**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 Йолкін Даніїл Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

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 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_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

import random  
  
NUMBER\_OF\_NODES = 300  
NUMBER\_OF\_ITERATIONS = 1000  
NUMBER\_OF\_WORKING\_BEES = 55  
NUMBER\_OF\_RECONNAISSANCE\_BEES = 5  
MAX\_NUMBER\_OF\_CONNECTIONS = 30  
MIN\_NUMBER\_OF\_CONNECTIONS = 1  
NODES\_CLEARED = []  
  
  
def generate\_graph():  
 graph = [[0 for \_ in range(0, NUMBER\_OF\_NODES)] for \_ in range(0, NUMBER\_OF\_NODES)]  
 for i in range(0, NUMBER\_OF\_NODES):  
 for j in range(0, NUMBER\_OF\_NODES):  
 value = random.randint(0, 1)  
 if i != j and not graph[i][j] and value:  
 count\_x\_1 = 0  
 count\_x\_2 = 0  
 count\_y\_1 = 0  
 count\_y\_2 = 0  
 for k in range(0, NUMBER\_OF\_NODES):  
 if graph[i][k]:  
 count\_x\_1 += 1  
 if graph[k][j]:  
 count\_y\_1 += 1  
 if graph[j][k]:  
 count\_x\_1 += 1  
 if graph[k][i]:  
 count\_x\_1 += 1  
 if count\_x\_1 >= MAX\_NUMBER\_OF\_CONNECTIONS or count\_x\_2 >= MAX\_NUMBER\_OF\_CONNECTIONS \  
 or count\_y\_1 >= MAX\_NUMBER\_OF\_CONNECTIONS or count\_y\_2 >= MAX\_NUMBER\_OF\_CONNECTIONS:  
 continue  
 else:  
 graph[i][j] = value  
 graph[j][i] = value  
 return graph  
  
  
def print\_matrix(matrix):  
 for i in range(0, NUMBER\_OF\_NODES):  
 string = ""  
 for j in range(0, NUMBER\_OF\_NODES):  
 string += str(matrix[i][j]) + " "  
 print(string)  
 return 0  
  
  
def print\_hi(name):  
 # Use a breakpoint in the code line below to debug your script.  
 print(f'Hi, {name}') # Press ⌘F8 to toggle the breakpoint.  
  
  
def ABC(graph):  
 number\_of\_iterations = 0  
 best\_current\_coloring = [0 for \_ in range(0, NUMBER\_OF\_NODES)]  
 flag = True  
 while number\_of\_iterations != NUMBER\_OF\_ITERATIONS:  
  
 # Copy the best found coloring  
 current\_coloring = list(best\_current\_coloring)  
  
 # Randomly select nodes  
 nodes\_selected = selecting\_nodes()  
  
 # Calculate nectar values  
 nectar\_values = count\_nectar(nodes\_selected, graph)  
  
 # Clear nectar from nodes that were already cleared  
 for node in nodes\_selected:  
 if node in NODES\_CLEARED:  
 nectar\_values[nodes\_selected.index(node)] = 0  
  
 # Rearrange nodes  
 temp\_node\_storage = []  
 nectar\_values\_copy = list(nectar\_values)  
 for i in range(0, len(nodes\_selected)):  
 max\_value = max(nectar\_values\_copy)  
 index = nectar\_values.index(max\_value)  
 while nodes\_selected[index] in temp\_node\_storage:  
 index += 1  
 temp\_node\_storage.append(nodes\_selected[index])  
 nectar\_values\_copy.pop(nectar\_values\_copy.index(max\_value))  
 nodes\_selected = list(temp\_node\_storage)  
  
 # Assign bees in proportion  
 bees = []  
 bees\_used = 0  
 nectar\_values.sort()  
 counter = 0  
 for i in range(0, len(nectar\_values)):  
 if nectar\_values[i] != 0:  
 bees\_count = int(NUMBER\_OF\_WORKING\_BEES \* nectar\_values[i])  
 bees\_used += bees\_count  
 bees.append(bees\_count)  
 else:  
 counter += 1  
 end\_counter = counter  
 while counter:  
 bees.append(int((NUMBER\_OF\_WORKING\_BEES - bees\_used) / end\_counter))  
 counter -= 1  
  
