

**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-15 Мешков Андрій Ігорович

(шифр, прізвище, ім'я, по батькові)

**Перевірів**

Головченко М.Н.

(прізвище, ім'я, по батькові)

Київ 2022

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ .....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ.....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ .....</b>	<b>5</b>
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ .....	5
3.1.1	<i>Вихідний код .....</i>	<i>5</i>
3.1.2	<i>Приклади роботи.....</i>	<i>7</i>
3.2	ТЕСТУВАННЯ АЛГОРИТМУ.....	10
3.2.1	<i>Значення цільової функції зі збільшенням кількості ітерацій .</i>	<i>10</i>
3.2.2	<i>Графіки залежності розв'язку від числа ітерацій .....</i>	<i>11</i>
	<b>ВИСНОВОК.....</b>	<b>12</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ.....</b>	<b>13</b>

## 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
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).

### 3.1 Програмна реалізація алгоритму

#### 3.1.1 Вихідний код

```

from random import shuffle
from queue import PriorityQueue
import networkx as nx
from collections import defaultdict

class Bee:
    def __init__(self, coloring, f):
        self.coloring = coloring
        self.f = f

    def __lt__(self, other):
        return self.f.__lt__(other.f)

    def __repr__(self):
        return f"\nFunction: {self.f} Coloring: {self.coloring}"

def generateGraph(ver, d):
    g = nx.watts_strogatz_graph(ver, d, 1)
    graph = {}
    for edge in g.edges:
        u, v = edge
        u += 1
        v += 1
        if u not in graph:
            graph[u] = []
        if v not in graph:
            graph[v] = []
        graph[u].append(v)
        graph[v].append(u)
    return graph

def findBestPlots(plots, n):
    best = []
    for i in range(n):
        best.append(plots.get())
    return best

def colorGraph(graph, colors):
    coloring = {vertex: 0 for vertex in graph.keys()}
    graph = list(graph.items())
    shuffle(graph)
    colorN = 0
    for vertex, neighbours in graph:
        for color in colors:
            if not neighbourhood(color, neighbours, coloring):
                coloring[vertex] = color
                if color > colorN:
                    colorN += 1
                break
    return coloring, colorN

def generatePlots(graph, plots, n, colors):
    while plots.qsize() < n:
        plot = Bee(*colorGraph(graph, colors))
        if plot not in plots.queue:

```

```

        plots.put(plot)

def neighbourhood(color, neighbours, coloring):
    if neighbours:
        for neighbour in neighbours:
            if neighbour in coloring and coloring[neighbour] == color:
                return True
    return False

def discover(plot, graph, foragersN):
    queue = []
    for node, neighbours in sorted(list(graph.items()), key=lambda x:
len(x[1]), reverse=True):
        if len(queue) >= foragersN:
            break
        for neighbour in neighbours:
            queue.append((node, neighbour))

    results = []

    for node, neighbour in queue:
        recoloring = dict(plot.coloring)
        recoloring[neighbour], recoloring[node] = recoloring[node],
recoloring[neighbour]

        if neighbourhood(recoloring[node], graph[node], recoloring) or \
neighbourhood(recoloring[neighbour], graph[neighbour],
recoloring):
            continue
        else:
            for color in range(1, plot.f + 1):
                if not neighbourhood(color, graph[neighbour], recoloring):
                    recoloring[neighbour] = color
                    results.append(Bee(recoloring,
len(set(recoloring.values()))))

    return min(results, key=lambda x: x.f) if results else plot

def foragerSearch(graph, newPlots, plots, n):
    for plot in plots:
        plot = discover(plot, graph, n)
        newPlots.put(plot)

def addBestPlots(plots, bestPlots):
    for plot in bestPlots:
        plots.put(plot)

def main():
    beesN = 60
    scoutsN = 5
    foragerN = beesN - scoutsN
    bestPlotScoutsN = 2
    randomScoutsN = scoutsN - bestPlotScoutsN
    vertices = 300
    degree = 50
    bestForagerN = 30
    randomForagerN = 5

    colors= []
    used_colors = []
    for c in range(50):
        colors.append(c)
        used_colors.append(c)

