SIMULATED ANNEALING & TRAVELLING SALESMAN PROBLEM

Problem Statement:

Given a list of city coordinates, we try to optimize the path taken by the salesman so that the path results in the least possible distance to be covered by the salesman. The salesman returns to the starting city at the end. Technically this should consider all possible permutations of cities and choose the minimum. Given N cities, the problem explodes to considering N! cases and the complexity is O(N!). Hence the problem is termed NP Hard. We try to obtain a solution that has less time complexity.

Approach:

We start off by considering a random order of cities initially, and work on improving this order from time to time. We use the technique of comparative optimization to obtain a better path every time.

HOW DO WE OBTAIN THIS BETTER SOLUTION?

At any given time, we call the present order of the cities as STATE. We randomly swap two numbers in the STATE, which is essentially changing the order we have and whether we accept this new state or not is figured out by the function find_next using COMPARITIVE OPTIMIZATION.

def find_next:

This function takes in the given state and two elements that are going to be swapped. It creates a TEMPORARY STATE by swapping the two elements. It checks if this new solution is any better by comparing the total distance covered in the new temporary state versus the old state. If the distance is less, the solution is better and it accepts TEMPORARY STATE as the current STATE. If not, it might or might not accept this new solution.

Here we bring in the concept of SIMULATED ANNEALING, where we calculate the cost of accepting this new solution as the difference between the distances obtained.

We define the probability of accepting this new solution as:

$$P = e^{\left(-\frac{COST}{TEMPERATURE}\right)}$$

Where,

COST= DISTANCE TRAVERSED IN TEMPORARY STATE- DISTANCE TRAVERSED IN OLD STATE
TEMPERATURE= is a constant that is pre defined and decays with a pre defined decay rate after specified interval of time.

With a probability P, the model accepts the inefficient TEMPORARY STATE as the CURRENT STATE. Else, it discards the Temporary state and RETAINS THE CURRENT STATE.

TIME COMPLEXITY:

We reduce the time complexity of the problem to O(N^3) which is better than O(N!). That is we do this swapping N^3 times to generate the final order from the initial random guess.

HOW DO WE SWAP ELEMENTS IN A GIVEN STATE?

We follow a systematic procedure for swapping the elements rather than random picking and swapping. We find that thus us better because it makes sure that every element in the order contributes to the optimization. We call N^2 swaps an epoch and we run N epoch.

Each element is swapped N times. Given an index i, we randomly pick an index k>i for swapping and generate temporary state. We calculate the percentage improvement after every epoch and study the improvement in the solution.

After N epochs we return the final order. We calculate the distance travelled according to the final order and compare it with that obtained from the initial random guess.

Results:

For sample, a dataset was created tsp_40.txt which contains 30 datapoints. The dataset has been attached in the submission file.

The results and plots for the same are attached below:

OUTPUT:

Results of run1:

started with distance: 22.631200635808142

order= [31, 14, 4, 23, 36, 39, 19, 32, 16, 37, 26, 3, 30, 22, 13, 24, 33, 8, 15, 35, 17, 18, 6, 7, 1, 5, 27, 2, 38, 28, 11, 12, 29, 21, 25, 0, 20, 9, 34, 10]

