Simulated Annealing

Large Scale Optimization for Data Science

Kale-ab Tessera, 1973752

,

Implementation of the Simulated Annealing Algorithm in solving the (Symmetric) Travelling Salesman Problem.

In the outer loop of our algorithm, we reduce T by setting T = 0.9 $\times$ T. While, in the inner loop, we obtain *L* different solutions at each T. These solutions are generated by randomly swapping two indexs(cities) of tour x.

Then we apply the Metropolis acceptance Probability=min $\{1, exp(-(fy - fx)/T)\}$ as the probability of accepting a new possible solution. The algorithm then repeats while T $> \epsilon$, with $\epsilon = 10^{-2}$, and stops otherwise.

In [5]:

```
import numpy as np
import math

class SimulatedAnnealing:
    def __init__(self,x,M,T,n,L,E):
        #Step 1 - Initialize values to what is passed in.
        self.initialX = x
        self.M = M
        self.T = T
        self.n = n
        self.L = L
        self.E = E

    # Distance function to calculate length of tour
    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.M[v_1][v_2]

        return sumDistance

    # Function to generate a new tour, with 2 cities swapped at random
    def generateNewTour(self,x):
        # Step 3.1 - 3.2 - replace = False ensures no duplicates indexs
        # Choosing two non-repeating random numbers for indexs
        rangeForRandomNumbers = np.arange(len(x))
        uniqueRandomIndexs = np.random.choice(rangeForRandomNumbers,2,replace=False)
        n1 = uniqueRandomIndexs[0]
        n2 = uniqueRandomIndexs[1]

        # Step 3.3 - 3.4 - create y
        y = np.copy(x)
        temp = y[n1]
        y[n1] = y[n2]
        y[n2] = temp

        return y
    def run(self):
        x = self.initialX
        fx = self.calculateLengthOfTour(x)
        T = self.T
        E = self.E
        # Step 2 - looping while T > E
        while T > E:
            # Step 3 - run for length L
            # L - The total number of solutions generated at a particular temperature.
            for i in range(L):
                # Step 3.1 - 3.3 - create new tour y
                y = self.generateNewTour(x)
                # Step 3.4 - evaulate y
                fy = self.calculateLengthOfTour(y)

                # Step 4
                if(fy < fx):
                    x = y
                    fx = fy
                else:
                    # Generate random number between 0 and 1
                    randomNum = np.random.rand()
                    # Metropolis acceptance Probability
                    # exp(-(fy-fx)/T) function is same as e^(-(fy-fx)/T)
                    if(randomNum < math.exp(-(fy-fx)/T)):
                        x = y
                        fx = fy

            # End forloop
            # Step 5 - Decreasing T
            T = 0.9*T

        #End while loop

        #Return best tour x and best fx
        return x,fx
```

In [6]:

```
#Step 1 - Initialize Values
T = 15
n = 7
L = 20
E = 10 ** -2
#Equivalent to ([1,7,2,4,6,5,3]), starting at index 0
x = np.array([0,6,1,3,5,4,2])
M = 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
        ])
M = M.reshape(n,n)
SA = SimulatedAnnealing(x,M,T,n,L,E)

# Run simulated annealing algorithm
bestx,bestfx = SA.run()
print("Best tour: ",bestx,"Best fx: ",bestfx)
```

```
Best tour:  [2 1 0 6 5 4 3] Best fx:  7
```