```

```

graph = generateGraph(vertices, degree)
plots = PriorityQueue()
generatePlots(graph, plots, scoutsN, colors)

for i in range(1001):
    if not i % 20:
        best = plots.get()
        print(f"Iteration {i}, min f: {best.f}")
        plots.put(best)

    bestPlots = findBestPlots(plots, bestPlotScoutsN)
    randomPlots = plots.queue

    plots = PriorityQueue()
    foragerSearch(graph, plots, bestPlots, bestForagerN)
    foragerSearch(graph, plots, randomPlots, randomForagerN)

    bestPlots = findBestPlots(plots, bestPlotScoutsN)
    plots = PriorityQueue()
    generatePlots(graph, plots, randomScoutsN, colors)
    addBestPlots(plots, bestPlots)

    best = plots.get()
    coloring = {k: v for k, v in sorted(best.coloring.items(), key=lambda
item: item[0])}
    print(f"Best coloring function: {best.f}\nColoring: {coloring}")

if __name__ == "__main__":
    main()

```

### 3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```

Iteration 0, min f: 19
Iteration 20, min f: 18
Iteration 40, min f: 18
Iteration 60, min f: 18
Iteration 80, min f: 18
Iteration 100, min f: 18
Iteration 120, min f: 18
Iteration 140, min f: 18
Iteration 160, min f: 18
Iteration 180, min f: 18
Iteration 200, min f: 18
Iteration 220, min f: 18
Iteration 240, min f: 18
Iteration 260, min f: 18
Iteration 280, min f: 18
Iteration 300, min f: 18
Iteration 320, min f: 18
Iteration 340, min f: 18
Iteration 360, min f: 18
Iteration 380, min f: 18
Iteration 400, min f: 18
Iteration 420, min f: 18
Iteration 440, min f: 18
Iteration 460, min f: 18
Iteration 480, min f: 18
Iteration 500, min f: 18
Iteration 520, min f: 18
Iteration 540, min f: 18
Iteration 560, min f: 18
Iteration 580, min f: 18
Iteration 600, min f: 18
Iteration 620, min f: 18
Iteration 640, min f: 18
Iteration 660, min f: 18
Iteration 680, min f: 18
Iteration 700, min f: 18
Iteration 720, min f: 18
Iteration 740, min f: 17
Iteration 760, min f: 17
Iteration 780, min f: 17
Iteration 800, min f: 17
Iteration 820, min f: 17
Iteration 840, min f: 17
Iteration 860, min f: 17
Iteration 880, min f: 17
Iteration 900, min f: 17
Iteration 920, min f: 17
Iteration 940, min f: 17
Iteration 960, min f: 17
Iteration 980, min f: 17
Iteration 1000, min f: 17

```

