Computational Physics Homework 5

Cristopher Cerda Puga

Implementation of V.Cerny algotihm to the TSP

We will go directly to the algotihm. The paper by Cerny describes the next algorithm:

> We start by introducing some notations. We consider the traveling salesman problem for N stations. Let D be a $N \times N$ matrix with the elements $D(i, j)$ giving the distance from the ith station to the jth station. Let ${s_i}_1^N$, ${c_i}_1^N$,${t_i}_1^N$ denote permutations of integers $1, 2, \ldots, N$. Then, the problem is to find a permutation c for which the total length
>
> $$d = D(c_N, c_1) + \sum_{k=1}^{N-1} D(c_k, c_{k+1})$$
>
> is minimal. We propose the following algorithm.

- Step 0. Choose an arbitrary starting permutation ${s_i}_1^N$. Choose a real number (temperature) T.

```python
import random
import matplotlib.pyplot as plt
import sys
import Dmatrix


# Store the station coordinates for plotting
station_coords = [(0,0),(1,0),(1,1),(0,1),(10,0),(5,6),(4,9),(5,9),(7,3),(4,2)]

# Define the number of stations (N) and the distance matrix D
D = Dmatrix.generate_distance_matrix(station_coords)
N = len(station_coords)

initial_temperature = 1000
starting_permutation = list(range(1, N + 1))
random.shuffle(starting_permutation)
```

- Step 1. Set $c_k = s_k$ for $k = 1, 2, \ldots, B$. Calculate the corresponding length

$$d = D(c_n, c_1) + \sum_{k=1}^{N-1} D(c_k, c_{k+1})$$

```python
# Step 1: Initialize the current tour (c) with the starting permutation
current_tour = starting_permutation

# Calculate the length (d) of the current tour
total_length = sum(D[current_tour[i] - 1][current_tour[i + 1] - 1] for i in range(N - 1))
total_length += D[current_tour[N - 1] - 1][current_tour[0] - 1]
```

- Step 2. Set $i = 1$.
- Step 3. Generate randomly an integer j, $1 \leqslant j \leqslant N, j \neq i$

```python
i = 1

while i <= N:
    # Step 3: Generate a random integer j, 1 <= j <= N, j != i
    j = random.choice([x for x in range(1, N + 1) if x != i])
```

- Step 4. Construct a trial permutation from the current permutation as follows. Find

$$\tilde{i} = min(i, j), \qquad \tilde{j} = max(i, j)$$

Set

$$\begin{aligned} t_k &= c_k & k &= 1, 2, \ldots, i-1 \\ t_{\tilde{i}+k} &= c_{\tilde{j}-k} & k &= 0, 1, 2, \ldots, \tilde{j} - \tilde{i} \\ t_k &= c_k & k &= \tilde{j}+1, \tilde{j}+2, \ldots, N. \end{aligned}$$

```
    # Step 4: Construct a trial permutation (t) based on i and j
    i_prime = min(i, j)
    j_prime = max(i, j)

    trial_tour = [0] * N

    for k in range(i_prime - 1):
        trial_tour[k] = current_tour[k]

    for k in range(j_prime - i_prime + 1):
        trial_tour[i_prime + k - 1] = current_tour[j_prime - k - 1]

    for k in range(j_prime, N):
        trial_tour[k] = current_tour[k]
```

- Step 5. Calculate the length corresponding to the trial permutation

$$d' = D(t_n, t_1) + \sum_{k=1}^{N-1} D(t_k, t_{k+1})$$

```
    # Step 5: Calculate the length (d') of the trial permutation
    total_length_trial = D[trial_tour[N - 1] - 1][trial_tour[0] - 1]
    for k in range(N - 1):
        total_length_trial += D[trial_tour[k] - 1][trial_tour[k + 1] - 1]
```

