ЛАБОРАТОРНА РОБОТА № 7
ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ

***Мета*:** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися дослідити метод мурашиних колоній.

Хід роботи:

Лістинг програми:

```
import random as rn
import numpy as np
from numpy.random import choice as np_choice

class AntAlgorithm(object):
    def __init__(self, distances, start, n_ants, n_best, n_iterations, decay,
alpha=1, beta=1):
        # Ініціалізація класу AntAlgorithm з необхідними параметрами
        self.distances = distances
        self.start = start
        self.pheromone = np.ones(self.distances.shape) / len(distances)
        self.all_inds = range(len(distances))
        self.n_ants = n_ants
        self.n_best = n_best
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta
        self.current_position = start
        self.next_position = None
        self.results = []

    def get_started(self):
            shortest_path = None # Найкоротший маршрут на поточній ітерації
            all_time_shortest_path = ("placeholder", np.inf) # Найкоротший
маршрут за всі ітерації
            for i in range(1, self.n_iterations + 1):
                all_paths = self.get_rout() # Отримання всіх маршрутів мурашок
                self.spread_pheronome(all_paths, self.n_best, shortest_path)
 # Розповсюдження феромонів
                shortest_path = min(all_paths, key=lambda x: x[1])    #
Знаходження найкоротшого маршруту на поточній ітерації
                if shortest_path[1] < all_time_shortest_path[1]:
```

					ДУ «Житомирська політехніка».23.122.4.000 – Лр7		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Дяченко В.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		.					
Керівник						ФІКТ Гр. КН-20-1(1)	
Н. контр.							
Зав. каф.							

```
                    all_time_shortest_path = shortest_path  # Оновлення
найкоротшого маршруту за всі ітерації
                self.pheromone = self.pheromone * self.decay  # Зменшення
феромонів

                 # Виведення результату кожної 50-ї ітерації та останньої
ітерації
                if i % 50 == 0 or i == self.n_iterations:
                    print("Step {}: {}".format(i, shortest_path[0]))
                    print("Total Distance: {}
km".format(all_time_shortest_path[1]))
                    self.results.append((i, shortest_path[0],
all_time_shortest_path[1]))

            return all_time_shortest_path, self.results if self.n_iterations >
0 else (None, [])



    def spread_pheronome(self, all_paths, n_best, shortest_path):

    # Розповсюдження феромонів на шляхах з найкращими маршрутами.

    # :param all_paths: Список усіх маршрутів
    # :param n_best: Кількість кращих маршрутів
    # :param shortest_path: Найкоротший маршрут на попередній ітерації

        sorted_paths = sorted(all_paths, key=lambda x: x[1])
        for path, dist in sorted_paths[:n_best]:
            for move in path:
                self.pheromone[move] += 1.0 / self.distances[move]

    def get_total_distance(self, path):

    #  Отримання загальної відстані для даного маршруту.
    # :param path: Маршрут
    # :return: Загальна відстань маршруту

        total_dist = 0
        for ele in path:
            total_dist += self.distances[ele]
        return total_dist

    def get_rout(self):
    # Отримання маршрутів для всіх мурашок.
    # :return: Список маршрутів для всіх мурашок

        all_paths = []
        for i in range(self.n_ants):
```

```
            path = self.get_path(self.current_position)