```

Best coloring function: 17
Coloring: {1: 3, 2: 11, 3: 5, 4: 9, 5: 1, 6: 11, 7: 11, 8: 7, 9: 13, 10:
8, 11: 12, 12: 0, 13: 13, 14: 14, 15: 3, 16: 3, 17: 13, 18: 8, 19: 10,
20: 10, 21: 14, 22: 15, 23: 1, 24: 12, 25: 6, 26: 13, 27: 9, 28: 8, 29:
2, 30: 7, 31: 9, 32: 10, 33: 12, 34: 0, 35: 5, 36: 11, 37: 15, 38: 1, 39
: 15, 40: 1, 41: 1, 42: 8, 43: 6, 44: 8, 45: 9, 46: 13, 47: 2, 48: 17, 4
9: 9, 50: 13, 51: 11, 52: 9, 53: 14, 54: 3, 55: 6, 56: 5, 57: 2, 58: 15,
59: 11, 60: 11, 61: 2, 62: 12, 63: 5, 64: 9, 65: 6, 66: 2, 67: 5, 68: 1
5, 69: 8, 70: 9, 71: 11, 72: 7, 73: 16, 74: 17, 75: 7, 76: 8, 77: 3, 78:
9, 79: 4, 80: 7, 81: 12, 82: 6, 83: 5, 84: 11, 85: 2, 86: 1, 87: 15, 88
: 7, 89: 12, 90: 9, 91: 7, 92: 2, 93: 2, 94: 11, 95: 11, 96: 4, 97: 13,
98: 2, 99: 4, 100: 2, 101: 6, 102: 10, 103: 3, 104: 6, 105: 9, 106: 1, 1
07: 11, 108: 17, 109: 8, 110: 11, 111: 4, 112: 2, 113: 13, 114: 2, 115:
12, 116: 2, 117: 8, 118: 8, 119: 16, 120: 1, 121: 4, 122: 9, 123: 14, 12
4: 14, 125: 9, 126: 10, 127: 16, 128: 5, 129: 5, 130: 5, 131: 2, 132: 3,
133: 14, 134: 5, 135: 14, 136: 1, 137: 2, 138: 4, 139: 3, 140: 4, 141:
12, 142: 3, 143: 7, 144: 6, 145: 12, 146: 4, 147: 2, 148: 1, 149: 7, 150
: 3, 151: 10, 152: 2, 153: 2, 154: 6, 155: 12, 156: 11, 157: 5, 158: 17,
159: 15, 160: 7, 161: 1, 162: 15, 163: 4, 164: 4, 165: 0, 166: 11, 167:
16, 168: 13, 169: 5, 170: 6, 171: 16, 172: 2, 173: 6, 174: 1, 175: 17,
176: 12, 177: 12, 178: 5, 179: 9, 180: 6, 181: 2, 182: 10, 183: 4, 184:
8, 185: 5, 186: 10, 187: 10, 188: 1, 189: 16, 190: 13, 191: 7, 192: 9, 1
93: 11, 194: 1, 195: 0, 196: 16, 197: 10, 198: 7, 199: 8, 200: 14, 201:
16, 202: 2, 203: 7, 204: 13, 205: 8, 206: 8, 207: 13, 208: 17, 209: 2, 2
10: 14, 211: 9, 212: 1, 213: 7, 214: 3, 215: 14, 216: 14, 217: 15, 218:
10, 219: 7, 220: 15, 221: 5, 222: 2, 223: 9, 224: 6, 225: 7, 226: 2, 227
: 11, 228: 4, 229: 15, 230: 4, 231: 7, 232: 15, 233: 14, 234: 3, 235: 8,
236: 9, 237: 14, 238: 5, 239: 10, 240: 10, 241: 6, 242: 1, 243: 9, 244:
4, 245: 3, 246: 3, 247: 3, 248: 10, 249: 10, 250: 1, 251: 6, 252: 7, 25
3: 6, 254: 4, 255: 9, 256: 3, 257: 1, 258: 6, 259: 5, 260: 5, 261: 13, 2
62: 8, 263: 8, 264: 6, 265: 3, 266: 12, 267: 17, 268: 1, 269: 16, 270: 3
, 271: 12, 272: 1, 273: 6, 274: 15, 275: 1, 276: 7, 277: 5, 278: 12, 279
: 4, 280: 6, 281: 3, 282: 12, 283: 4, 284: 1, 285: 10, 286: 3, 287: 4, 2
88: 8, 289: 13, 290: 15, 291: 16, 292: 8, 293: 8, 294: 1, 295: 9, 296: 4
, 297: 4, 298: 10, 299: 17, 300: 2}

```

Рисунок 3.1 – Приклад роботи програми