- Step 6. If $d' < d$, go to Step 7; otherwise, generate a random number $x, 0 < x < 1$. Then $x < \exp\{(d - d')/T\}$, go to Step 7; othersie go to Step 8.
- Step 7. Set $c_k = t_k, k = 1, 2, \ldots, N$. Set $d = d'$.
- Step 8. Increase i by one. Then, if $i \leqslant N$, go to Step 3; otherwise, go to Step 2.

```
    # Step 6: Accept or reject the trial solution based on the change in length and temperature
    if total_length_trial < total_length:
        current_tour = trial_tour
        total_length = total_length_trial

    # Step 8: Increase i by one
    i += 1

    # Check if the current solution is the best found so far
    if total_length < best_length:
        best_tour = current_tour.copy()
        best_length = total_length

    # Reduce the temperature using the cooling rate
    current_temperature *= cooling_rate
```

Now, implementing a cooling rate to explore a bigger set of solutions (because this algorithm is not deterministic) and find the best of the solutions:

```python
import random
import numpy as np
import matplotlib.pyplot as plt
import sys
import Dmatrix


# Store the station coordinates for plotting

station_coords = [put here your cords]
N = len(station_coords)
D = Dmatrix.generate_distance_matrix(station_coords)
# Define the number of stations (N) and the distance matrix D

initial_temperature = 1000
cooling_rate = 0.999

# Initialize the best tour and length
best_tour = []
best_length = sys.maxsize

current_temperature = initial_temperature

# Create lists to store best tours and their lengths for each iteration
best_tours_history = []
best_lengths_history = []

# Step 0: choose an arbitrary starting permutation {s_i}_1^N
starting_permutation = list(range(1, N + 1))
random.shuffle(starting_permutation)

# Step 1: Initialize the current tour (c) with the starting permutation
current_tour = starting_permutation

while current_temperature > 1e-3:  # Adjust the threshold as needed

    # Calculate the length (d) of the current tour
    total_length = sum(D[current_tour[i] - 1][current_tour[i + 1] - 1] for i in range(N - 1))
    total_length += D[current_tour[N - 1] - 1][current_tour[0] - 1]

    i = 1

    while i <= N:
        # Step 3: Generate a random integer j, 1 <= j <= N, j != i
        j = random.choice([x for x in range(1, N + 1) if x != i])

        # Step 4: Construct a trial permutation (t) based on i and j
        i_prime = min(i, j)
        j_prime = max(i, j)

        trial_tour = [0] * N

        for k in range(i_prime - 1):
            trial_tour[k] = current_tour[k]

        for k in range(j_prime - i_prime + 1):
            trial_tour[i_prime + k - 1] = current_tour[j_prime - k - 1]
```

```
        for k in range(j_prime, N):
            trial_tour[k] = current_tour[k]


        # Step 5: Calculate the length (d') of the trial permutation
        total_length_trial = D[trial_tour[N - 1] - 1][trial_tour[0] - 1]
        for k in range(N - 1):
            total_length_trial += D[trial_tour[k] - 1][trial_tour[k + 1] - 1]

        # Step 6: Accept or reject the trial solution based on the change in length and temperature
        if total_length_trial < total_length:
            current_tour = trial_tour
            total_length = total_length_trial

        # Step 8: Increase i by one
        i += 1

    # Check if the current solution is the best found so far
    if total_length < best_length:
        best_tour = current_tour.copy()
        best_length = total_length

    # Append the current best tour and length to the history lists
    best_tours_history.append(best_tour)
    best_lengths_history.append(best_length)

    # Reduce the temperature using the cooling rate
    current_temperature *= cooling_rate

# Plot the last best tour
plt.figure(figsize=(10, 6))
plt.scatter([coord[0] for coord in station_coords], [coord[1] for coord in station_coords],
            c='blue', marker='o', label='Stations', s=100)

# Add labels for each point
for i, coord in enumerate(station_coords):
    plt.text(coord[0], coord[1], str(i + 1), fontsize=11, ha='center', va='center', color='white')

# Create lines to represent the last best tour
tour_x = [station_coords[i - 1][0] for i in best_tour]
tour_y = [station_coords[i - 1][1] for i in best_tour]
tour_x.append(station_coords[best_tour[0] - 1][0])
tour_y.append(station_coords[best_tour[0] - 1][1])

# Print the final best tour and length
print("Best Tour:", best_tour)
print("Best Length:", best_length)

# Plot the last best tour
plt.plot(tour_x, tour_y, c='red', linestyle='-', marker='o', label='Best Tour', markersize=8)
plt.title("Last Best TSP Solution")
plt.xlabel("X Coordinate")
plt.ylabel("Y Coordinate")
plt.legend()
plt.grid(True)
plt.show()
```

```python

# Create a line plot to visualize the convergence of lengths
plt.plot(best_lengths_history, marker='o', linestyle='-', color='b')
plt.title("Convergence of Lengths")
plt.xlabel("Iteration")
plt.ylabel("Length")
plt.grid(True)
plt.show()
```