epoch= 0 percentage improvement= 8.470954766146749

epoch= 1 percentage improvement= 6.7142526772612126

epoch= 2 percentage improvement= -8.620343874905027

epoch= 3 percentage improvement= 0.31798399684512924

epoch= 4 percentage improvement= 5.344834906105642

epoch= 5 percentage improvement= -0.4594135414385764

epoch= 6 percentage improvement= -0.3666834392454067

epoch= 7 percentage improvement= 6.757813002293381

epoch= 8 percentage improvement= -6.461429452179994

epoch= 9 percentage improvement= 7.921717154392438

epoch= 10 percentage improvement= -4.684096176826706

epoch= 11 percentage improvement= 6.864795413937665

epoch= 12 percentage improvement= 19.959393771636993

epoch= 13 percentage improvement= -7.895032783895033

epoch= 14 percentage improvement= 11.734295662602621

epoch= 15 percentage improvement= 20.390327143193474

epoch= 16 percentage improvement= 2.292156973022367

epoch= 17 percentage improvement= 27.17979299220124

epoch= 18 percentage improvement= -6.605163563199399

epoch= 19 percentage improvement= 5.791874344770228

epoch= 20 percentage improvement= 12.372354652894279

epoch= 21 percentage improvement= 3.41276529607533

epoch= 22 percentage improvement= -0.8077620633235973

epoch= 23 percentage improvement= -0.20388409621387382

epoch= 24 percentage improvement= 0.868589251872396

epoch= 25 percentage improvement= -11.963435680638284

epoch= 26 percentage improvement= 13.513300437330889

epoch= 27 percentage improvement= 0.17608184058655557

epoch= 28 percentage improvement= 1.1593881152194134

epoch= 29 percentage improvement= -1.8164953010208176

epoch= 30 percentage improvement= -0.30064280269015603

epoch= 31 percentage improvement= 2.3042498638989715

epoch= 32 percentage improvement= 0.19117727982493732

epoch= 33 percentage improvement= -4.236716616628871

epoch= 34 percentage improvement= 3.8257235444665616

epoch= 35 percentage improvement= -1.6374166732919444

epoch= 36 percentage improvement= 0.6713913101369434

epoch= 37 percentage improvement= -0.2054650089295972

epoch= 38 percentage improvement= 0.9292369231193962

epoch= 39 percentage improvement= 1.1168574568983238

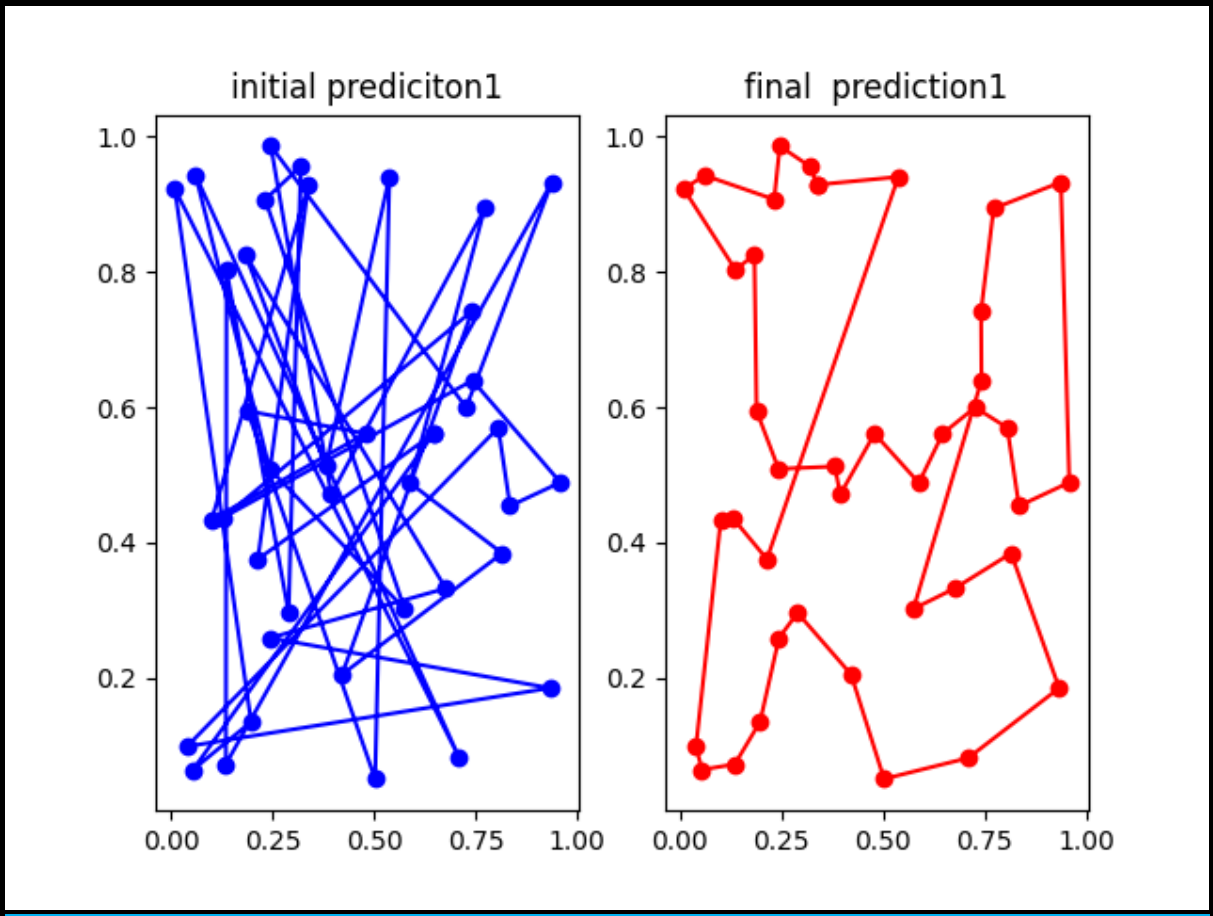
final_distance= 6.181416327119386

final order= [4, 32, 14, 15, 23, 22, 10, 38, 34, 3, 11, 17, 19, 20, 30, 18, 26, 21, 8, 35, 7, 2, 33, 24, 13, 27, 31, 6, 36, 1, 16, 25, 39, 37, 28, 9, 0, 29, 5, 12]

percentage_improvement= 72.68630848803107

total time taken= 6.072789199999988

PLOTS FOR INITIAL AND FINAL ORDER:



