ЛАБОРАТОРНА РОБОТА № 7

***Тема:* «ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ**»

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

**Хід роботи**

Посилання на програмнй код на Github:

<https://github.com/dengaevsky/Labs_AI/tree/main/lab7>

***Завдання 1.* Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера**

Використовуючи мову Python, розробити програму, що реалізує метод мурашиних колоній, для рішення задачі комівояжера, який їздить по містах України.

Відстані між обласними центрами. Місто за варіантом — Запоріжжя:

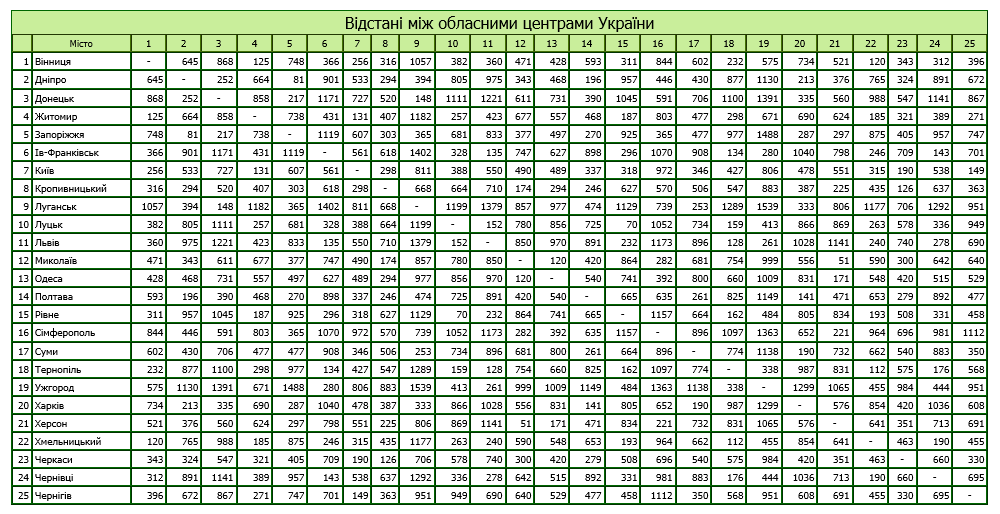


Рис. 1. Відстані між обласними центрами. Місто за варіантом — Запоріжжя

Лістинг класу AntColony у файлі ant\_colony.py:

**import** numpy **as** np  
**from** numpy.random **import** choice **as** np\_choice  
  
  
**class** AntColony(object):  
 **def** \_\_init\_\_(self, distances, n\_ants, n\_best, n\_iterations, decay, alpha=1.0, beta=1.0):  
 # Квадратна матриця відстаней. Діагональ вважається np.inf.  
 self.distances = distances  
  
 # Кількість мурах, що запускаються за ітерацію  
 self.n\_ants = n\_ants  
  
 # Кількість кращих мурах, які відкладають феромон  
 self.n\_best = n\_best  
  
 # Кількість ітерацій  
 self.n\_iterations = n\_iterations  
  
 # Швидкість розпаду феромону  
 self.decay = decay  
  
 # Eкспонента на феромоні, вища альфа надає феромону більшої ваги  
 self.alpha = alpha  
  
 # Eкспонента на дистанції, вища бета надає дистанції більшої ваги.  
 self.beta = beta  
  
 self.pheromone = np.ones(self.distances.shape) / len(distances)  
 self.all\_indexes = range(len(distances))  
  
 # Запуск алгоритму  
 **def** run(self, start=0):  
 shortest\_path = **None** all\_time\_shortest\_path = ("placeholder", np.inf)  
  
 **for** i **in** range(self.n\_iterations):  
 all\_paths = self.calc\_all\_paths(start)  
 self.spread\_pheronome(all\_paths, self.n\_best, shortest\_path=shortest\_path)  
 shortest\_path = min(all\_paths, key=**lambda** x: x[1])  
  
 **if** shortest\_path[1] < all\_time\_shortest\_path[1]:  
 all\_time\_shortest\_path = shortest\_path  
  
 self.pheromone = self.pheromone \* self.decay  
  
 **return** all\_time\_shortest\_path  
  
 # Ініціалізація значеннями феромонів  
 **def** spread\_pheronome(self, all\_paths, n\_best, shortest\_path):  
 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]  
  
 # Обрахунок довжини шляху  
 **def** calc\_path(self, path):  
 total\_dist = 0  
  
 **for** element **in** path:  
 total\_dist += self.distances[element]  
 **return** total\_dist  
  
 # Обрахунок довжини всіх шляхів  
 **def** calc\_all\_paths(self, start):  
 all\_paths = []  
  
 **for** i **in** range(self.n\_ants):  
 path = self.gen\_path(start)  
 all\_paths.append((path, self.calc\_path(path)))  
 **return** all\_paths  
  