```

2120: /python /tes/ lib/python/debugpy
Iteration 0, min f: 13
Iteration 20, min f: 12
Iteration 40, min f: 12
Iteration 60, min f: 12
Iteration 80, min f: 12
Iteration 100, min f: 12
Iteration 120, min f: 12
Iteration 140, min f: 12
Iteration 160, min f: 12
Iteration 180, min f: 12
Iteration 200, min f: 12
Iteration 220, min f: 12
Iteration 240, min f: 12
Iteration 260, min f: 12
Iteration 280, min f: 12
Iteration 300, min f: 12
Iteration 320, min f: 12
Iteration 340, min f: 12
Iteration 360, min f: 12
Iteration 380, min f: 12
Iteration 400, min f: 12
Iteration 420, min f: 12
Iteration 440, min f: 12
Iteration 460, min f: 12
Iteration 480, min f: 12
Iteration 500, min f: 12
Iteration 520, min f: 12
Iteration 540, min f: 12
Iteration 560, min f: 12
Iteration 580, min f: 12
Iteration 600, min f: 12
Iteration 620, min f: 12
Iteration 640, min f: 12
Iteration 660, min f: 12
Iteration 680, min f: 12
Iteration 700, min f: 12
Iteration 720, min f: 12
Iteration 740, min f: 12
Iteration 760, min f: 12
Iteration 780, min f: 12
Iteration 800, min f: 12
Iteration 820, min f: 12
Iteration 840, min f: 12
Iteration 860, min f: 12
Iteration 880, min f: 12
Iteration 900, min f: 12
Iteration 920, min f: 12
Iteration 940, min f: 12
Iteration 960, min f: 12
Iteration 980, min f: 12
Iteration 1000, min f: 12

```

