

**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 Басараб Олег Андрійович  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

Сопов О. О.  
(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>10</b>
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ .....	10
3.1.1	<i>Вихідний код.....</i>	<i>10</i>
3.1.2	<i>Приклади роботи.....</i>	<i>13</i>
3.2	ТЕСТУВАННЯ АЛГОРИТМУ .....	14
3.2.1	<i>Значення цільової функції зі збільшенням кількості ітерацій..</i>	<i>14</i>
3.2.2	<i>Графіки залежності розв'язку від числа ітерацій.....</i>	<i>16</i>
	<b>ВИСНОВОК .....</b>	<b>17</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>18</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
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 ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,4$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 3$ , $\rho = 0,4$ , $L_{\min}$ 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 ( $\alpha = 3$ , $\beta = 2$ , $\rho = 0,3$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 4$ , $\rho$

	$= 0,6$ , $L_{\min}$ 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 ( $\alpha = 4$ , $\beta = 2$ , $\rho = 0,3$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,7$ , $L_{\min}$ 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 ( $\alpha = 3$ , $\beta = 2$ , $\rho = 0,7$ , $L_{\min}$ 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 ( $\alpha = 3$ , $\beta = 2$ , $\rho = 0,6$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,4$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 3$ , $\rho = 0,4$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,4$ , $L_{\min}$ 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 ( $\alpha = 2$ , $\beta = 3$ , $\rho = 0,4$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 35$ , починають маршрут в різних випадкових вершинах).

## Варіант 2

## 3.1 Програмна реалізація алгоритму

## 3.1.1 Вихідний код

```

from audioop import reverse
from xml.etree.ElementTree import tostring
import numpy as np
import random
import matplotlib.pyplot as plt

class Graph:
    def __init__(self, edge_weights, default_pheromone_level = None) -> None:

        assert edge_weights.shape[0] == edge_weights.shape[1]

        self.vertex_cardinality = edge_weights.shape[0]
        self.edge_weights = edge_weights

        if default_pheromone_level:
            self.pheromone_levels = np.full_like(edge_weights,
default_pheromone_level).astype('float64')
        else:
            self.pheromone_levels = np.full_like(edge_weights,
self.edge_weights.mean()).astype('float64')

    def __str__(self):

        return f'Vertex Cardinality = {str(self.vertex_cardinality)}\nEdge Weights
Matrix:\n{self.edge_weights}\nPheromone Levels Matrix:\n{self.pheromone_levels}'

def cycle_length(g, cycle):

    length = 0
    i = 0

    while i < len(cycle) - 1:
        length += g.edge_weights[cycle[i]][cycle[i+1]]
        i += 1

    return length

def traverse_graph(g, initial_vertex = 0, alpha_value = 2.0, beta_value = 4.0):

    visited = np.asarray([False for _ in range(g.vertex_cardinality)])
    visited[initial_vertex] = True
    cycle = [initial_vertex]
    curr_vertex = initial_vertex
    path_length = 0

    for _ in range(g.vertex_cardinality - 1):

        jumps_neighbors = []
        jumps_values = []
        jumps = []
        for vertex in range(g.vertex_cardinality):
            if not visited[vertex]:
                pheromone_level = max(g.pheromone_levels[curr_vertex][vertex], 1e-5)
                v = (pheromone_level**alpha_value) /
(g.edge_weights[curr_vertex][vertex]**beta_value)