 # Переміщення до наступного пункту переміщення  
 **def** gen\_path(self, start):  
 path = []  
  
 visited = set()  
 visited.add(start)  
  
 prev = start  
 **for** i **in** range(len(self.distances) - 1):  
 move = self.pick\_move(self.pheromone[prev], self.distances[prev], visited)  
 path.append((prev, move))  
 prev = move  
 visited.add(move)  
  
 path.append((prev, start))  
 **return** path  
  
 # Обрання наступного пункту переміщення  
 **def** pick\_move(self, pheromone, dist, visited):  
 pheromone = np.copy(pheromone)  
 pheromone[list(visited)] = 0  
  
 row = pheromone \*\* self.alpha \* ((1.0 / dist) \*\* self.beta)  
 normalized\_row = row / row.sum()  
  
 move = np\_choice(self.all\_indexes, 1, p=normalized\_row)[0]  
 **return** move

Лістинг програмного коду у файлі LR\_7\_task\_1.py:

**from** data **import** distances, start\_city, cities  
**from** ant\_colony **import** AntColony  
**import** matplotlib.pyplot **as** plt  
  
ant\_colony = AntColony(distances, 30, 10, 500, 0.8)  
  
result = ant\_colony.run(start=start\_city)  
print(f"Найкоротший шлях: {result[1]} км")  
  
# Знайдений шлях  
path = "Шлях: "  
**for** i **in** result[0]:  
 path += f"{cities[i[0]]} -> "  
print(path[:-4])  
  
# Графік для найкоротшого маршруту  
fig = plt.figure(figsize=(10, 10))  
  
plt.xticks([i + 1 **for** i **in** range(len(cities))])  
plt.yticks([i **for** i **in** range(len(cities))], cities)  
  
plt.xlabel("Номери міст")  
plt.ylabel("Назви міст")  
  
plt.title("Маршрут коміявожера")  
  
plt.plot([i + 1 **for** i **in** range(len(result[0]))], [i[0] **for** i **in** result[0]], ms=12, marker='o', mfc='b', mew=4,  
 color='#FF5733', linestyle='--')  
  
plt.grid()  
plt.show()

Результат виконання програми:

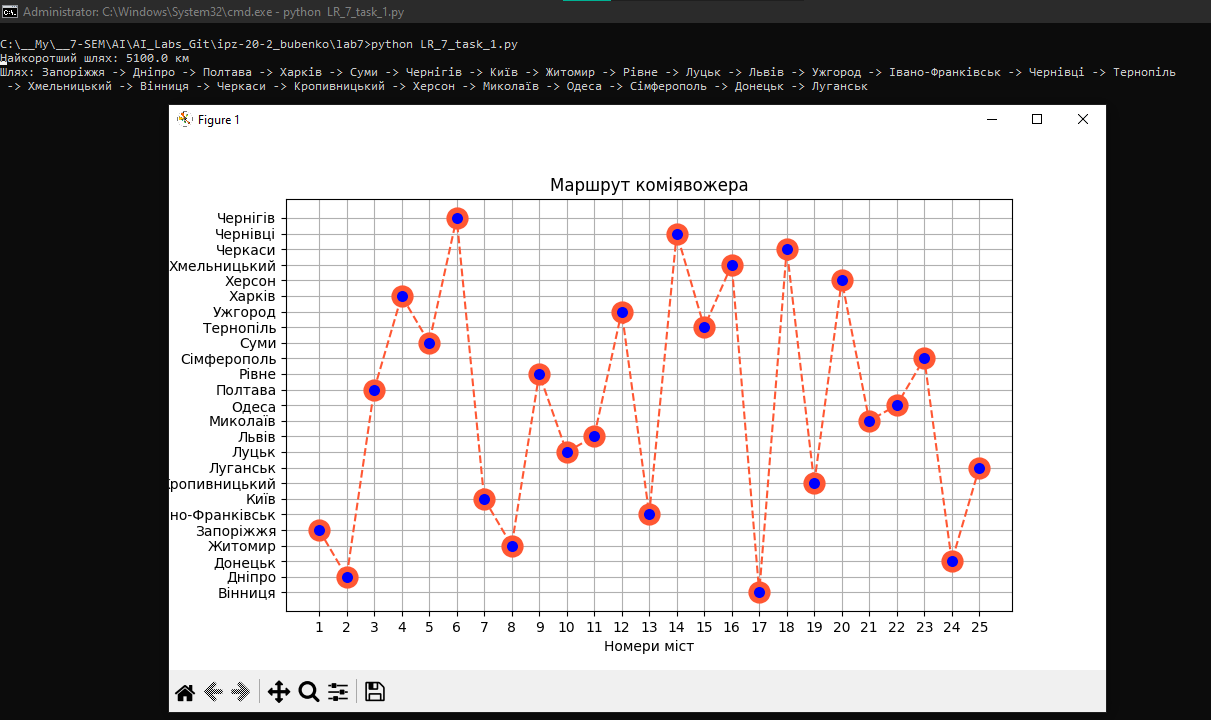


Рис. 2. Результат виконання програми (кількість ітерацій — 100, кількість кращих мурах — 5)

