**ЗВІТ**

**З лабораторної роботи №7**

**ДОСЛІДЖЕННЯ МУРАШИНИХ АЛГОРИТМІВ**

КН-20-1 навчальної групи

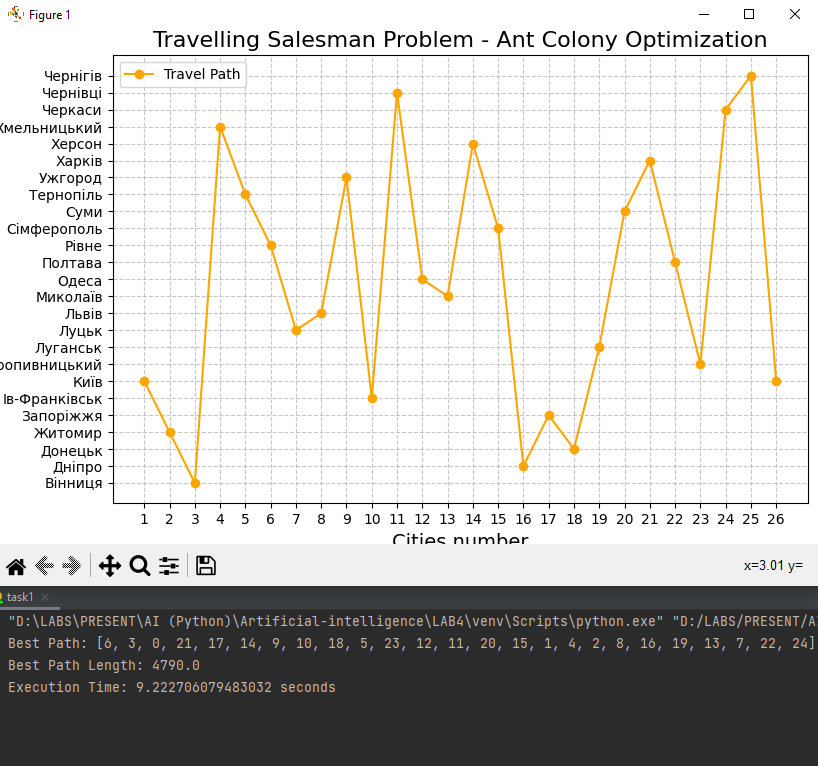
Кірія Даніли Олеговича варіант №6

***Мета:*** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися дослідити деякі типи нейронних мереж.

**Завдання 2.1.** Дослідження мурашиного алгоритму на прикладі рішення задачі комівояжера

Лістинг коду:

import numpy as np  
import matplotlib.pyplot as plt  
import time  
  
class TravellingAnt:  
 def \_\_init\_\_(self, start\_city, num\_cities):  
 self.current\_position = start\_city  
 self.next\_position = -1  
 self.tabu\_list = [self.current\_position]  
 self.visited\_count = 1  
 self.travel\_path = [self.current\_position]  
 self.path\_length = 0  
  
def initialize\_ants(num\_ants, num\_cities, start\_city):  
 ants = [TravellingAnt(start\_city, num\_cities) for \_ in range(num\_ants)]  
 return ants  
  
def update\_pheromones(pheromone\_matrix, ants, decay\_factor=0.5, pheromone\_constant=100, Q=0.5):  
 for ant in ants:  
 for i in range(len(ant.travel\_path) - 1):  
 pheromone\_matrix[ant.travel\_path[i], ant.travel\_path[i + 1]] += Q / ant.path\_length  
 pheromone\_matrix[ant.travel\_path[i + 1], ant.travel\_path[i]] += Q / ant.path\_length  
  
 pheromone\_matrix \*= decay\_factor  
  
def choose\_next\_destination(ant, distance\_matrix, pheromone\_matrix, alpha=1.1, beta=2.1):  
 current\_city = ant.current\_position  
 available\_cities = [i for i in range(len(distance\_matrix)) if i not in ant.tabu\_list]  
  
 probabilities = [((pheromone\_matrix[current\_city, city] \*\* alpha) \* (1 / distance\_matrix[current\_city, city] \*\* beta))  
 for city in available\_cities]  
  
 probabilities /= np.sum(probabilities)  
 next\_city = np.random.choice(available\_cities, p=probabilities)  
  
 return next\_city  
  
def update\_ant\_state(ant, next\_city, distance\_matrix):  
 ant.next\_position = next\_city  
 ant.tabu\_list.append(next\_city)  
 ant.visited\_count += 1  
 ant.path\_length += distance\_matrix[ant.current\_position, next\_city]  
 ant.current\_position = next\_city  
 ant.travel\_path.append(next\_city)  
  
