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-з21 Клименко М. М.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.Н.* _____
(прізвище, ім'я, по батькові)

Київ 2024

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою. Зокрема мурашиного алгоритму, для розв’язування задачі комівояжера. Це включає в себе розробку алгоритму, програмну реалізацію та аналіз його ефективності шляхом тестування. Основна увага зосереджується на вивченні поведінки алгоритму в залежності від числа ітерацій та визначенні якості отриманих розв'язків.

2 ЗАВДАННЯ

Згідно варіанту 5, 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**
5	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

```
import random
import numpy as np

# Параметри
num_nodes = 150
num_ants = 35
alpha = 2
beta = 3
rho = 0.4
iterations = 1000
distance_range = (5, 50)

# Функція для генерації матриці відстаней
def generate_distance_matrix(num_nodes, distance_range):
    matrix = np.random.randint(distance_range[0], distance_range[1], size=(num_nodes, num_nodes))
    np.fill_diagonal(matrix, 0)
    return matrix

# Жадібний алгоритм для знаходження Lmin
def greedy_algorithm(distance_matrix, start_node=0):
    path = [start_node]
    total_distance = 0

    while len(path) < num_nodes:
        last_node = path[-1]
        next_node = np.argmin([distance_matrix[last_node][i] if i not in path else float('inf') for i in range(num_nodes)])
        path.append(next_node)
        total_distance += distance_matrix[last_node][next_node]

    total_distance += distance_matrix[path[-1]][path[0]]  # Повернення до початкової точки
    return total_distance
```

```
        # Функція для розрахунку ймовірності
        def calculate_probability(from_node, to_node, pheromone_matrix, distance_matrix):
            pheromone = pheromone_matrix[from_node][to_node] ** alpha
            inverse_distance = (1.0 / distance_matrix[from_node][to_node]) ** beta
            return pheromone * inverse_distance

        # Клас мурахи
        class Ant:
            def __init__(self):
                self.path = []
                self.total_distance = 0


            def find_path(self, distance_matrix, pheromone_matrix):
                    self.path = [random.randint(0, num_nodes - 1)]
                    self.total_distance = 0

                    for _ in range(num_nodes - 1):
                        current_node = self.path[-1]
                        probabilities = [calculate_probability(current_node, i, pheromone_matrix, distance_matrix) if i not in self.path else 0 for i in range(num_nodes)]

                        if sum(probabilities) > 0:
                            next_node = random.choices(range(num_nodes), probabilities, k=1)[0]
                        else:
                            # Всі ймовірності рівні нулю, обираємо випадковий невідвіданий вузол
                            unvisited_nodes = [i for i in range(num_nodes) if i not in self.path]
                            next_node = random.choice(unvisited_nodes)

                        self.path.append(next_node)
                        self.total_distance += distance_matrix[current_node][next_node]

                    # Повернення до початкової точки
                    self.total_distance += distance_matrix[self.path[-1]][self.path[0]]
```

```
        # Функція для оновлення феромону
        def update_pheromone(pheromone_matrix, ants, rho):
            pheromone_matrix *= (1 - rho)  # Випаровування феромону
            for ant in ants:
                for i in range(num_nodes - 1):
                    pheromone_matrix[ant.path[i]][ant.path[i+1]] += 1.0 / ant.total_distance

        # Ініціалізація
        distance_matrix = generate_distance_matrix(num_nodes, distance_range)
        pheromone_matrix = np.ones_like(distance_matrix, dtype=float)
        ants = [Ant() for _ in range(num_ants)]
        best_path = None
        best_distance = float('inf')

        # Жадібний алгоритм для знаходження Lmin
        lmin = greedy_algorithm(distance_matrix)

        # Основний цикл алгоритму
        for iteration in range(iterations):
            for ant in ants:
                ant.find_path(distance_matrix, pheromone_matrix)
                if ant.total_distance < best_distance:
                    best_distance = ant.total_distance
                    best_path = ant.path[:]

            update_pheromone(pheromone_matrix, ants, rho)

            if iteration % 20 == 0:
                print(f"Ітерація {iteration}: Краща відстань = {best_distance}")

        # Вивід найкращого знайденого шляху
        print(f"Найкращий шлях: {best_path}")
        print(f"Найкраща відстань: {best_distance}")
```