Some tests to the algorithm.

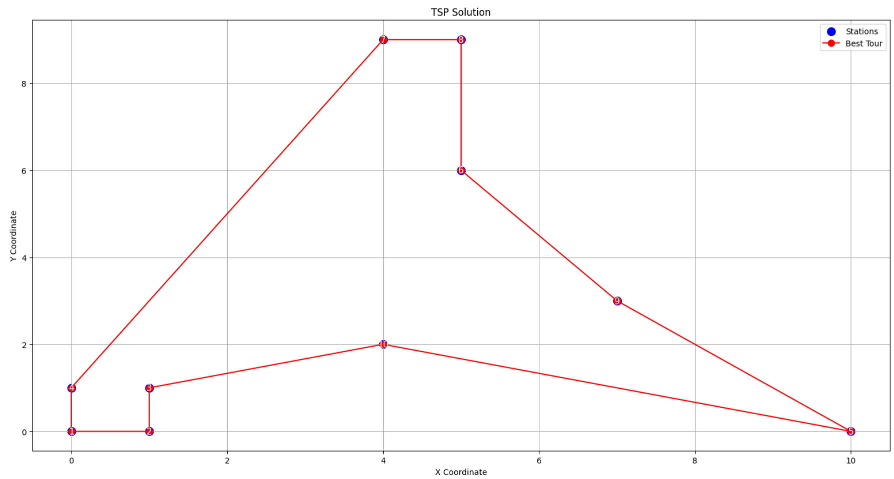

Figure 1: TSP solution for some random stations: (0,0),(1,0),(1,1),(0,1),(10,0),(5,6),(4,9),(5,9),(7,3),(4,2)