```

```

        jumps.append((vertex, v))
        jumps_neighbors.append(vertex)
        jumps_values.append(v)

    next_vertex = random.choices(jumps_neighbors, weights = jumps_values, k =
1) [0]

    visited[next_vertex] = True
    curr_vertex = next_vertex
    cycle.append(curr_vertex)

    cycle.append(initial_vertex)
    path_length = cycle_length(g, cycle)

    return cycle, path_length

def calculate_optimal_length(g: Graph) -> np.float64:

    min_length = np.inf

    for i in range(g.vertex_cardinality):
        initial_vertex = i
        curr_vertex = initial_vertex

        visited = np.asarray([False for _ in range(g.vertex_cardinality)])
        visited[i] = True
        curr_length = 0

        for _ in range(g.vertex_cardinality):
            curr_min_vertex = curr_vertex
            curr_min = np.inf

            for vertex in range(g.vertex_cardinality):
                if not visited[vertex] and curr_min >
g.edge_weights[curr_vertex][vertex]:
                    curr_min_vertex = vertex
                    curr_min = g.edge_weights[curr_vertex][curr_min_vertex]

            visited[curr_min_vertex] = True
            curr_length += g.edge_weights[curr_vertex][curr_min_vertex]
            curr_vertex = curr_min_vertex

        curr_length += g.edge_weights[curr_vertex][initial_vertex]

        # print(curr_length)

        if min_length > curr_length:
            min_length = curr_length

    return min_length

def ant_colony_optimization(g, alpha_value = 2.0, beta_value = 4.0, min_length =
None, iterations = 1000, ants_per_iteration = 30, degradation_factor = .4,
plot_the_graph = False):

    if not min_length:
        min_length = calculate_optimal_length(g)
        print("L_min = ", min_length)

    best_cycle = (None, np.inf)
    old_best = (None, np.inf)
    inertia = 0
    patience = 50

    x = []
    y = []

```

```

    for i in range(iterations):
        cycles = []
        cycles = [traverse_graph(g, random.randint(0, g.vertex_cardinality - 1),
alpha_value, beta_value)
                    for _ in range(ants_per_iteration)]
        curr_best_cycle = (None, np.inf)
        best_cycle_as_for_now = best_cycle

        if best_cycle[0]:
            cycles.append(best_cycle)
            old_best = best_cycle

        g.pheromone_levels *= (1 - degradation_factor)

        for cycle, path_length in cycles:

            if path_length < best_cycle[1]:
                best_cycle = (cycle, path_length)

            if curr_best_cycle[1] > path_length and cycle !=
best_cycle_as_for_now[0]:
                curr_best_cycle = (cycle, path_length)

            delta = min_length / path_length
            j = 0

            while j < len(cycle) - 1:

                g.pheromone_levels[cycle[j]][cycle[j+1]] += delta
                j += 1

        if i % 20 == 0:
            if plot_the_graph:
                x.append(i + 1)
                y.append(curr_best_cycle[1])
                # print(curr_best_cycle[1])

        if best_cycle[0]:

            if old_best == best_cycle:
                inertia += 1
            else:
                inertia = 0

            if inertia > patience:
                g.pheromone_levels += g.pheromone_levels.mean()
                inertia = 0

        if plot_the_graph:
            plt.xlabel("Iterations")
            plt.ylabel("Path Length")
            plt.plot(x,y)
            plt.savefig('iterations_vs_length.pdf', format = 'pdf')

    return best_cycle

def get_random_graph(n, rand_min = 5, rand_max = 50):

    assert n > 1

    matrix = np.random.randint(rand_min, rand_max + 1, (n, n)).astype('float64')
    for i in range(n):
        matrix[i][i] = 0
        j = i + 1
        while j < n:
            matrix[j][i] = matrix[i][j]
            j += 1