3.1.2 Приклади роботи

На рисунку 3.1 показаний приклад роботи програми

```
Ітерація 0: Краща відстань = 1287
Ітерація 20: Краща відстань = 813
Ітерація 40: Краща відстань = 786
Ітерація 60: Краща відстань = 786
Ітерація 80: Краща відстань = 786
Ітерація 100: Краща відстань = 786
Ітерація 120: Краща відстань = 786
Ітерація 140: Краща відстань = 786
Ітерація 160: Краща відстань = 786
Ітерація 180: Краща відстань = 786
Ітерація 200: Краща відстань = 786
Ітерація 220: Краща відстань = 786
Ітерація 240: Краща відстань = 786
Ітерація 260: Краща відстань = 786
Ітерація 280: Краща відстань = 786
Ітерація 300: Краща відстань = 786
Ітерація 320: Краща відстань = 786
Ітерація 340: Краща відстань = 786
Ітерація 360: Краща відстань = 786
Ітерація 380: Краща відстань = 786
Ітерація 400: Краща відстань = 786
Ітерація 420: Краща відстань = 786
Ітерація 440: Краща відстань = 786
Ітерація 460: Краща відстань = 786
Ітерація 480: Краща відстань = 786
Ітерація 500: Краща відстань = 786
Ітерація 520: Краща відстань = 786
Ітерація 540: Краща відстань = 786
Ітерація 560: Краща відстань = 786
Ітерація 580: Краща відстань = 786
Ітерація 600: Краща відстань = 786
Ітерація 620: Краща відстань = 786
Ітерація 640: Краща відстань = 786
Ітерація 660: Краща відстань = 786
Ітерація 680: Краща відстань = 786
Ітерація 700: Краща відстань = 786
Ітерація 720: Краща відстань = 786
Ітерація 740: Краща відстань = 786
Ітерація 760: Краща відстань = 786
Ітерація 780: Краща відстань = 786
Ітерація 800: Краща відстань = 786
Ітерація 820: Краща відстань = 786
Ітерація 840: Краща відстань = 786
Ітерація 860: Краща відстань = 786
Ітерація 880: Краща відстань = 786
Ітерація 900: Краща відстань = 786
Ітерація 920: Краща відстань = 786
Ітерація 940: Краща відстань = 786
Ітерація 960: Краща відстань = 786
Ітерація 980: Краща відстань = 786
Найкращий шлях: [61, 124, 35, 123, 89, 102, 112, 19, 119, 76, 9, 91, 11, 134, 74, 21, 2, 107, 20, 75, 71, 141, 18, 79, 93, 28, 104, 90,
121, 142, 139, 39, 23, 62, 110, 53, 0, 97, 4, 42, 99, 135, 78, 144, 3, 24, 95, 120, 13, 146, 55, 49, 129, 30, 5, 85, 136, 51, 31, 1, 113
, 73, 122, 148, 32, 94, 108, 56, 36, 116, 46, 130, 47, 101, 86, 105, 132, 103, 60, 38, 43, 54, 64, 66, 80, 127, 106, 40, 6, 58, 77, 29,
12, 118, 115, 63, 140, 50, 88, 44, 138, 45, 25, 87, 22, 41, 8, 109, 131, 37, 48, 117, 147, 98, 16, 70, 83, 111, 125, 96, 57, 34, 26, 128
, 27, 100, 65, 149, 67, 145, 137, 7, 92, 133, 52, 14, 126, 33, 81, 72, 17, 82, 10, 114, 84, 69, 59, 15, 143, 68]
Найкраща відстань: 786
```

3.1 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.

Табл. 3.1

Кількість ітерацій	0	20	40	60	80	100	120	140	160	180	200
Значення цільової функції	1287	813	786	786	786	786	786	786	786	786	786

Табл. 3.1

Кількість ітерацій	220	240	260	280	300	320	340	360	380	400	420
Значення цільової функції	786	786	786	786	786	786	786	786	786	786	786

Табл. 3.1

Кількість ітерацій	440	460	480	500	520	540	560	580	600	620	640
Значення цільової функції	786	786	786	786	786	786	786	786	786	786	786

Табл. 3.1

Кількість ітерацій	660	680	700	720	740	760	780	800	820	840	860
Значення цільової функції	786	786	786	786	786	786	786	786	786	786	786

Табл. 3.1

Кількість ітерацій	880	900	920	940	960	980	1000
Значення цільової функції	786	786	786	786	786	786	786