def run\_ant\_colony\_optimization(num\_ants, num\_iterations, distance\_matrix, alpha=1.1, beta=2.2, decay\_factor=0.5, pheromone\_constant=100, Q=0.5, start\_city=None):  
 num\_cities = len(distance\_matrix)  
 pheromone\_matrix = np.ones((num\_cities, num\_cities))  
  
 best\_path = None  
 best\_path\_length = np.inf  
  
 start\_time = time.time()  
  
 for iteration in range(num\_iterations):  
 ants = initialize\_ants(num\_ants, num\_cities, start\_city)  
  
 for ant in ants:  
 for \_ in range(num\_cities - 1):  
 next\_city = choose\_next\_destination(ant, distance\_matrix, pheromone\_matrix, alpha, beta)  
 update\_ant\_state(ant, next\_city, distance\_matrix)  
  
 ant.path\_length += distance\_matrix[ant.travel\_path[-1], ant.travel\_path[0]]  
  
 if ant.path\_length < best\_path\_length:  
 best\_path\_length = ant.path\_length  
 best\_path = ant.travel\_path.copy()  
  
 update\_pheromones(pheromone\_matrix, ants, decay\_factor, pheromone\_constant, Q)  
  
 end\_time = time.time()  
 execution\_time = end\_time - start\_time  
  
 return best\_path, best\_path\_length, execution\_time  
  
def plot\_cities\_path(cities, path):  
 path\_indices = [i + 1 for i in path]  
 vertexes = range(1, len(cities) + 2)  
  
 # Add the starting city at the end to complete the tour  
 path\_indices.append(path\_indices[0])  
  
 fig, ax = plt.subplots(figsize=(12, 8))  
  
 ax.plot(vertexes, path\_indices, marker='o', linestyle='-', color='orange', label='Travel Path')  
  
 ax.set\_yticks(range(1, len(cities) + 1))  
 ax.set\_yticklabels(cities, fontsize=10)  
 ax.set\_xticks(vertexes)  
  
 plt.title('Travelling Salesman Problem - Ant Colony Optimization', fontsize=16)  
 plt.xlabel('Cities number', fontsize=14)  
 plt.ylabel('Cities name', fontsize=14)  
 plt.grid(True, linestyle='--', alpha=0.7)  
 plt.tight\_layout()  
 plt.legend()  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 cities\_list = ["Вінниця", "Дніпро", "Донецьк", "Житомир", "Запоріжжя", "Ів-Франківськ", "Київ",  
 "Кропивницький", "Луганськ", "Луцьк", "Львів", "Миколаїв", "Одеса", "Полтава",  
 "Рівне", "Сімферополь", "Суми", "Тернопіль", "Ужгород", "Харків", "Херсон",  
 "Хмельницький", "Черкаси", "Чернівці", "Чернігів"]  
  