```

```

graph = Graph(matrix)

    return graph

from ant_colony_optimization import *

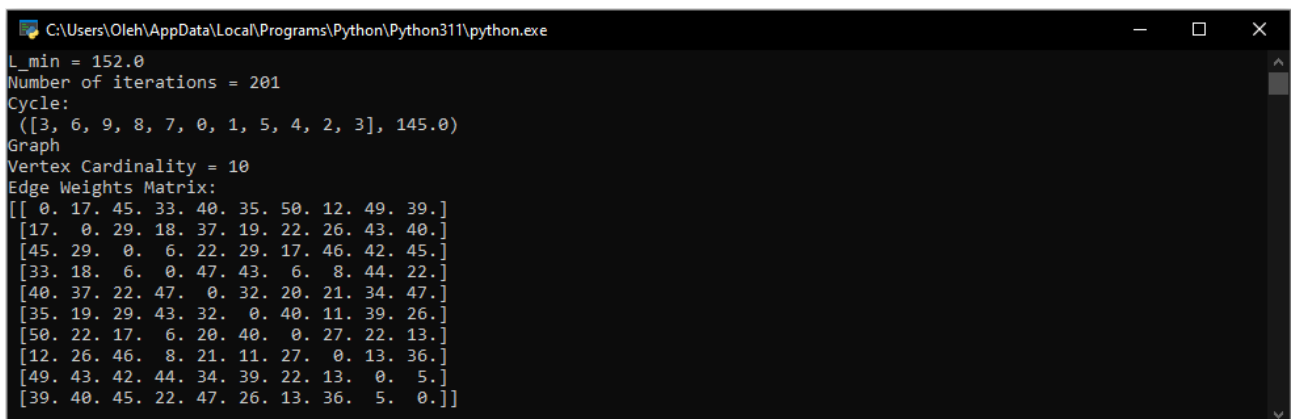
def main():
    test_graph = get_random_graph(100)
    it = 1001
    min_length = calculate_optimal_length(test_graph)
    print("L_min =", min_length)
    print("Number of iterations =", it)
    print("Cycle:\n", ant_colony_optimization(g = test_graph, iterations = it,
plot_the_graph = True, min_length = min_length))
    print("Graph")
    print(test_graph)

if __name__ == "__main__":
    main()

```

### 3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

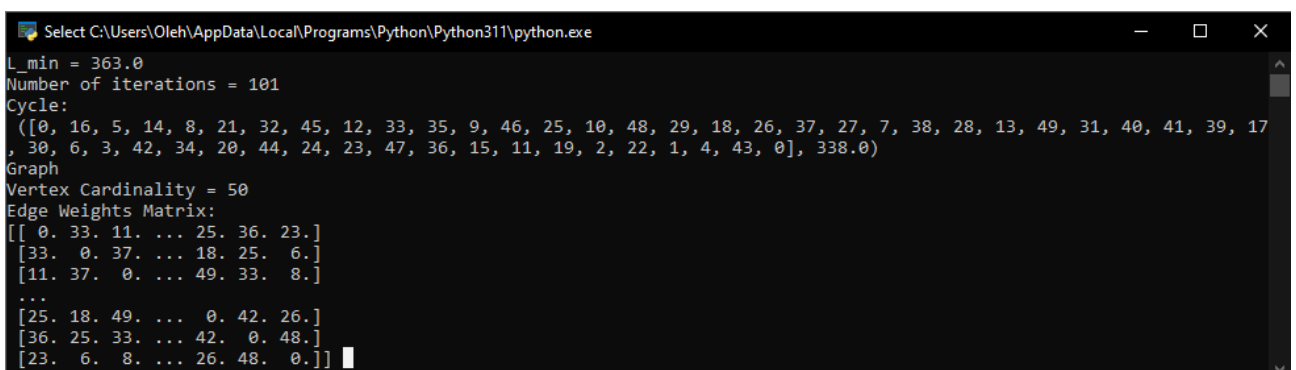


```

C:\Users\Oleh\AppData\Local\Programs\Python\Python311\python.exe
L_min = 152.0
Number of iterations = 201
Cycle:
([3, 6, 9, 8, 7, 0, 1, 5, 4, 2, 3], 145.0)
Graph
Vertex Cardinality = 10
Edge Weights Matrix:
[[ 0. 17. 45. 33. 40. 35. 50. 12. 49. 39.]
 [17.  0. 29. 18. 37. 19. 22. 26. 43. 40.]
 [45. 29.  0.  6. 22. 29. 17. 46. 42. 45.]
 [33. 18.  6.  0. 47. 43.  6.  8. 44. 22.]
 [40. 37. 22. 47.  0. 32. 20. 21. 34. 47.]
 [35. 19. 29. 43. 32.  0. 40. 11. 39. 26.]
 [50. 22. 17.  6. 20. 40.  0. 27. 22. 13.]
 [12. 26. 46.  8. 21. 11. 27.  0. 13. 36.]
 [49. 43. 42. 44. 34. 39. 22. 13.  0.  5.]
 [39. 40. 45. 22. 47. 26. 13. 36.  5.  0.]]