```

Iteration 1000, min f: 12
Best coloring function: 12
Coloring: {1: 4, 2: 2, 3: 1, 4: 7, 5: 2, 6: 7, 7: 3, 8: 7, 9: 9, 10: 5, 11: 8,
12: 4, 13: 1, 14: 7, 15: 9, 16: 9, 17: 8, 18: 2, 19: 2, 20: 9, 21: 3, 22: 3, 23
: 7, 24: 3, 25: 7, 26: 1, 27: 8, 28: 3, 29: 4, 30: 6, 31: 4, 32: 6, 33: 0, 34:
4, 35: 10, 36: 10, 37: 3, 38: 11, 39: 8, 40: 6, 41: 5, 42: 2, 43: 12, 44: 8, 45
: 1, 46: 1, 47: 4, 48: 8, 49: 1, 50: 2, 51: 4, 52: 7, 53: 5, 54: 6, 55: 5, 56:
2, 57: 7, 58: 1, 59: 6, 60: 1, 61: 1, 62: 5, 63: 4, 64: 10, 65: 6, 66: 9, 67: 1
, 68: 2, 69: 9, 70: 4, 71: 6, 72: 6, 73: 1, 74: 2, 75: 8, 76: 2, 77: 10, 78: 1,
79: 3, 80: 10, 81: 9, 82: 3, 83: 10, 84: 6, 85: 10, 86: 2, 87: 4, 88: 2, 89: 2
, 90: 2, 91: 8, 92: 9, 93: 4, 94: 8, 95: 3, 96: 3, 97: 0, 98: 11, 99: 4, 100: 1
, 101: 5, 102: 12, 103: 1, 104: 5, 105: 1, 106: 2, 107: 10, 108: 0, 109: 2, 110
: 4, 111: 2, 112: 0, 113: 6, 114: 6, 115: 8, 116: 3, 117: 0, 118: 1, 119: 8, 12
0: 7, 121: 11, 122: 7, 123: 6, 124: 1, 125: 8, 126: 4, 127: 0, 128: 1, 129: 1,
130: 3, 131: 5, 132: 6, 133: 12, 134: 4, 135: 11, 136: 5, 137: 9, 138: 8, 139:
1, 140: 2, 141: 3, 142: 10, 143: 3, 144: 8, 145: 6, 146: 11, 147: 3, 148: 0, 14
9: 8, 150: 4, 151: 6, 152: 9, 153: 3, 154: 1, 155: 1, 156: 5, 157: 1, 158: 6, 1
59: 2, 160: 0, 161: 3, 162: 1, 163: 4, 164: 7, 165: 7, 166: 6, 167: 4, 168: 10,
169: 1, 170: 7, 171: 8, 172: 3, 173: 3, 174: 10, 175: 8, 176: 8, 177: 4, 178:
11, 179: 0, 180: 5, 181: 10, 182: 2, 183: 1, 184: 7, 185: 1, 186: 6, 187: 5, 18
8: 2, 189: 7, 190: 7, 191: 1, 192: 5, 193: 7, 194: 12, 195: 7, 196: 6, 197: 1,
198: 7, 199: 3, 200: 1, 201: 4, 202: 3, 203: 5, 204: 7, 205: 3, 206: 11, 207: 9
, 208: 11, 209: 1, 210: 3, 211: 6, 212: 6, 213: 10, 214: 11, 215: 2, 216: 6, 21
7: 9, 218: 8, 219: 5, 220: 7, 221: 10, 222: 4, 223: 5, 224: 8, 225: 3, 226: 6,
227: 0, 228: 4, 229: 3, 230: 10, 231: 5, 232: 1, 233: 9, 234: 8, 235: 8, 236: 1
, 237: 7, 238: 12, 239: 9, 240: 10, 241: 4, 242: 5, 243: 4, 244: 5, 245: 4, 246
: 2, 247: 11, 248: 1, 249: 2, 250: 2, 251: 10, 252: 7, 253: 7, 254: 8, 255: 11,
256: 2, 257: 2, 258: 6, 259: 3, 260: 4, 261: 0, 262: 2, 263: 1, 264: 9, 265: 9
, 266: 6, 267: 5, 268: 3, 269: 5, 270: 11, 271: 2, 272: 8, 273: 6, 274: 8, 275:
5, 276: 3, 277: 4, 278: 4, 279: 2, 280: 8, 281: 6, 282: 2, 283: 7, 284: 6, 285
: 4, 286: 12, 287: 4, 288: 5, 289: 8, 290: 0, 291: 2, 292: 5, 293: 11, 294: 9,
295: 5, 296: 1, 297: 2, 298: 8, 299: 2, 300: 4}

```

Рисунок 3.2 – Приклад роботи програми

## 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 функції	Ітерація	Значення функції
0	19	520	18
20	18	540	18
40	18	560	18
60	18	580	18
80	18	600	18
100	18	620	18
120	18	640	18
140	18	660	18
160	18	680	18
180	18	700	18
200	18	720	18
220	18	740	17
240	18	760	17
260	18	780	17
280	18	800	17
300	18	820	17
320	18	840	17
340	18	860	17
360	18	880	17
380	18	900	17
400	18	920	17
420	18	940	17
440	18	960	17
460	18	980	17
480	18	1000	17
500	18		

### 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 розроблено алгоритм вирішення задачі розфарбування графа класичним бджолиним алгоритмом та виконано його програмну реалізацію на мові програмування Python. В алгоритмі використовується 5 розвідників, з ділянок, що вони знайшли, обирається 2 найперспективніші для подальшого пошуку в околиці 30-ма фуражирами кожен, пошук в околиці інших виконується 5-ма фуражирами. Фуражир перевіряє можливість заміни кольорів двох суміжних вершин, і зміни кольору однієї з них на кращій. В якості евристики було вирішено починати з найбільш ступеневої вершини.

Було зафіксовано якість виконання алгоритму кожні 20 ітерації впродовж 1000 ітерацій. Побудовано графік залежності якості від кількості ітерацій. За допомогою евристичного алгоритму вдалось покращити якість з 19 до 17 кольорів.

Отже, класичний бджолиний алгоритм може використовуватися в якості метаевристичного алгоритму розв'язання задачі розфарбування графу для знаходження найбільш оптимального розв'язку.

## 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%;
- висновки – 5%.