EE2703 - Assignment 7

Ata Karim Subhani <EE21B025>

March 29, 2023

1 Aim:

- To find the global minima of the given function.
- Apply **Simulated Annealing** to optimize the famous **Travelling Salesman Problem**

2 Instruction:

- Uploaded zip file contains two python script for both part of the assignment along with the gif of the animation for Travelling Salesman Problem.
- Since first part of the assignment was discussed in the class so I didn't write anything about that in the report file.
- This report mainly contains the explanation of second part of the assignment.

3 Explanation of part B

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import random
```

[9]:
```
# For 10 cities simulation use no_of_iterations = 1000 and change no_of_cities =␣
 ↪10
# For 100 cities simulation use no_of_iterations = 100000 and change␣
 ↪no_of_cities = 100

file = "10.txt"
no_of_iterations = 1000
no_of_cities = 100
no_of_frames =100
```

Accesing location of cities and storing them appropriately

[4]:
```
locations = []
with open(file, "r") as points:
    loc = points.readlines()
    for i,ele  in enumerate(loc[1:]):
```

```
        x=float(ele.split()[0])
        y=float(ele.split()[1])
        locations.append([x,y, i])
locations.append(locations[0])
```

`total_distance` function takes the location of cities in order and find the total distance traversed.

[5]:
```
def total_distance(locations):
    sum = 0
    for i in range(len(locations)-1):
        sum += np.
 ↪sqrt((locations[i][0]-locations[i+1][0])**2+(locations[i][1]-locations[i+1][1])**2)
    return sum
```

`cooling` function is used to decrease the temperature exponentially.

[6]:
```
def cooling(temp, alpha):
    return temp*alpha
```

`check_accept` function is used to decide whether to take or not take the path in particular iteration. When distance to be traversed is less than the minimum distance traveresed so far, then we surely accept that path. But if distance to be traversed is more than the minimum distance traveresed so far, then we accept that path with decreasing probability.

[7]:
```
def check_accept(temp, current_solution, new_solution):
    prob = min(1, np.exp(-(new_solution - current_solution) / temp))
    if(prob > random.uniform(0,1)):
        return True
    else:
        return False
```

`best_route` function return all the possible paths which we have iterated and also the least distance found in each iteration. I randomly chose two cities and reversed the order in which I traversed earlier. Now I decided to take this as a valid path or not on the basis of Simulated Annealing.

[12]:
```
def best_route(cordinate, no_of_iterations, Temp=1000, alpha=0.99):
    current_distance = total_distance(cordinate)
    best_distance = []
    best_distance.append(current_distance)
    Route = []
    Route.append(cordinate.copy())
    for _ in range(no_of_iterations):
        # Swap any two random location
        swap_list_indx = range( 1, len(cordinate)-1)
        i = random.randint(swap_list_indx[0], swap_list_indx[-1])
        j = random.randint(swap_list_indx[0], swap_list_indx[-1])
        if i == j:
            while i == j:
                j = random.randint(swap_list_indx[0], swap_list_indx[-1])
```

```
        new_order = cordinate.copy()
        new_order[min(i,j):max(j,i)+1] = reversed(new_order[min(i,j):max(j,i)+1])
        current_distance = total_distance(cordinate)
        new_distance = total_distance(new_order)
        if(new_distance<best_distance[-1]):
            best_distance.append(new_distance)
        else:
            best_distance.append(best_distance[-1])

        Temp = cooling(Temp, alpha)
        if(check_accept(Temp, current_distance, new_distance)):
            cordinate = new_order.copy()
        Route.append(cordinate.copy())
    return best_distance, Route

best_distance , best_path =  best_route(locations, no_of_iterations)
```

3.1 Code for Animation.

[ ]:
```
fig, ax = plt.subplots()
def onestep(frame):
    global best_distance
    ax.clear()
    distance = total_distance(best_path[frame])

    ax.set_title('Current Distance = '+str(np.round(distance, decimals=2))+'\n'
                 +"Best Distance = "+str(np.round(best_distance[frame],␣
 ↪decimals=2))+
                '\n Iterations = '+str(frame) )
    ax.plot([best_path[frame][i][0] for i in range(no_of_cities+1)],␣
 ↪[best_path[frame][i][1] for i in range(no_of_cities+1)])
    ax.plot([best_path[frame][i][0] for i in range(no_of_cities+1)],␣
 ↪[best_path[frame][i][1] for i in range(no_of_cities+1)], 'ro')
ani= FuncAnimation(fig, onestep, frames=[(no_of_iterations//no_of_frames)*i for␣
 ↪i in range(no_of_frames+1)], interval=100,repeat=False)
plt.show()
```

3.2 Inferences:

- For 10 cities minimum distance was found to be 34 units for 1000 iterations.
- For 100 cities minimum distance was found to be around 83 units for 50000 iterations.