Довжина шляху: 5100.0 км

Шлях: Запоріжжя -> Дніпро -> Полтава -> Харків -> Суми -> Чернігів -> Київ -> Житомир -> Рівне -> Луцьк -> Львів -> Ужгород -> Івано-Франківськ -> Чернівці -> Тернопіль -> Хмельницький -> Вінниця -> Черкаси -> Кропивницький -> Херсон -> Миколаїв -> Одеса -> Сімферополь -> Донецьк -> Луганськ

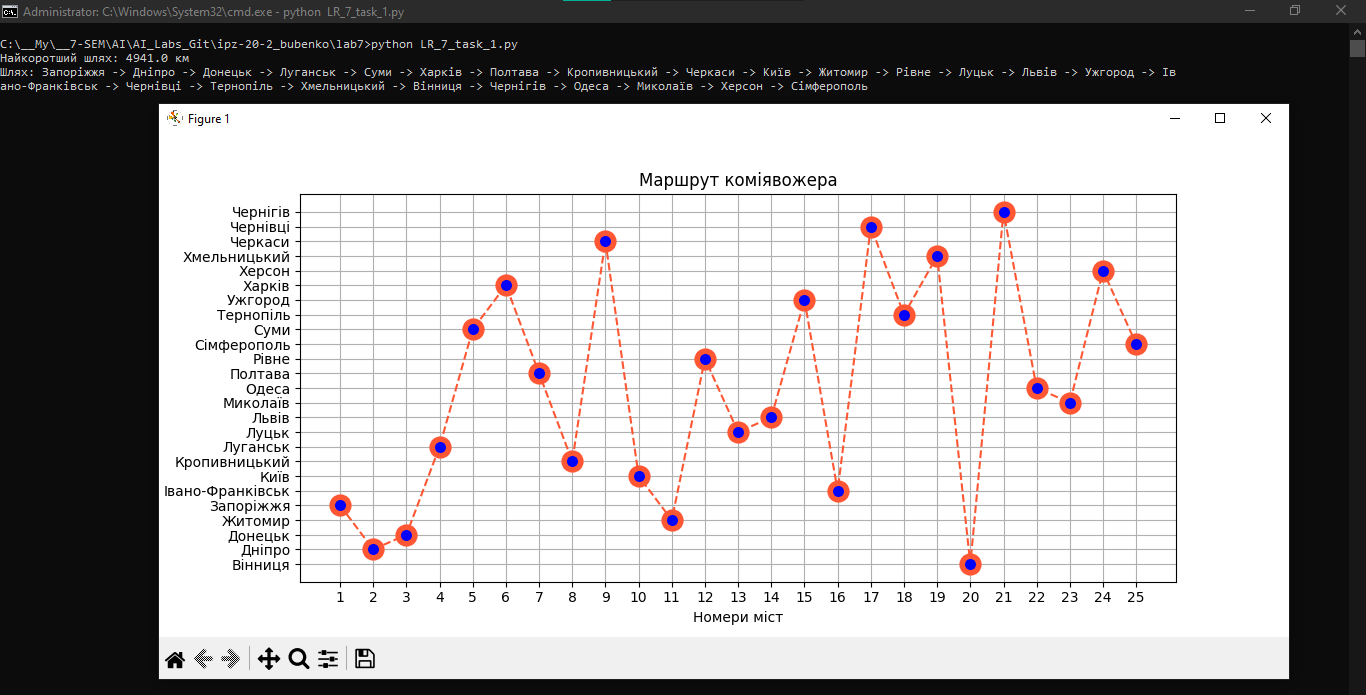


Рис. 3. Результат виконання програми (кількість ітерацій — 500, кількість кращих мурах — 10)

Довжина шляху: 4941.0 км

Шлях: Запоріжжя -> Дніпро -> Донецьк -> Луганськ -> Суми -> Харків -> Полтава -> Кропивницький -> Черкаси -> Київ -> Житомир -> Рівне -> Луцьк -> Львів -> Ужгород -> Івано-Франківськ -> Чернівці -> Тернопіль -> Хмельницький -> Вінниця -> Чернігів -> Одеса -> Миколаїв -> Херсон -> Сімферополь