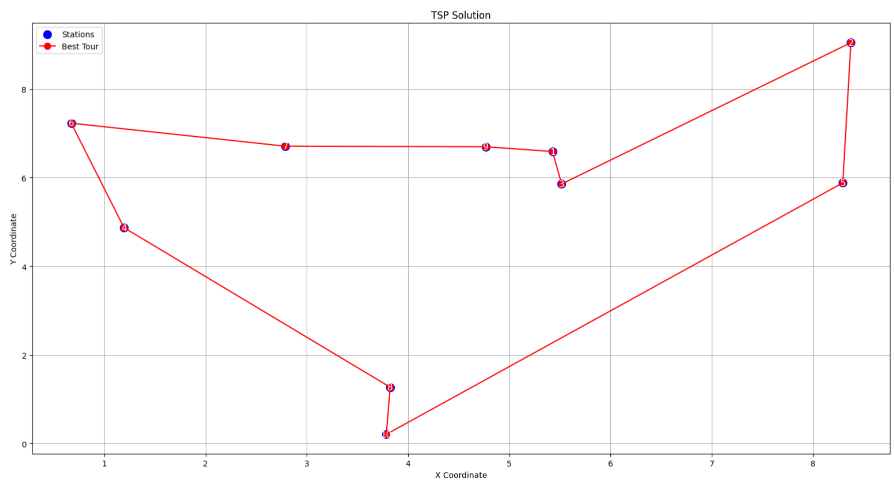

Figure 2: TSP solution for some random stations: (7.18595741997901, 7.2533416948888725), (9.859871610223017, 0.7662647426988956), (2.307605846919235 , 8.742757278977052), (8.520838782972039, 4.021150355784897), (2.4908764962726093, 5.758060625913367), (1.919270454080455, 0.369268569649589), (1.5991480695797233, 6.343000939793056), (7.3042243364926 94, 0.3873113060467248), (5.790183008710079, 9.279734749343852), (2.226796786397618, 0.5195162771424 4)

For a small quantity of stations, the algorithm behaves as expected, giving us the absolute minima of the trajectories. Now, for more points:

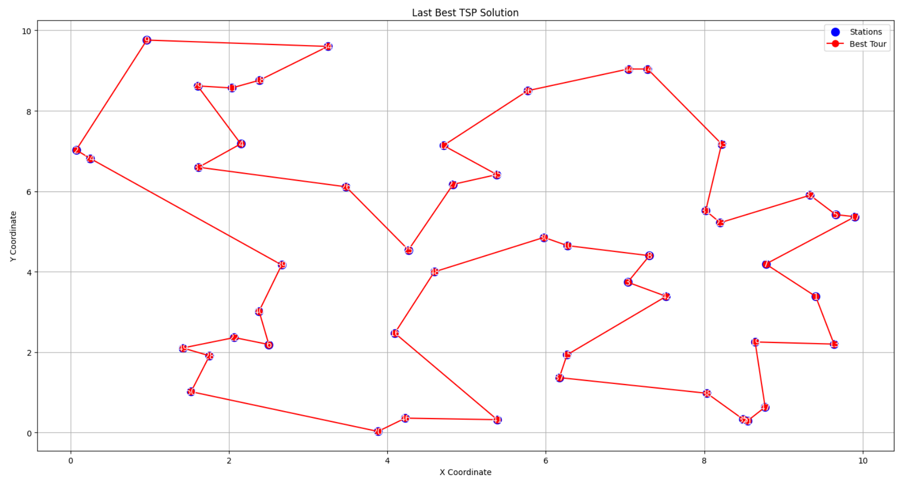

Figure 3: TSP solutions for some random stations.

We can even consider a more symmetric case. Let us work in the case of 119 points distributed along the circunference of a cricle of radius 10:

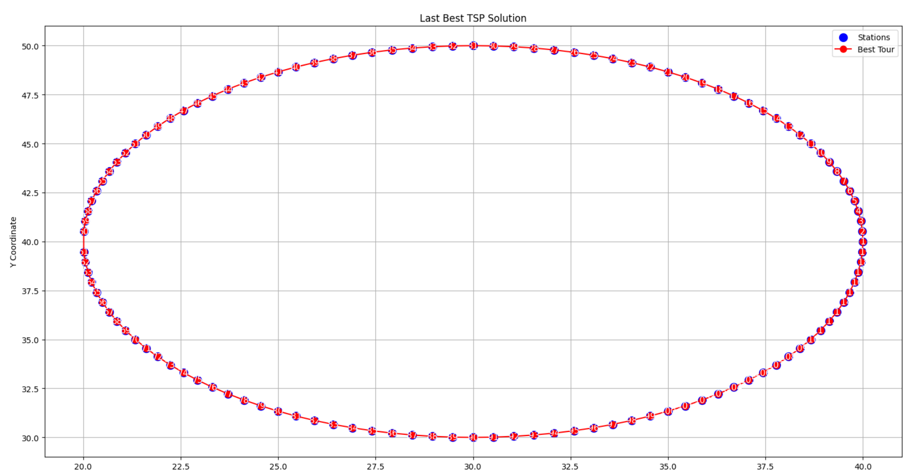

Figure 4: Symmetric TSP

Is clear that we get the optimal solution, $l = 62.82431713143947$, pretty close to the length of the circumference of the circle,