Results of run2:

started with distance: 21.606536497530826

order= [13, 22, 12, 18, 14, 17, 23, 1, 0, 10, 20, 7, 31, 37, 15, 29, 3, 25, 27, 8, 38, 5, 2, 39, 19, 26, 30, 36, 24, 32, 16, 21, 9, 4, 34, 11, 33, 6, 28, 35]

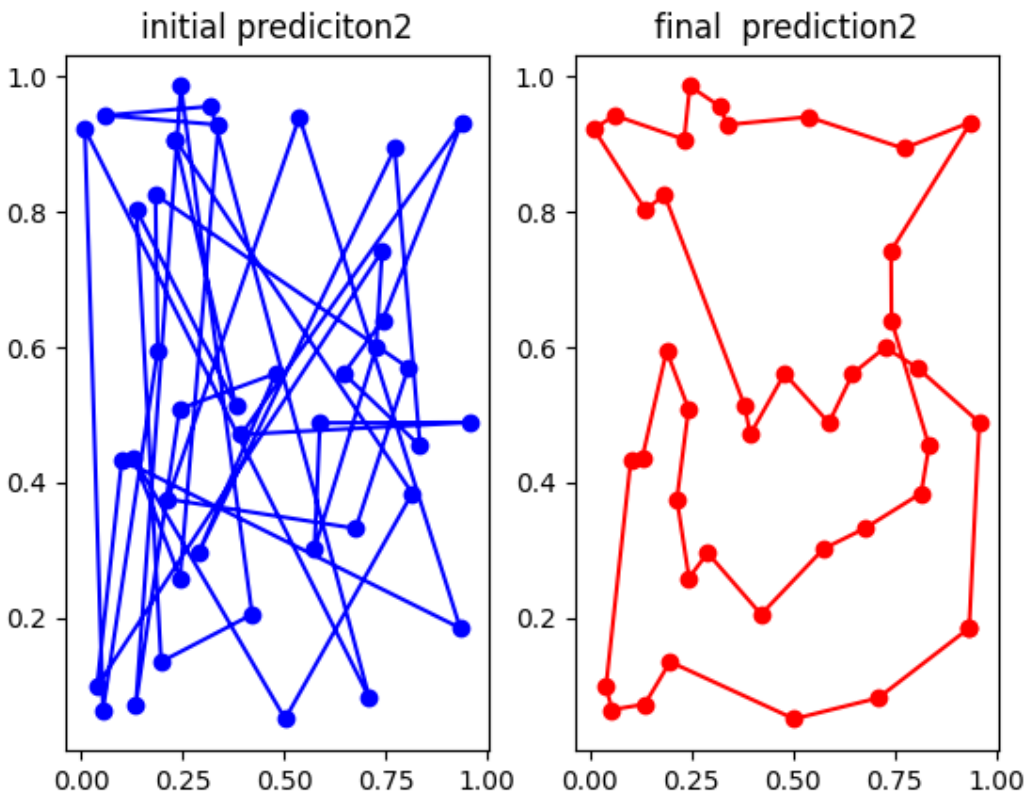
final_distance= 6.084058776703163

final order= [3, 22, 10, 38, 21, 26, 27, 13, 33, 24, 18, 30, 20, 19, 17, 1, 16, 36, 6, 31, 8, 35, 2, 7, 32, 4, 12, 5, 29, 0, 9, 28, 37, 39, 25, 15, 23, 14, 34, 11]

percentage_improvement= 71.84158239614924

total time taken= 6.43567679999978

PLOTS FOR INITIAL AND FINAL ORDER:



ANIMATION FOR THE SAME IS ATTACHED IN THE FILE

OBSERVATIONS:

1. Every run of the code gives different results, but every optimization is at least 50%
2. Systemised swapping is better than random swap
3. The solution obtained is not the optimized solution
4. If percentage improvement after an epoch is 0, it means path is unchanged

LIMITATIONS:

1. More number of runs can give better results
2. Unique results aren't obtained from this model.

STEPS TO RUN THE CODE:

1. Sample test case is embedded in the file, just running the code gives the corresponding results
2. For some other testcase call the function tsp() with parameter as the list of coordinates of the cities.