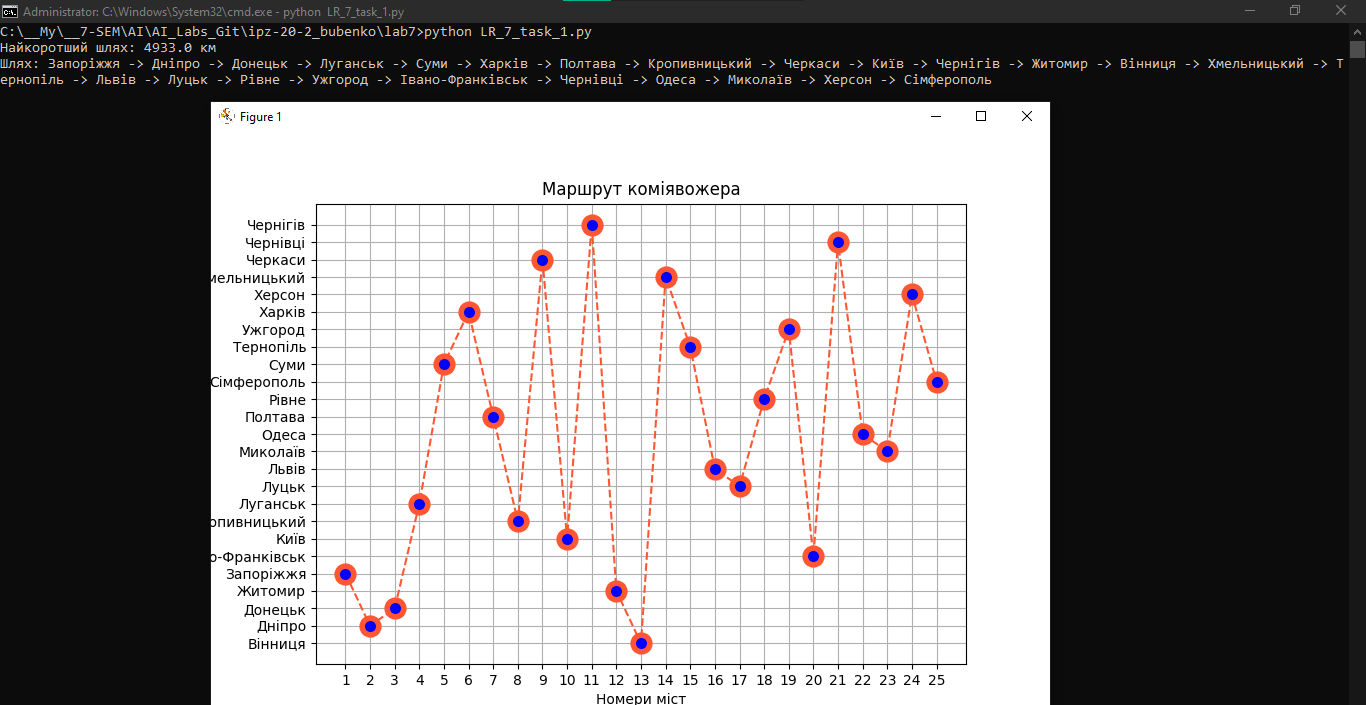


Рис. 4. Результат виконання програми (кількість ітерацій — 500, кількість кращих мурах — 20)

Довжина шляху: 4933.0 км

Шлях: Запоріжжя -> Дніпро -> Донецьк -> Луганськ -> Суми -> Харків -> Полтава -> Кропивницький -> Черкаси -> Київ -> Чернігів -> Житомир -> Вінниця -> Хмельницький -> Тернопіль -> Львів -> Луцьк -> Рівне -> Ужгород -> Івано-Франківськ -> Чернівці -> Одеса -> Миколаїв -> Херсон -> Сімферополь

***Висновки до завдання***

Отже, було реалізовано клас мурашиної колонії AntColony, для реалізації задачі комівояжера для заданої відправної точки, а за отриманими результатами побудовано графік. Було виконано порівняння роботи алгоритму при різних значеннях основних параметрів — кількості ітерацій та кількості кращих мурах. Виявлено, що при збільшенні значень цих параметрів збільшується ефективність алгоритму та алгоритм знаходить коротший шлях, однак суттєво збільшується тривалість роботи алгоритму. Це дає змогу зробити збалансований вибір параметрів в залежності від вимог конкретної задачі.

***Висновок:*** у ході виконання лабораторної роботи я використовуючи спеціалізовані бібліотеки та мову програмування Python навчився та дослідив метод мурашиних колоній.