EE2703 - Week 7

Sidharth S Kumar EE21B130

March 30, 2023

1 Importing Libraries

[11]:
```
%matplotlib ipympl
import numpy as np
import matplotlib.pyplot as plt
from functools import partial
```

---

2 General annealing function

[12]:
```
def Annealing(cost, random_jump, init_guess, T, decay,k, iterations):

    x = init_guess
    c = cost(x)

    # Iteration over the given range to find optimum using Annealing
    for i in range(iterations):

        # Taking a new random guess in the temp range
        x_ = random_jump(x,(np.random.random_sample() - 0.5) * T)
        c_ = cost(x_)

        # Confirm jump if its better
        if c_ < c:
            x = x_
            c = c_

        # Conditional jump with Arrhenius Probability
        else:
            toss = np.random.random_sample()
            if toss < np.exp(-(c_ - c) / (k * T)):
                x = x_
                c = c_

        # Modifying the temp
        T *= decay
```

```
        return x
```

Here I define a general function to perform annealing to minimize a function. THe function taking parameters - Cost function - Initial Guess - Initial Temperature - Decay rate and k - Random jump distance - Number of iterations

---

3 Travelling Salesman

3.1 Defining basic functions

[13]:
```
# Function to find the cost of each of the paths
def tsp_cost(points, path):

    sum_dist = 0

    # Iterating over all the points in the order of path to give the current␣
 ↪path length
    for i in range(-1, len(path) - 1):
        sum_dist += np.linalg.norm(points[path[i]] - points[path[i + 1]])

    return sum_dist

# Function to perform random swaps between two points
def tsp_swap(path, distance):

    swaps = np.random.randint(int(np.abs(distance)) + 1) + 1
    new_path = path.copy()

    for _ in range(swaps):
        swap0, swap1 = np.random.randint(len(path), size=2)
        tmp = new_path[swap0]
        new_path[swap0] = new_path[swap1]
        new_path[swap1] = tmp

    return new_path
```

The above cell defines two general functions that are used in the travelling salesman problem extensively. One to find the cost of any path that is chosen and other to find the random swap between two points in the path

3.2 Choosing input file

3.2.1 10 cities, 500 iterations

[14]:
```
input_filename = "tsp_10.txt"
start_temp_scale = 1/5
iterations = 500
```

3.2.2 100 cities, 5,000 iterations

[4]:
```
input_filename = "tsp_100.txt"
start_temp_scale = 1/10
iterations = 5000
```

3.2.3 100 cities, 30,000 iterations

[ ]:
```
input_filename = "tsp_100.txt"
start_temp_scale = 1/5
iterations = 30000
```

3.3 Read the list of cities

[15]:
```
lines = open(input_filename, "r").readlines()

# List of points(given)
cities = []

for line in lines[1:]:
    s = line.split()
    cities.append(np.array([float(s[0]), float(s[1])]))
```

3.4 Get result

[16]:
```
# Creating an intial path of random order
init_path = list(range(len(cities)))

# FInding the optimum path using Annealing
optimum_path = Annealing(partial(tsp_cost,␣
 ↪cities),tsp_swap,init_path,len(cities) * start_temp_scale,0.99,0.
 ↪1,iterations,)

# Printing the results
print(optimum_path)
print(tsp_cost(cities, optimum_path))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
56.41824974726559
```

The above cell starts by making an intial random guess for the path and then optimize it using annealing.. Then prints both the result and the cost of the total distance

3.5 Plot result

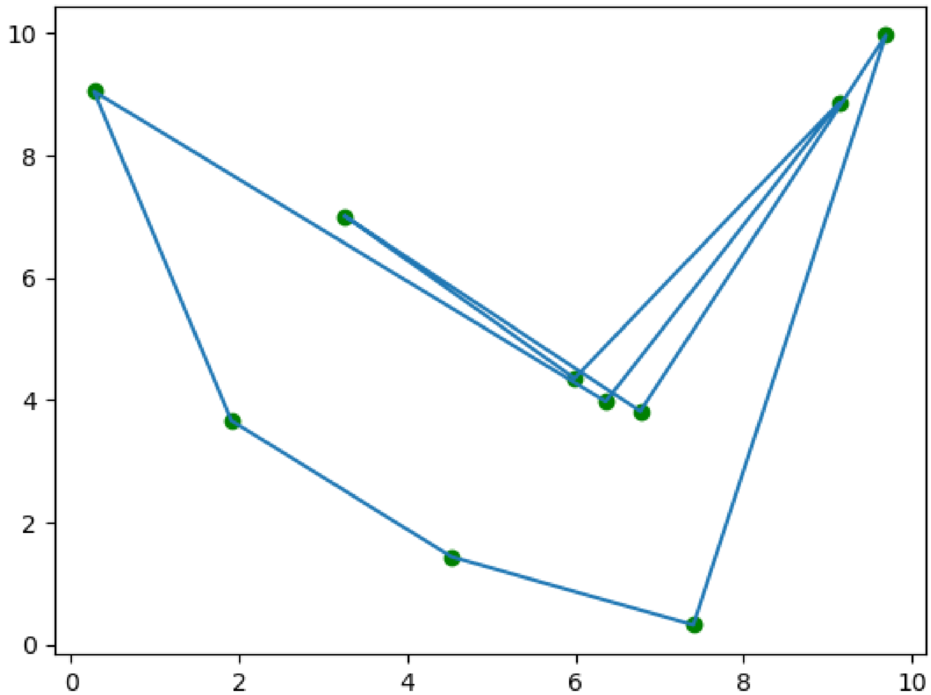
[17]:
```
# Create a new figure and axes objects
fig, ax = plt.subplots()

# Plot the cities as green circles using their coordinates from the cities list
lnpoints, = ax.plot([cities[i][0] for i in range(len(cities))],[cities[i][1]␣
 ↪for i in range(len(cities))],'go')

ln, = ax.plot([], [])

# Create a list of coordinates for the TSP path by referencing the indices in␣
 ↪the optimum_path list
coords = []
for c in optimum_path:
    coords.append(cities[c])
coords.append(cities[optimum_path[0]]) # Complete the path by adding the␣
 ↪starting city as the endpoint
coords = np.array(coords)

# Set the data for the plot line object to be the coordinates of the TSP path
ln.set_data(coords[:, 0], coords[:, 1])
plt.show()
```



10
8
6
4
2
0
0
2
4
6
8
10