 # print(sum(bees))  
 # print(bees)  
  
 # Check each node according to nectar value  
 for node in nodes\_selected:  
 node\_connections = []  
  
 # Find all adjacent nodes to the node  
 for i in range(0, NUMBER\_OF\_NODES):  
 if graph[node][i]:  
 node\_connections.append(i)  
  
 # Crop the list of connection according to the number of allocated bees  
 if bees[nodes\_selected.index(node)] < len(node\_connections):  
 node\_connections = node\_connections[:bees[nodes\_selected.index(node)]-1:]  
  
 # Check each node adjacent to current node  
 for sub\_node in node\_connections:  
 sub\_node\_connections = []  
  
 # Find all adjacent nodes to the node  
 for j in range(0, NUMBER\_OF\_NODES):  
 if graph[sub\_node][j] and j != node:  
 sub\_node\_connections.append(j)  
  
 # Determine all colors used by adjacent nodes  
 colors\_used = []  
 for sub\_node\_connection in sub\_node\_connections:  
 colors\_used.append(current\_coloring[sub\_node\_connection])  
  
 # Paint adjacent node in color that wasn't used by adjacent nodes  
 color\_selected = 0  
 for i in range(1, NUMBER\_OF\_NODES):  
 if i not in colors\_used:  
 color\_selected = i  
 break  
 current\_coloring[sub\_node] = color\_selected  
  
 # Determine all colors used by adjacent nodes  
 sub\_node\_colors = []  
 for sub\_node in node\_connections:  
 sub\_node\_colors.append(current\_coloring[sub\_node])  
  
 # Paint node in color that wasn't used by adjacent nodes  
 for i in range(1, NUMBER\_OF\_NODES):  
 if i not in sub\_node\_colors:  
 current\_coloring[node] = i  
 break  
  
 # Check whether new solution is better than previously considered best  
 if not best\_current\_coloring.\_\_contains\_\_(0):  
 if max(current\_coloring) <= max(best\_current\_coloring):  
 best\_current\_coloring = list(current\_coloring)  
 else:  
 best\_current\_coloring = list(current\_coloring)  
  
 # Indicate that nectar should be cleared from the node  
 NODES\_CLEARED.append(node)  
  
 # Check whether all nodes were painted  
 if not best\_current\_coloring.\_\_contains\_\_(0) and flag:  
 print("All nodes painted!")  
 flag = False  
 number\_of\_iterations += 1  
  
 # Print the result of every 20th iteration  
 if number\_of\_iterations % 20 == 0:  
 print("Number of iteration: " + str(number\_of\_iterations))  
 print("Number of colors: " + str(max(best\_current\_coloring)))  
  
 # Print the result  
 print(best\_current\_coloring)  
  
 return best\_current\_coloring  
  
  
def selecting\_nodes():  
 nodes\_selected = []  
 while len(nodes\_selected) != NUMBER\_OF\_RECONNAISSANCE\_BEES:  
 node = random.randint(0, NUMBER\_OF\_NODES-1)  
 if node not in nodes\_selected:  
 nodes\_selected.append(node)  
 return nodes\_selected  
  
  
def count\_nectar(nodes: list[int], graph: list[list[int]]):  
 nectar\_values = []  
 for node in nodes:  
 current\_nectar = 0  
 for value in graph[node]:  
 if value:  
 current\_nectar += 1  
 nectar\_values.append(current\_nectar)  
 nectar\_sum = sum(nectar\_values)  
 for i in range(0, len(nectar\_values)):  
 nectar\_values[i] = nectar\_values[i]/nectar\_sum  
 return nectar\_values  
  
  
# Press the green button in the gutter to run the script.  
if \_\_name\_\_ == '\_\_main\_\_':  
 matrix = generate\_graph()  
 print\_matrix(matrix)  
 ABC(matrix)

