Travelling Salesman Problem

6410110079 น.ส.เจณิตตา รัตนกันทา
6410110626 นายกฤตนัย ชูเกลี้ยง

เกี่ยวกับ DATASET

Dataset นี้เป็นการจำลองข้อมูลเกี่ยวกับการตำแหน่งพิกัดของเมืองต่างๆ โดยทุกๆ เมืองจะอยู่บนระนาบ 2 มิติที่ไม่มีความสูง/ลึก มีข้อมูลเปรียบเทียบเป็น 1 แถวต่อ 1 เมือง โดยชุดข้อมูลที่ใช้ในครั้งนี้มีทั้งหมด 100 ข้อมูล

นำเข้า LIBRARY ที่จำเป็น:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
```

- numpy (np) ใช้สำหรับการคำนวณทางคณิตศาสตร์ เช่น การสร้าง array และการคำนวณต่าง ๆ
- pandas (pd) ใช้ในการจัดการกับข้อมูลตาราง เช่น การโหลดข้อมูลจากไฟล์ CSV
- matplotlip (plt) ใช้ในการวาดกราฟแสดงระยะห่างระหว่างเมืองแต่ละเมือง
- datetime ใช้ในการจับเวลาที่ใช้ในการหาคำตอบของอัลกอริทึม

อ่านข้อมูลจากไฟล์:

```
# Load the dataset
data = pd.read_csv("medium.csv")
X = data["X"].values
Y = data["Y"].values
data["City"] = np.arange(1, len(data) + 1)
data.to_csv("medium.csv")
cities = data["City"].values
```

- โหลดข้อมูลจากไฟล์ medium.csv และเก็บค่าพิกัด X, Y ของเมืองแต่ละเมืองไว้ในอาร์เรย์ X และ Y
- เพิ่มคอลัมน์ "City" ให้กับ data ซึ่งเป็นหมายเลขเมือง (1, 2, 3, ...) แล้วบันทึกข้อมูลกลับไปที่ไฟล์ CSV เดิม

ฟังก์ชันคำนวณระยะทาง EUCLIDEAN ระหว่างสองเมือง:

```
# Function to calculate the Euclidean distance between two cities
def euclidean_distance(city1, city2):
    return np.sqrt((X[city2] - X[city1]) ** 2 + (Y[city2] - Y[city1]) ** 2)
```

- ฟังก์ชันนี้คำนวณระยะทางระหว่างเมืองสองเมือง (เมือง city1 และ city2) โดยใช้ระยะทาง Euclidean ซึ่งเป็นสูตรคำนวณระยะทางระหว่างสองจุดในระนาบ

ฟังก์ชันวัตถุประสงค์ สำหรับ TSP:

```python
# Objective function for TSP: Total distance of the path
def objective_function(path):
    total_distance = 0
    for i in range(1, len(path)):
        total_distance += euclidean_distance(path[i - 1], path[i])
    # Add the distance to return to the start
    total_distance += euclidean_distance(path[-1], path[0])
    return total_distance
```

- ฟังก์ชันนี้ใช้คำนวณระยะทางรวมของเส้นทาง path โดยคำนวณระยะทางระหว่างเมืองที่เดินทางต่อเนื่อง และเพิ่มระยะทางการกลับไปที่เมืองเริ่มต้นตอนสุดท้าย

ฟังก์ชันสร้างเส้นทางเริ่มต้นแบบสุ่ม:

```python
# Generate a random path (initial solution)
def random_solution():
    path = np.arange(len(X))
    np.random.shuffle(path)
    return path
```

- ฟังก์ชันนี้สร้างเส้นทางแบบสุ่ม โดยเรียงลำดับหมายเลขเมืองแบบสุ่มเพื่อใช้เป็นทางออกเริ่มต้น

ฟังก์ชันการกลายพันธุ์ (MUTATE) เส้นทาง:

```python
# Mutate the path (swap two cities)
def mutate_solution(solution):
    new_solution = solution.copy()
    i, j = np.random.choice(len(new_solution), 2, replace=False)
    new_solution[i], new_solution[j] = new_solution[j], new_solution[i]
    return new_solution
```

- ฟังก์ชันนี้ทำการสลับเมืองสองเมืองในเส้นทางแบบสุ่ม เพื่อเป็นการเปลี่ยนแปลงเส้นทางเดิม ซึ่งเป็นส่วนหนึ่งของการสำรวจแนวทางใหม่ ๆ

อัลกอริทึม ARTIFICIAL BEE COLONY (ABC)

```python
# Artificial Bee Colony (ABC) Algorithm for TSP
def ABC_algorithm():
    num_bees = 20  # Population size
    max_iterations = 100  # Maximum number of iterations
    limit = 50  # Limit for abandonment