```

Рисунок 3.1 – Приклад роботи програми для графа з кількістю вершин 10 та кількістю ітерації 201



```

Select C:\Users\Oleh\AppData\Local\Programs\Python\Python311\python.exe
L_min = 363.0
Number of iterations = 101
Cycle:
([0, 16, 5, 14, 8, 21, 32, 45, 12, 33, 35, 9, 46, 25, 10, 48, 29, 18, 26, 37, 27, 7, 38, 28, 13, 49, 31, 40, 41, 39, 17, 30, 6, 3, 42, 34, 20, 44, 24, 23, 47, 36, 15, 11, 19, 2, 22, 1, 4, 43, 0], 338.0)
Graph
Vertex Cardinality = 50
Edge Weights Matrix:
[[ 0. 33. 11. ... 25. 36. 23.]
 [33.  0. 37. ... 18. 25.  6.]
 [11. 37.  0. ... 49. 33.  8.]
 ...
 [25. 18. 49. ...  0. 42. 26.]
 [36. 25. 33. ... 42.  0. 48.]
 [23.  6.  8. ... 26. 48.  0.]]

```

Рисунок 3.2 – Приклад роботи програми для графа з кількістю вершин 50 та кількістю ітерації 101

## 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.

Кількість ітерацій	Довжина поточного найкращого циклу
1	786.0
21	613.0
41	591.0
61	615.0
81	598.0
101	602.0
121	620.0
141	623.0
161	594.0
181	600.0
201	603.0
221	598.0
241	610.0
261	621.0
281	597.0
301	603.0
321	596.0
341	597.0
361	589.0
381	600.0
401	596.0
421	596.0
441	589.0
461	596.0

481	595.0
501	606.0
521	596.0
541	609.0
561	609.0
581	597.0
601	609.0
621	600.0
641	600.0
661	596.0
681	602.0
701	589.0
721	604.0
741	596.0
761	596.0
781	596.0
801	600.0
821	596.0
841	589.0
861	589.0
881	596.0
901	604.0
921	607.0
941	597.0
961	597.0
981	596.0
1001	597.0