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

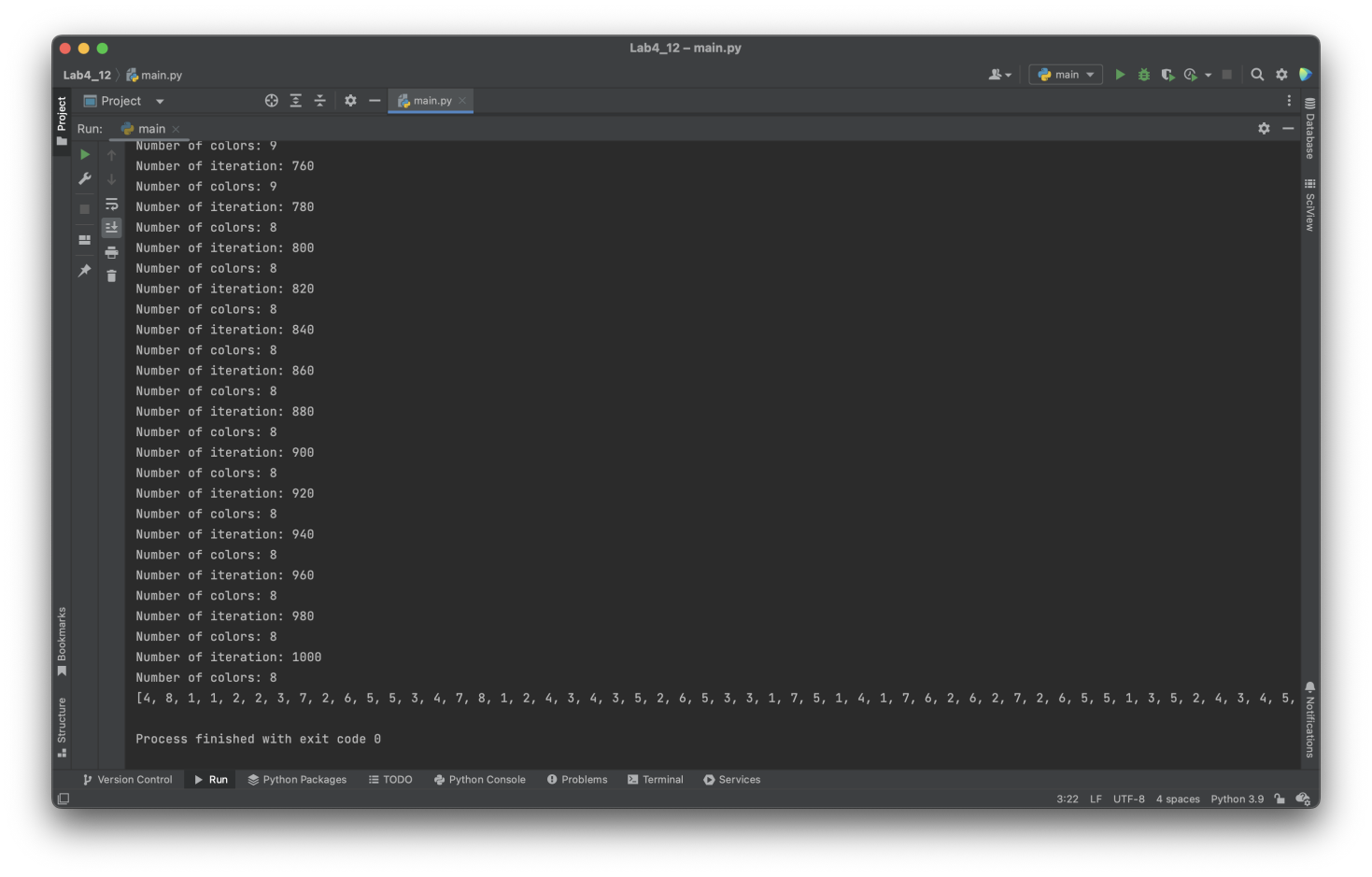


Рисунок 3.1 – Приклад роботи програми

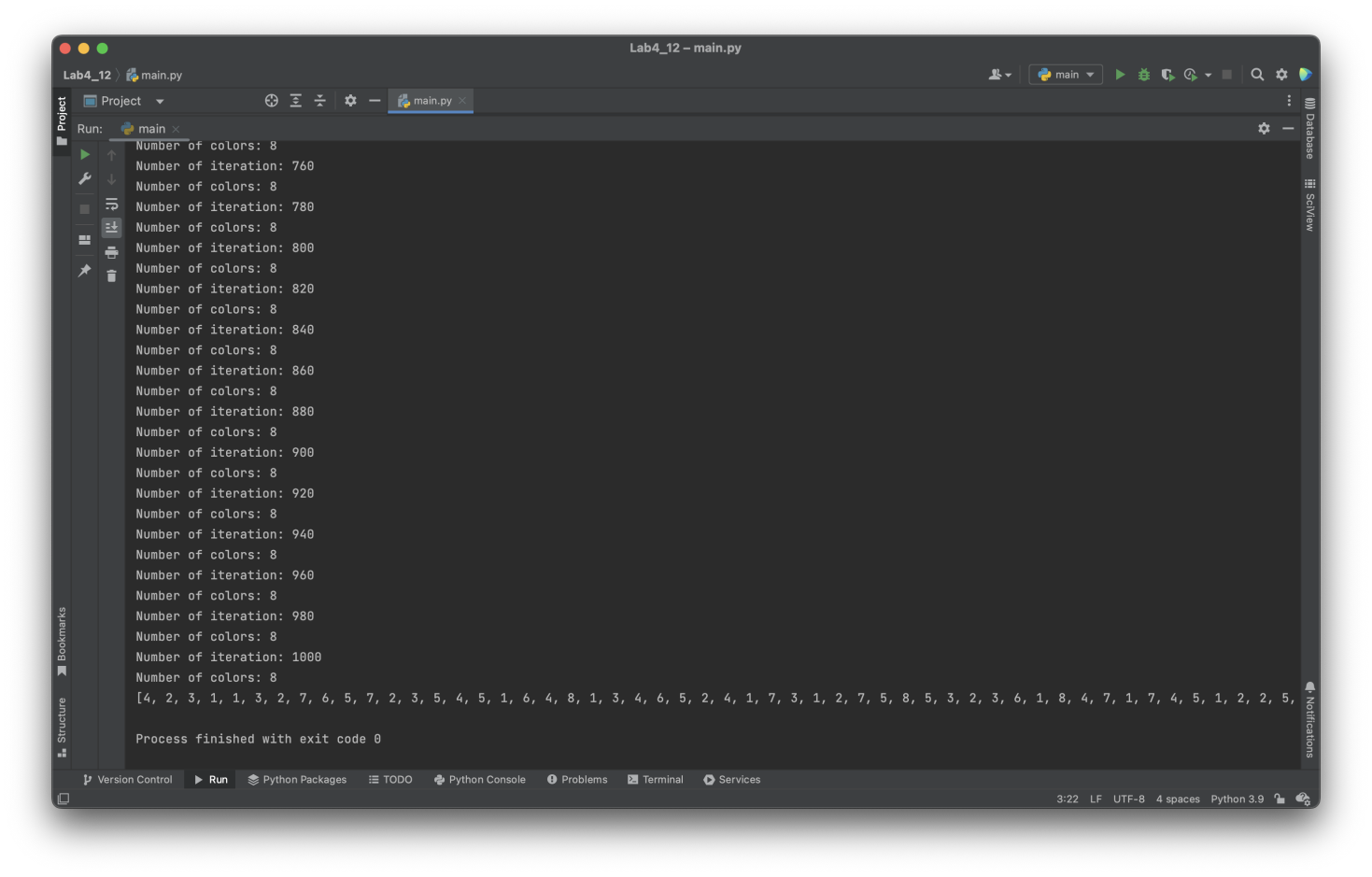


Рисунок 3.2 – Приклад роботи програми

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

Number of iteration: 20

Number of colors: 10

All nodes painted!

