CS6380:Artificial Intelligence

Assignment-1 | Report

Name: Nithin Uppalapati **Roll:** EE18B035

Date: 16 Oct. 21

Problem Statement:

TSP – Travelling Salesman problem, tasked with finding the shortest route between a set of points and locations that must be visited. In the problem statement, the points are the cities a salesperson might visit. The salesman's goal is to keep both the travel costs and the distance travelled as low as possible.

It was first described by Irish mathematician W.R. Hamilton and British mathematician Thomas Kirkman in the 1800s through the creation of a game that was solvable by finding a Hamilton cycle, which is a non-overlapping path between all nodes.

Difficulties in Solving the Problem

As this is a NP-hard problem, there is no deterministic way of finding the optimal path with less than O(n!) complexity. In case if we attempt to solve the problem by a deterministic search method, the complexity will be O(n!) and thus a combinatorial explosion in time complexity. In order to solve the problem, I've explored few of Stochastic processes such as SA, GA, LBSA.

Generating Datasets:

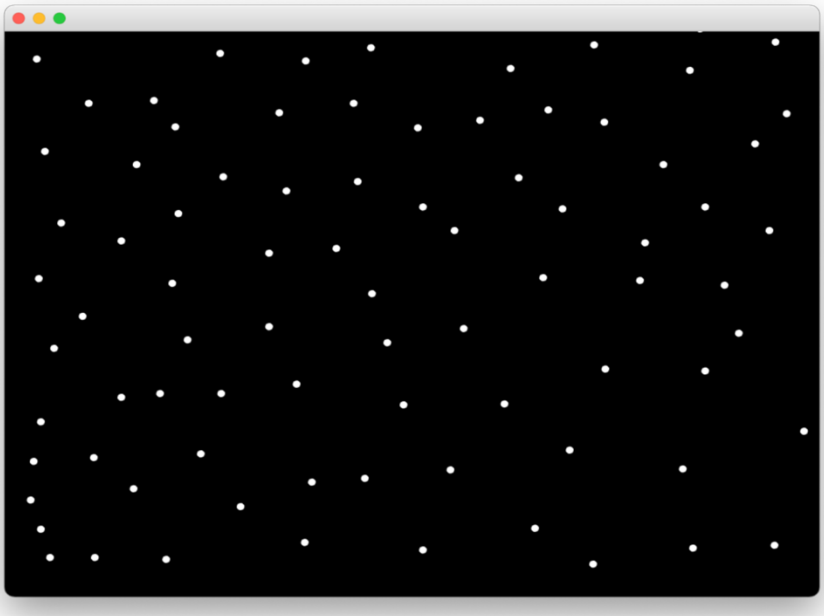
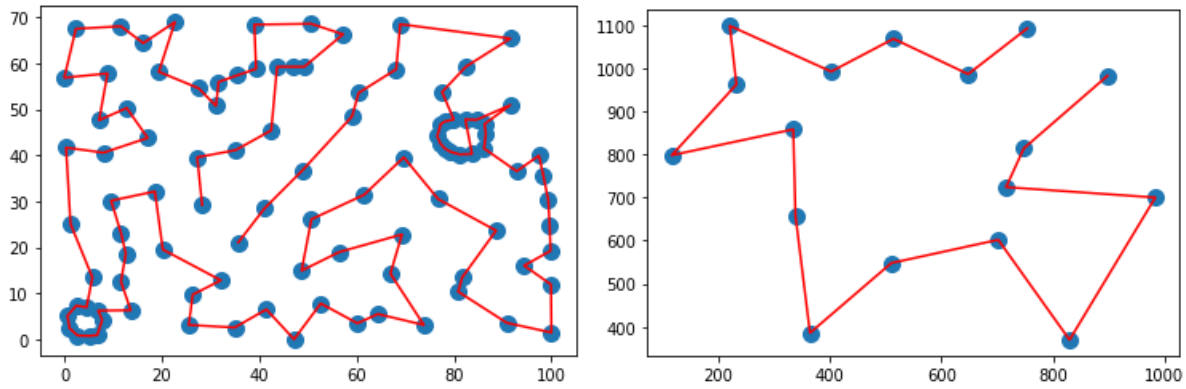


