Значення	Ітерація	Значення
0	179	340	158	680	146
20	208	360	160	700	150
40	192	380	163	720	146
60	188	400	164	740	159
80	180	420	149	760	155
100	181	440	169	780	156
120	179	460	152	800	156
140	172	480	160	820	157
160	175	500	152	840	150
180	165	520	155	860	152
200	183	540	151	880	153
220	184	560	147	900	152
240	181	580	140	920	158
260	172	600	150	940	148
280	162	620	140	960	158
300	167	640	144	980	152
320	171	660	141	1000	150

3.2.2 Графік залежності розв'язку від числа ітерацій

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

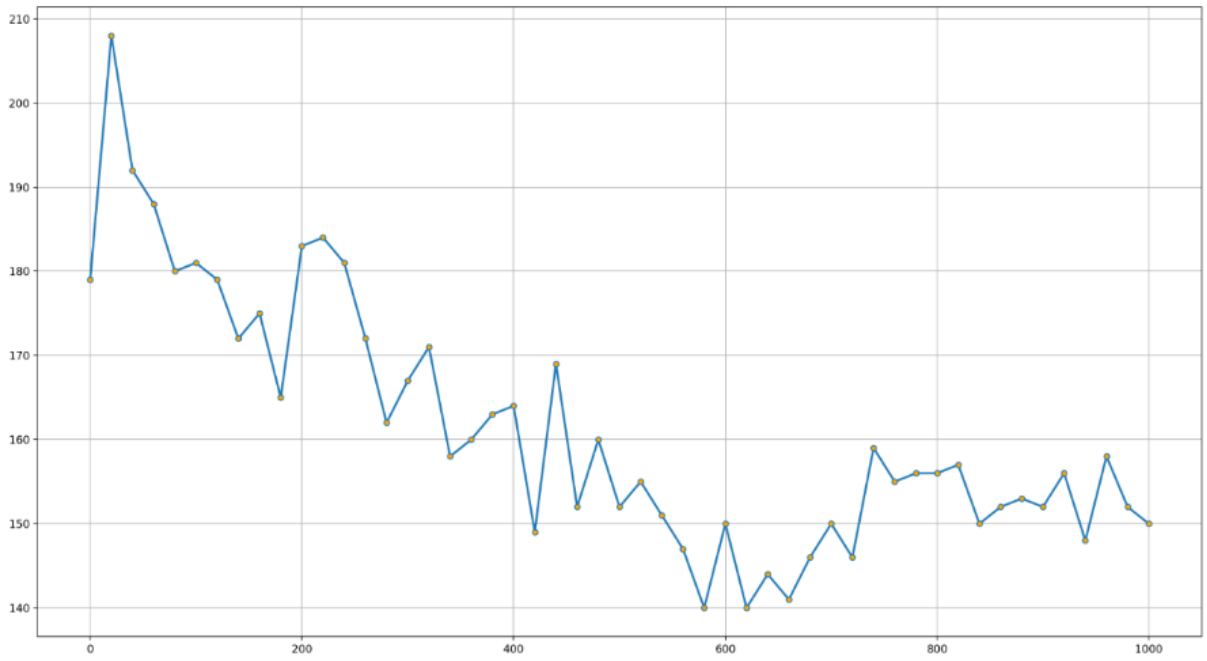

Рисунок 3.3 – Графік залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи я реалізував Класичний бджолиний алгоритм. Для невеликих графів, де Жадібний алгоритм показує себе не із найкращої сторони, добре видно переваги Бджолиного алгоритму, попри те, що він працює набагато повільніше.

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%.