    # Initialize population and fitness
    population = [random_solution() for _ in range(num_bees)]
    fitness = [objective_function(sol) for sol in population]
    trial_counter = [0] * num_bees  # Counter for scout bee phase
```

- num_bees: จำนวนผึ้ง (ประชากร)
- max_iterations: จำนวนครั้งของการทำซ้ำ
- limit: ขีดจำกัดสำหรับการละทิ้งทางออกที่ไม่มีการพัฒนา
- สร้างประชากรเริ่มต้นโดยสุ่มเส้นทาง และคำนวณค่า fitness (ระยะทางรวม) ของแต่ละเส้นทาง

อัลกอริทึม ARTIFICIAL BEE COLONY (ABC)

```python
# Initialize population and fitness
population = [random_solution() for _ in range(num_bees)]
fitness = [objective_function(sol) for sol in population]
trial_counter = [0] * num_bees  # Counter for scout bee phase

for iteration in range(max_iterations):
    # Employed bee phase
    for i in range(num_bees):
        mutant = mutate_solution(population[i])
        if objective_function(mutant) < fitness[i]:
            population[i] = mutant
            fitness[i] = objective_function(mutant)
            trial_counter[i] = 0  # Reset trial counter for improved solution
        else:
            trial_counter[i] += 1

    # Calculate selection probability based on fitness (for onlooker bees)
    fitness_sum = sum(fitness)
    probabilities = [f / fitness_sum for f in fitness]
```

ขั้นตอนของอัลกอริทึม ABC

1. Employed Bee Phase
    - ผึ้งสำรวจเปลี่ยนแปลงเส้นทางเดิมด้วยการกลายพันธุ์ และเก็บทางออกใหม่ถ้ามีค่า fitness ที่ดีกว่า

อัลกอริทึม ARTIFICIAL BEE COLONY (ABC)

```
    # Onlooker bee phase
    for _ in range(num_bees):
        selected = np.random.choice(num_bees, p=probabilities)
        mutant = mutate_solution(population[selected])
        if objective_function(mutant) < fitness[selected]:
            population[selected] = mutant
            fitness[selected] = objective_function(mutant)
            trial_counter[selected] = 0

    # Scout bee phase
    for i in range(num_bees):
        if trial_counter[i] > limit:  # Abandon solution if limit is exceeded
            population[i] = random_solution()
            fitness[i] = objective_function(population[i])
            trial_counter[i] = 0

    # Print progress
    best_solution = population[np.argmin(fitness)]
    print(f"Iteration {iteration + 1}, Best Fitness: {min(fitness)}")

# Return the best solution found
best_index = np.argmin(fitness)
return population[best_index], min(fitness)
```

2. Onlooker Bee Phase
   - ผึ้งผู้รอเลือกเส้นทางจากเส้นทางในปัจจุบันโดยใช้ความน่าจะเป็นที่ขึ้นกับ fitness และทำการสำรวจเส้นทางใหม่
3. Scout Bee Phase
   - ถ้าเส้นทางใดไม่ได้รับการปรับปรุงหลังจากทำการสำรวจหลายครั้ง ผึ้งนั้นจะถูกเปลี่ยนเส้นทางใหม่โดยสุ่ม
4. สรุปผล
   - ค้นหาเส้นทางที่มีค่า fitness ต่ำที่สุด (ระยะทางรวมสั้นที่สุด) และพิมพ์ผล

รันอัลกอริทึมและจับเวลา:

```python
# Run the ABC algorithm and set the timer
start_time = datetime.datetime.now()
best_solution, best_fitness = ABC_algorithm()
end_time = datetime.datetime.now()
execution_time = (end_time - start_time).total_seconds()