 distance\_matrix = 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],  
 [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],  
 [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],  
 [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],  
 [748, 81, 217, 738, np.inf, 1119, 607, 303, 365, 681, 833, 377, 497, 270, 925, 365, 477, 977, 1488, 287, 297,  
 875,  
 405, 957, 747],  
 [366, 901, 1171, 431, 1119, np.inf, 561, 618, 1402, 328, 135, 747, 627, 898, 296, 1070, 908, 134, 280, 1040,  
 798,  
 246, 709, 143, 701],  
 [256, 533, 727, 131, 607, 561, np.inf, 298, 811, 388, 550, 490, 489, 337, 318, 972, 346, 427, 806, 478, 551,  
 315,  
 190, 538, 149],  
 [316, 294, 520, 407, 303, 618, 298, np.inf, 668, 664, 710, 174, 294, 246, 627, 570, 506, 547, 883, 387, 225,  
 435,  
 126, 637, 363],  
 [1057, 394, 148, 1182, 365, 1402, 811, 668, np.inf, 1199, 1379, 857, 977, 474, 1129, 739, 253, 1289, 1539, 333,  
 806,  
 1177, 706, 1292, 951],  
 [382, 805, 1111, 257, 681, 328, 388, 664, 1199, np.inf, 152, 780, 856, 725, 70, 1052, 734, 159, 413, 866, 869,  
 263,  
 578, 336, 949],  
 [360, 975, 1221, 423, 833, 135, 550, 710, 1379, 152, np.inf, 850, 970, 891, 232, 1173, 896, 128, 261, 1028,  
 1141,  
 240, 740, 278, 690],  
 [471, 343, 611, 677, 377, 747, 490, 174, 857, 780, 850, np.inf, 120, 420, 864, 282, 681, 754, 999, 556, 51, 590,  
 300, 642, 640],  
 [428, 468, 731, 557, 497, 627, 489, 294, 977, 856, 970, 120, np.inf, 540, 741, 392, 800, 660, 1009, 831, 171,  
 548,  
 420, 515, 529],  
 [593, 196, 390, 468, 270, 898, 337, 246, 474, 725, 891, 420, 540, np.inf, 665, 635, 261, 825, 1149, 141, 471,  
 653,  
 279, 892, 477],  
 [311, 957, 1045, 187, 925, 296, 318, 627, 1129, 70, 232, 864, 741, 665, np.inf, 1157, 664, 162, 484, 805, 834,  
 193,  
 508, 331, 458],  
 [844, 446, 591, 803, 365, 1070, 972, 570, 739, 1052, 1173, 282, 392, 635, 1157, np.inf, 896, 1097, 1363, 652,  
 221,  
 964, 696, 981, 1112],  
 [602, 430, 706, 477, 477, 908, 346, 506, 253, 734, 896, 681, 800, 261, 664, 896, np.inf, 774, 1138, 190, 732,  
 662,  
 540, 883, 350],  
 [232, 877, 1100, 298, 977, 134, 427, 547, 1289, 159, 128, 754, 660, 825, 162, 1097, 774, np.inf, 338, 987, 831,  
 112,  
 575, 176, 568],  
 [575, 1130, 1391, 671, 1488, 280, 806, 883, 1539, 413, 261, 999, 1009, 1149, 484, 1363, 1138, 338, np.inf, 1299,  
 1065, 455, 984, 444, 951],  
 [734, 213, 335, 690, 287, 1040, 478, 387, 333, 866, 1028, 556, 831, 141, 805, 652, 190, 987, 1299, np.inf, 576,  
 854,  
 420, 1036, 608],  
 [521, 376, 560, 624, 297, 798, 551, 225, 806, 869, 1141, 51, 171, 471, 834, 221, 732, 831, 1065, 576, np.inf,  
 641,  
 351, 713, 691],  
 [120, 765, 988, 185, 875, 246, 315, 435, 1177, 263, 240, 590, 548, 653, 193, 964, 662, 112, 455, 854, 641,  
 np.inf,  
 463, 190, 455],  
 [343, 324, 547, 321, 405, 709, 190, 126, 706, 578, 740, 300, 420, 279, 508, 696, 540, 575, 984, 420, 351, 463,  
 np.inf, 660, 330],  
 [312, 891, 1141, 389, 957, 143, 538, 637, 1292, 336, 278, 642, 515, 892, 331, 981, 883, 176, 444, 1036, 713,  
 190,  
 660, np.inf, 695],  
 [396, 672, 867, 271, 747, 701, 149, 363, 951, 949, 690, 640, 529, 477, 458, 1112, 350, 568, 951, 608, 691, 455,  
 330,  
 695, np.inf]  
 ])  
  
 start\_city\_index = 6  
 num\_ants = 20  
 num\_iterations = 300  
 alpha\_value = 1  
 beta\_value = 2  
 decay\_factor\_value = 0.8  
 pheromone\_constant\_value = 100  
 Q\_value = 1  
  
 best\_path, best\_path\_length, execution\_time = run\_ant\_colony\_optimization(  
 num\_ants=num\_ants, num\_iterations=num\_iterations, distance\_matrix=distance\_matrix,  
 start\_city=start\_city\_index, alpha=alpha\_value, beta=beta\_value, decay\_factor=decay\_factor\_value,  
 pheromone\_constant=pheromone\_constant\_value, Q=Q\_value  
 )  
  
 print("Best Path:", best\_path)  
 print("Best Path Length:", best\_path\_length)  
 print(f"Execution Time: {execution\_time} seconds")  
 plot\_cities\_path(cities\_list, best\_path)

****

Результат виконання програми.

|  |  |  |
| --- | --- | --- |
| Параметри | Довжина шляху | Час виконання (сек.) |
| num\_ants = 80 num\_iterations = 233 alpha\_value = 1.5 beta\_value = 2.2 decay\_factor\_value = 0.5 pheromone\_constant\_value = 100 Q\_value = 1 | 4979.0 | 29.5830 |
| num\_ants = 30 num\_iterations = 265 alpha\_value = 1 beta\_value = 4.2 decay\_factor\_value = 0.7 pheromone\_constant\_value = 100 Q\_value = 1 | 5099.0 | 2.9191 |
| num\_ants = 100 num\_iterations = 300 alpha\_value = 1.5 beta\_value = 4.9 decay\_factor\_value = 0.9 pheromone\_constant\_value = 100 Q\_value = 1 | 5041.0 | 54.2802 |

***Висновки:* Висновки:** використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися дослідив метод мурашиних колоній.

GitHub: <https://github.com/invicibleee/Artificial-intelligence.git>