            all_paths.append((path, self.get_total_distance(path)))
        return all_paths

    def get_path(self, start):

   # Отримання маршруту для одного мурашки.

   # :param start: Початкове місто для мурашки
   # :return: Маршрут для мурашки

        path = []
        tabu = set()
        tabu.add(start)
        previous_step = start
        for i in range(len(self.distances) - 1):
            move = self.get_next_position(self.pheromone[previous_step],
self.distances[previous_step], tabu)
            path.append((previous_step, move))
            previous_step = move
            tabu.add(move)
        self.next_position = start
        path.append((previous_step, start))
        self.current_position = self.next_position
        return path

    def get_next_position(self, pheromone, dist, tabu):

  #  Отримання наступної позиції для мурашки.

  #  :param pheromone: Матриця феромонів для поточної позиції
  #  :param dist: Матриця відстаней для поточної позиції
  #  :param tabu: Множина заборонених позицій
  #  :return: Наступна позиція для мурашки

        pheromone = np.copy(pheromone)
        pheromone[list(tabu)] = 0

        row = pheromone ** self.alpha * ((1.0 / dist) ** self.beta)

        norm_row = row / row.sum()
        move = np_choice(self.all_inds, 1, p=norm_row)[0]
        return move


n_ants = len(table_cities)
n_best = 3
n_iterations = 250
decay = 0.95
start = 3 # variant 4
```

```
ant_algorithm = AntAlgorithm(table_distances, start, n_ants, n_best,
n_iterations, decay, 1, 1)
final_path, iteration_results = ant_algorithm.get_started()

import matplotlib.pyplot as plt

# Витягуємо індекси з найкоротшого маршруту та збільшуємо кожне значення на 1
path_indices = [i[0] + 1 for i in final_path[0]]
num_cities = 25
# Вказуємо позначки на осі x як числа від 1 до 25
x_ticks = range(1, num_cities + 1)

# Отримання результатів для кожної 50-ї ітерації
for result in iteration_results:
    iteration_number, iteration_path, iteration_distance = result

    # Витягуємо індекси з маршруту ітерації та збільшуємо кожне значення на 1
    iteration_indices = [i[0] + 1 for i in iteration_path]

    # Побудова графіка для маршруту ітерації зі спеціальними позначками та
відзначеннями на осі y
    fig, ax = plt.subplots(figsize=(12, max(6, 0.3 * num_cities)))
    ax.plot(x_ticks, iteration_indices, marker='o', linestyle='-', col-
or='red', label=f'Iteration {iteration_number} ({iteration_distance:.2f} km)')
    ax.set_yticks(range(1, num_cities + 1))
    ax.set_yticklabels(table_cities, fontsize=10)   # Використовуємо назви
міст як мітки на осі y
    ax.set_xticks(x_ticks)
    plt.title(f'Маршрут пройдений мікровояжером (Ітерація {itera-
tion_number})', fontsize=16)
    plt.xlabel('Номера міст', fontsize=14)
    plt.ylabel('Назви міст', fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.legend()
    plt.show()
import numpy as np
table_distances = np.array( [
[  np.inf,  645,  868,  125,  748,  366,  256,  316,  1057, 382,  360,  471,
 428,  593,  311,  844,  602,  232,  575,  734,  521,  120,  343,  312,  396],
#1
[645,  np.inf,  252,  664,  81, 901,  533,  294,  394,  805,  975,  343,  468,
 196,  957,  446,  430,  877,  1130, 213,  376,  765,  324,  891,  672], #2
[868, 252,   np.inf,  858,  217,  1171, 727,  520,  148,  1111, 1221, 611,
 731,  390,  1045, 591,  706,  1100, 1391, 335,  560,  988,  547,  1141, 867],
#3
[125, 664,  858,   np.inf,  738,  431,  131,  407,  1182, 257,  423,  677,
 557,  468,  187,  803,  477,  298,  671,  690,  624,  185,  321,  389,
 271],    …
```