3.2.2 Графіки залежності розв'язку від числа ітерацій

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

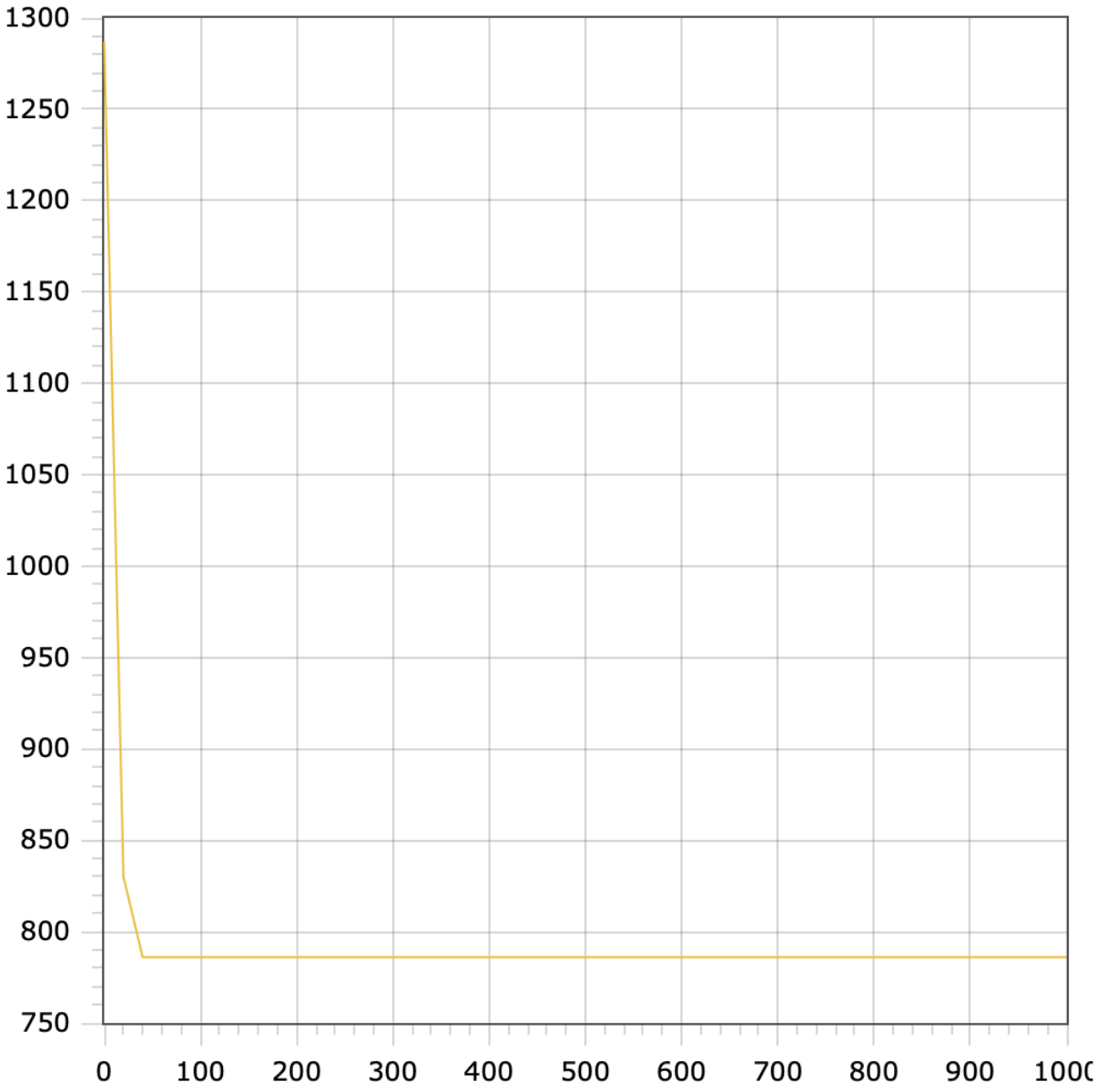

Рисунок 3.3 – Графік залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи було досягнуто встановленої мети через успішну реалізацію мурашиного алгоритму для задачі комівояжера. Програмна імплементація використовувала задані параметри алгоритму для оптимізації пошуку найкоротшого маршруту серед 150 вершин. Результати тестування підтвердили ефективність алгоритму, особливо відзначивши стабілізацію значення цільової функції на ранніх стадіях ітерацій. Графік залежності якості розв'язку від числа ітерацій ілюстрував цю стабілізацію, що свідчить про здатність алгоритму швидко знаходити ефективні рішення. Таким чином, лабораторна робота демонструє потенціал метаевристичних методів у розв'язуванні складних оптимізаційних задач та їх важливість для практичного застосування у різноманітних областях.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 10.12.2023 включно максимальний бал дорівнює – 5. Після 10.12.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація алгоритму – 55%;
- робота з гіт – 20%;
- тестування алгоритму– 20%;
- висновок – 5%.

+1 додатковий бал можна отримати за виконання роботи до 3.12.2023