### 3.2.2 Графік залежності розв'язку від числа ітерацій

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

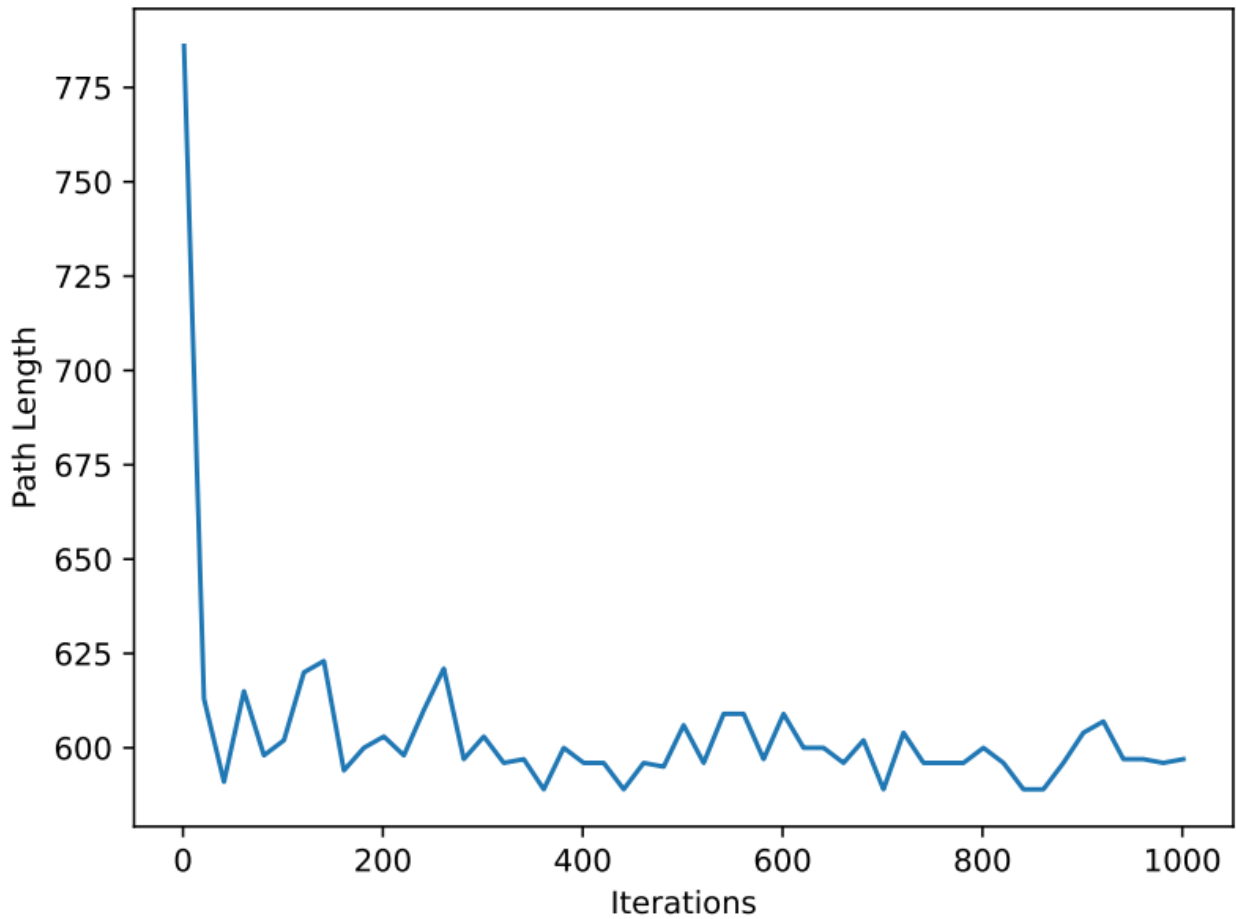


Рисунок 3.3 – Графік залежності довжини поточного найкращого циклу від числа ітерацій



## ВИСНОВОК

В рамках даної лабораторної роботи було опрацьовано та здійснено програмну реалізацію мурашиного алгоритму (з заданим набором параметрів) для розв'язку задачі комівояжера мовою python. Було виконано тестування алгоритму з параметром тривалості життя колонії 1001 (кількість ітерацій, яку здійснив алгоритм) і тридцятьма мурахами на кожній ітерації. Було відслідковано значення довжини поточного найкращого Гамільтонового циклу на кожній двадцятій ітерації з початком відліку у першій (початок відліку саме в першій ітерації зумовлений тим, що алгоритм здійснює найбільший «стрибок» якості розв'язку саме на перших ітераціях). Отримані дані було використано для побудови графіку залежності довжини поточного (на даній ітерації) найкращого Гамільтонового циклу від кількості ітерацій.

Тестування продемонструвало, що найбільший «стрибок» якості розв'язку алгоритм здійснює саме на перших ітераціях. Також варто зауважити, що отриманий розв'язок за достатньої кількості ітерацій перевершує або дорівнює знаходженню довжини Гамільтонового циклу за допомогою жадібного 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%.