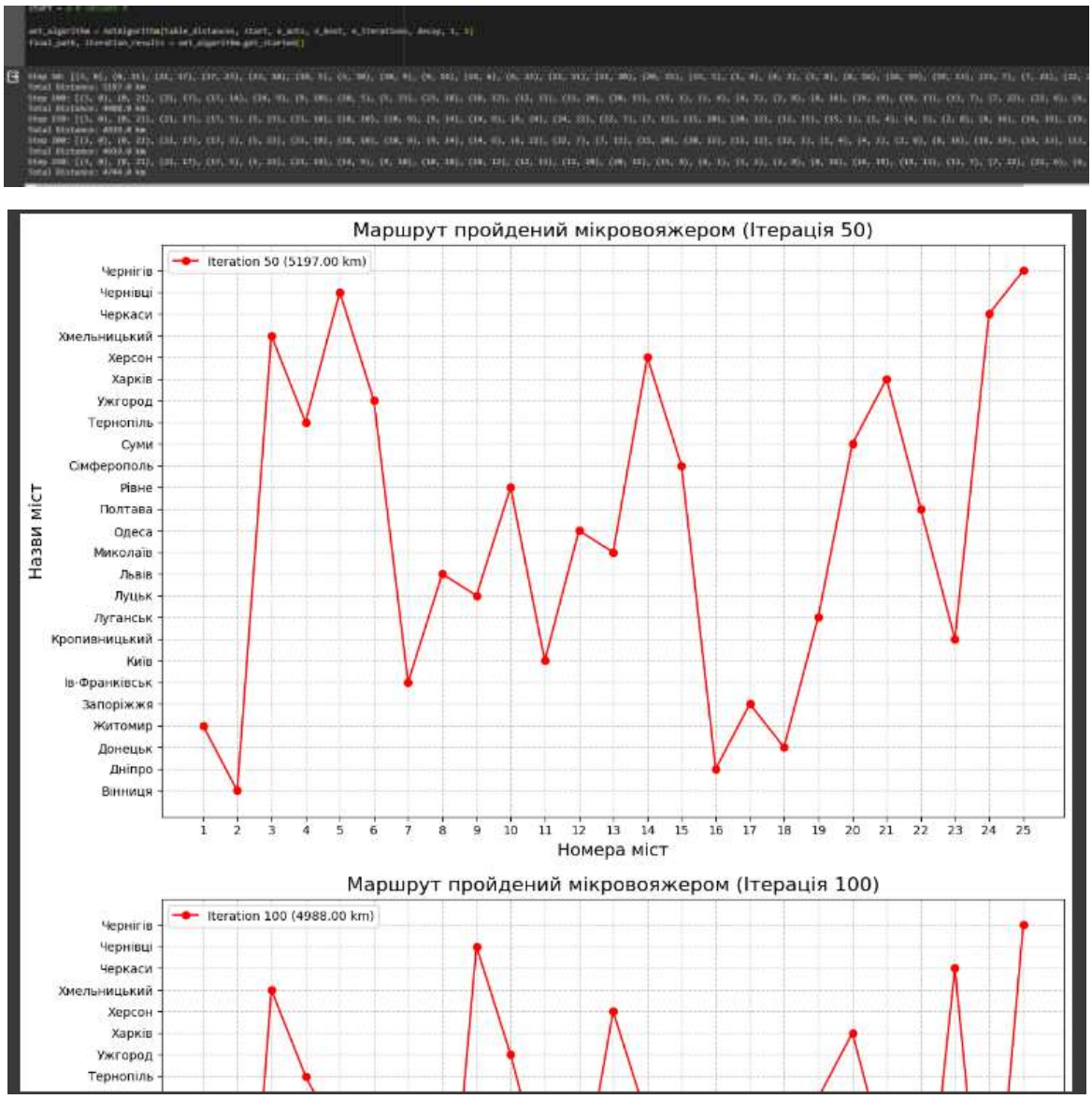

Рис.1. Результат роботи програми.

Рис.2. Задача 1.

Рис.3. Задача 2.

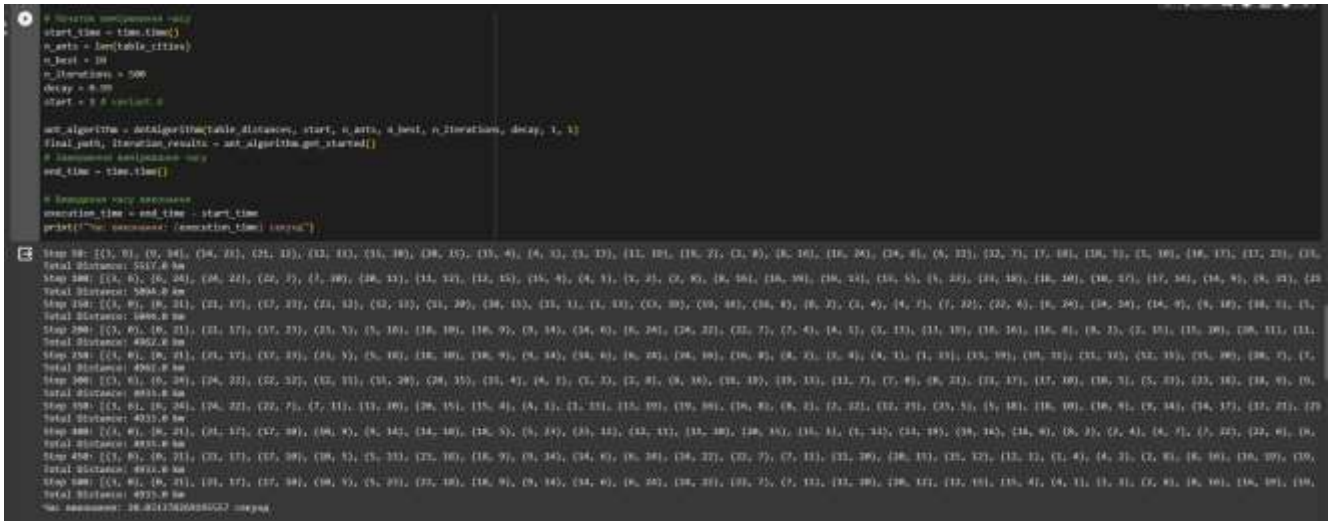

Рис.4. Задача 3.

Табл.1. Порівняльний аналіз.

