Symmetric Travelling Salesman Problem

Large Scale Optimization for Data Science

Kale-ab Tessera, 1973752

,

The TSP (Travelling Salesman Problem) is an optimization problem, where a salesman or agent, wants to visit **n** distinct cities (represented as vertices), while choosing the shortest **distance travelled/cost** (represented by edges). The problem can be rephrased as finding a **"Hamiltonian circuit with minimum cost/length"**.

In this implementation, we use the 2-opt Heuristic approach on a Symmetric Travelling Salesman Problem, where we begin with an initial T and replace 2 edges with two new edges, with the aim to find a Tour with a smaller length/cost. This algorithm gradually improves and sets the current T to the **local minimum in it's Neighbourhood** (a set of two adjacent tours).

In [151]:

```
import numpy as np
class TravellingSalesman:

    def __init__(self, initialT, d_ij, showTourOutput = False):
        #Step 1 - InitialT & d_ij is set to what is passed in.
        self.T = initialT
        self.d_ij = d_ij
        # Step 2.1 - Set D to be length of tour T
        self.D = self.calculateLengthOfTour(initialT)
        self.showTourOutput = showTourOutput

    def calculateLengthOfTour(self,tour):
        sumDistance = 0
        for ind in range(len(tour)):
            v_1 = tour[ind]

            #If at the end of tour, use home V
            if(ind == (len(tour) -1 )):
                v_2 = tour[0]
            else:
                v_2 = tour[ind+1]

            sumDistance += self.d_ij[v_1][v_2]

        return sumDistance

    def runTS(self):
        N = len(self.T)

        #Step 2.2 - i = 1, i = i +1, while i < N-3 (Python indexs begin at 0)
        for i in range(N-3):
        #Step 3 - j = i +2, j = j+1, while j < N-1(Python indexs begin at 0)
            for j in range(i+2,N-1):
                    newT = np.copy(self.T)

        #Step 4.1 - Break link (i,i+1) - create link (i,j)
                    newT[i+1] = self.T[j]
                    ind = i +2
        #Step 4.2 - Break link (j,j+1) - create link (i+1,j+1)
        #Change the order of the edges to follow the new edges created
                    for k in range(j-1,i,-1):
                        newT[ind] = self.T[k]
                        ind += 1
                    newT_D = self.calculateLengthOfTour(newT)

                    if(self.showTourOutput == True):
                        print("i: ",i,"j: ",j,"Possible T",newT, "Cost: ", newT_D,
"CurrentBestCost:", self.D)

        #Step 4.3
        # If cost(newT) < D then set T := newT & update D & go back to Step 2 - self.runTS()
            # else
```

```
                # continue Step 3 (j = j + 1 and continue while j < N)
                # or continue Step 2.2 (increase i, while i< N-2)
                    if(newT_D < self.D):
                        self.T = np.copy(newT)
                        self.D = np.copy(newT_D)
                        self.runTS()
```

In [152]:

```
#Step 1 - Setting InitialT and D_ij
#Equivalent to ([2,6,1,7,5,3,4]), starting at index 0
T = np.array([1,5,0,6,4,2,3])
D_ij = np.array(
        [ 0, 1, 3, 5, 2, 1, 1,
          1, 0, 1, 6, 9, 4, 3,
          3, 1, 0, 1, 5, 3, 2,
          5, 6, 1, 0, 1, 2, 5,
          2, 9, 5, 1, 0, 1, 6,
          1, 4, 3, 2, 1, 0, 1,
          1, 3, 2, 5, 6, 1, 0
        ])
D_ij = D_ij.reshape(7,7)

#Passing in initialT and D_ij
TS = TravellingSalesman(T,D_ij,False)
```

In [153]:

```
TS.runTS()
print("Shortest Distance: ",TS.D)
print("Best Path: ", TS.T)
```

```
Shortest Distance:  14
Best Path:  [1 0 4 5 6 2 3]
```