Number of iteration: 40

Number of colors: 11

Number of iteration: 60

Number of colors: 9

Number of iteration: 80

Number of colors: 9

Number of iteration: 100

Number of colors: 9

Number of iteration: 120

Number of colors: 9

Number of iteration: 140

Number of colors: 9

Number of iteration: 160

Number of colors: 9

Number of iteration: 180

Number of colors: 9

Number of iteration: 200

Number of colors: 9

Number of iteration: 220

Number of colors: 9

Number of iteration: 240

Number of colors: 9

Number of iteration: 260

Number of colors: 9

Number of iteration: 280

Number of colors: 9

Number of iteration: 300

Number of colors: 9

Number of iteration: 320

Number of colors: 9

Number of iteration: 340

Number of colors: 9

Number of iteration: 360

Number of colors: 9

Number of iteration: 380

Number of colors: 9

Number of iteration: 400

Number of colors: 9

Number of iteration: 420

Number of colors: 9

Number of iteration: 440

Number of colors: 9

Number of iteration: 460

Number of colors: 9

Number of iteration: 480

Number of colors: 9

Number of iteration: 500

Number of colors: 9

Number of iteration: 520

Number of colors: 9

Number of iteration: 540

Number of colors: 9

Number of iteration: 560

Number of colors: 9

Number of iteration: 580

Number of colors: 9

Number of iteration: 600

Number of colors: 9

Number of iteration: 620

Number of colors: 9

Number of iteration: 640

Number of colors: 9

Number of iteration: 660

Number of colors: 9

Number of iteration: 680

Number of colors: 9

Number of iteration: 700

Number of colors: 9

Number of iteration: 720

Number of colors: 9

Number of iteration: 740

Number of colors: 9

Number of iteration: 760

Number of colors: 9

Number of iteration: 780

Number of colors: 8

Number of iteration: 800

Number of colors: 8

Number of iteration: 820

Number of colors: 8

Number of iteration: 840

Number of colors: 8

Number of iteration: 860

Number of colors: 8

Number of iteration: 880

Number of colors: 8

Number of iteration: 900

Number of colors: 8

Number of iteration: 920

Number of colors: 8

Number of iteration: 940

Number of colors: 8

Number of iteration: 960

Number of colors: 8

Number of iteration: 980

Number of colors: 8

Number of iteration: 1000

Number of colors: 8

### Графіки залежності розв'язку від числа ітерацій

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

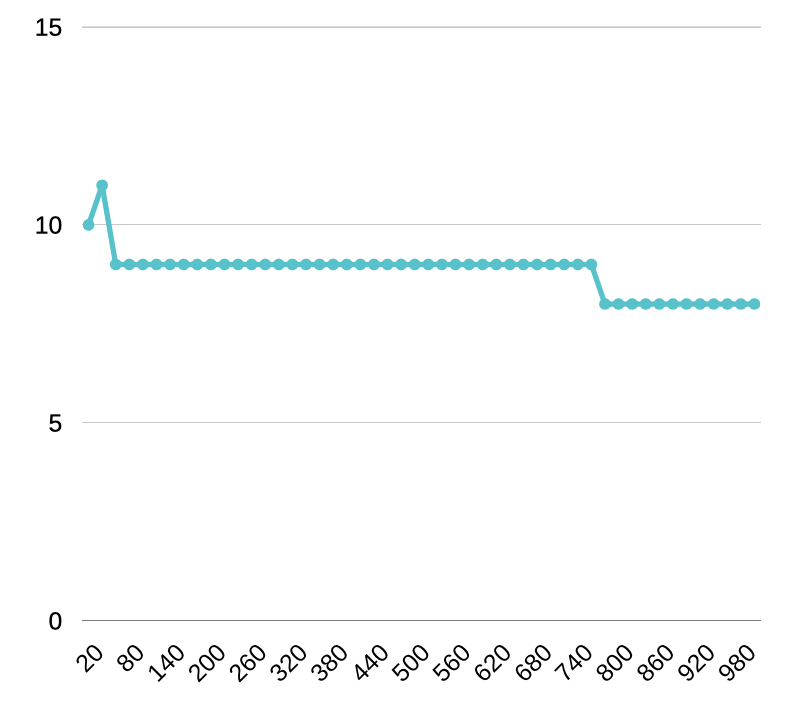


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи було розроблено бджолиний алгоритм для розфарбування графа на 300 вершин.

Було проведено тестування алгоритму. Протягом тестування на 1000 ітерацій отримали хроматичне число 8 для першого випадкового графа на 300 вершин та хроматичне число 8 для другого випадкового графа на 300 вершин.

Також було наведено значення цільової функції кожні 20 ітерацій та побудовано графік залежності.

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%.