№ Задачі	ants	best	iterations	decay	Довжина маршруту	Час виконання
1	5	1	250	0.95	4933.0 km	1.8 с
2	25	1	50	0.95	6276.0 km	2.5 с
3	25	10	500	0.99	4799.0 km	20.4 с

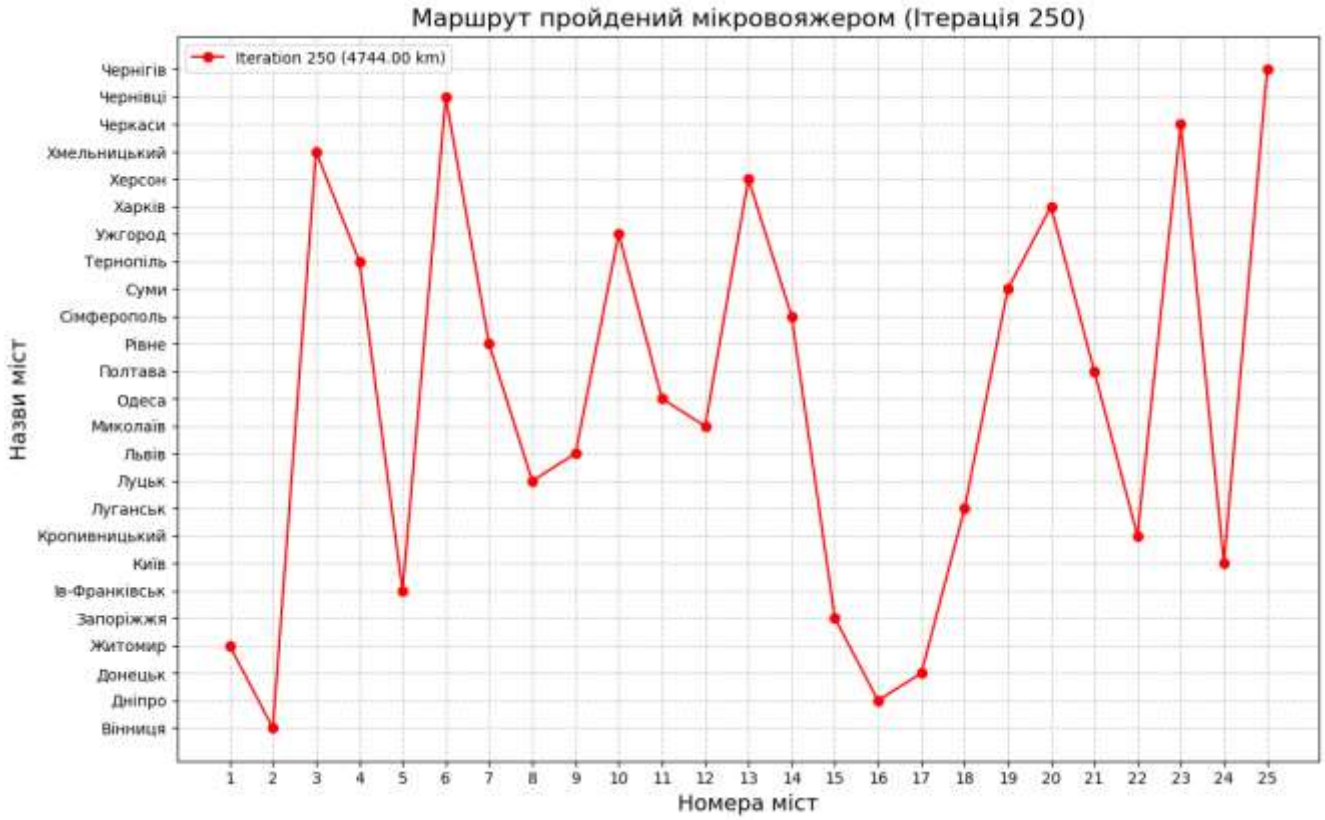

Рис.5. Графік найкоротшого маршруту з початком у місті згідно варіанту (4).

***Висновки*:** виконавши дану лабораторну роботу, ми дослідили метод мурашиних колоній, як узагальнений результат тестування за різних умов привели таблицю 1 та рисунок 5.

Для тестування створеної програми ми виконали задачу в різних умовах з різними початково заданими даними (для того щоб було простіше зробити висновки про її роботу). Виміряли час виконання та порівняли фінальний результат.

- Задача 1: Мала кількість мурашок і середня к-ть ітерацій для оцінки базового виконання.
- Задача 2: Збільшена кількість мурашок для вивчення їх впливу на результат.
- Задача 3: Використання багатьох кращих маршрутів та великої кількості ітерацій для оцінки впливу параметрів.

Збільшення кількості ітерацій може підвищити точність результатів, як видно за результатами задачі 1 та 2. Щоб зменшити використання ресурсів та час – рекомендується збільшувати к-ть ітерацій, а не мурах відповідно. Використання більшої кількості кращих маршрутів (best) може сприяти знаходженню кращих шляхів. Параметр decay впливає на швидкість втрати феромонів, і вищі значення можуть сприяти розгортанню феромонів на кращих маршрутах.

Тож враховуючи час виконання та довжину маршруту, задача 3 має найкращі результати хоча і вимагає більше часу для обчислень (20 с – не найгірший результат).

Найкоротший отриманий шлях з міста під індексом 4 (Житомир) за методом мурашиних колоній складає 4744 км.