print(
    "Best Path (city indices):",
    best_solution,
    "\nBest Fitness (total distance):",
    best_fitness,
    "\nABC Execution Time:",
    execution_time,
    "seconds",
)
```

- วัดเวลาการทำงานของอัลกอริทึม และพิมพ์เส้นทางที่ดีที่สุด, ระยะทางรวมที่สั้นที่สุด และเวลาที่ใช้ในการรัน

แสดงกราฟเส้นทางที่ดีที่สุด:

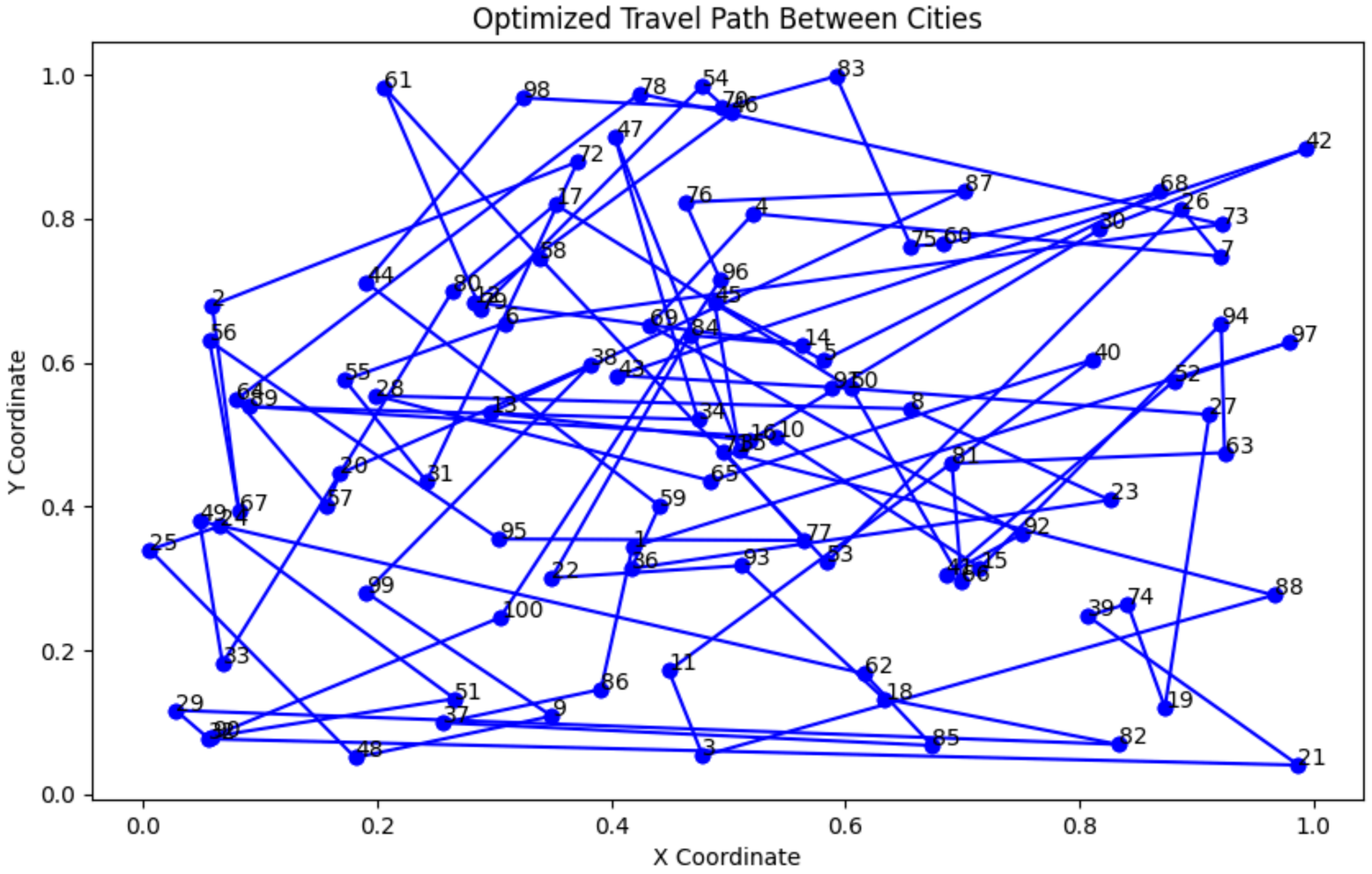
```python
# Plot the best path
plt.figure(figsize=(10, 6))
for i in range(len(best_solution)):
    start_city = best_solution[i]
    end_city = best_solution[(i + 1) % len(best_solution)]
    plt.plot([X[start_city], X[end_city]], [Y[start_city], Y[end_city]], "bo-")
    plt.text(X[start_city], Y[start_city], cities[start_city])

plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.title("Optimized Travel Path Between Cities")
plt.show()
```

- แสดงกราฟเส้นทางที่ดีที่สุดที่ได้จากอัลกอริทึม โดยใช้พิกัด X และ Y ของแต่ละเมือง

ผลการรัน:

```
Iteration 96, Best Fitness: 30.33399429488522
Iteration 97, Best Fitness: 30.33399429488522
Iteration 98, Best Fitness: 30.33399429488522
Iteration 99, Best Fitness: 30.33399429488522
Iteration 100, Best Fitness: 30.33399429488522
Best Path (city indices): [57 52 25  6  3 68 91 40 51 96  0 85 36 84 61 48 32 79 16
 49 65 80 62 93
 14  9 88 33 46 34 44  4 67 59 74 82 69 97 43 58 35 22  7 27 64 39 10  2
 87 70 76 94 55 66  1 71 30 54  5 72 77 63 56 19 37 98  8 47 24 23 50 89
 99 95 75 86 12 15 29 41 42 90 26 18 73 38 20 31 28 81 17 92 21 83 13 11
 45 53 78 60]
Best Fitness (total distance): 30.33399429488522
ABC Execution Time: 2.97895 seconds
```

ผลการรัน:


Optimized Travel Path Between Cities
Y Coordinate
X Coordinate
0.0
0.2
0.4
0.6
0.8
1.0

Thank you!