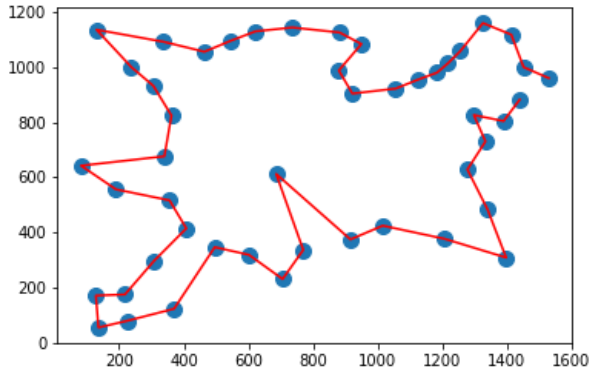
Fig -1

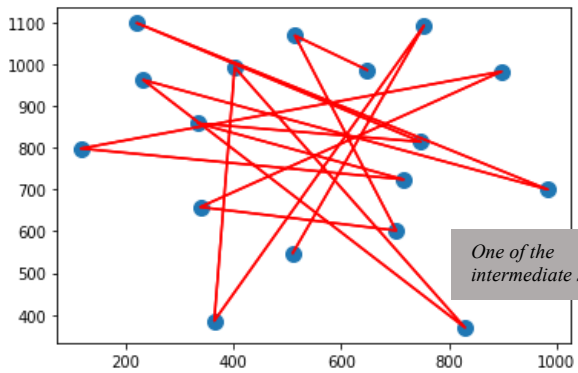
I have implemented a program in python3, which asks for users to input the datapoints from a GUI Input mode and creates a ***stdin.txt*** file in a folder. So we can just create the dataset as simple as placing dots in the GUI pop-up window. One point to note is that this method creates only the dataset which has Euclidean mode of measure. *Above is the (Fig-1) window in which it has 82 dots, corresponding to 82 cities.*

Evaluating the Solution to the created Datasets:

I have visualised the paths in Python Plots, which helped me whether in understanding the performance of my Algorithm implementation.







One of the intermediate solutions

The blue dots are the cities, and the red lines are the paths.

Approaches in Solving the Problem:

1. Simulated Annealing - SA
2. Genetic Algorithm - GA
3. List Based Simulated Annealing - LBSA

Metropolis acceptance criterion : ***probability compared to uniform(0,1)***

Simulated Annealing (SA):

This method is also called as Stochastic Hill climb. It is a combination of best first search and random walk. I have implemented this

Method: In simulated annealing, we give the initial temperature and also, we give define a proper cooling function and according to those temperatures we calculate the probabilities of acceptance (Metropolis acceptance). As the search proceeds, we gradually reduce the temperature and thus with an external termination, we will terminate the search process after few epochs.

First drawback of this problem is there is a fixed initial temperature to start with and then the cooling process is also fixed, which is not good. In-order to improve this method we have to somehow make the temperature and it's cooling process to be also dependent on the size of the problem (no. of cities).

Second drawback of this problem is, the ΔE, which is the nature of Evaluation. As this is directly the difference between the costs, we encounter a problem while doing the Metropolis acceptance criterion. As $\Delta E/T$ is required for the sigmoid function in order to compare the acceptance prob. with the uniform prob. and recall that we have a fixed temperature to start with, so if ΔE is more than the acceptance is more and vice versa. To avoid this process, I normalised the co-ordinates (confine them to 100x100 grid) so that now we have an upper limit on ΔE.

To conclude, this process really works well with lesser number of cities (<50). And as the number of co-ordinates increases, we have to explore different cooling methods in-order to solve them too.

Genetic Algorithm (GA):

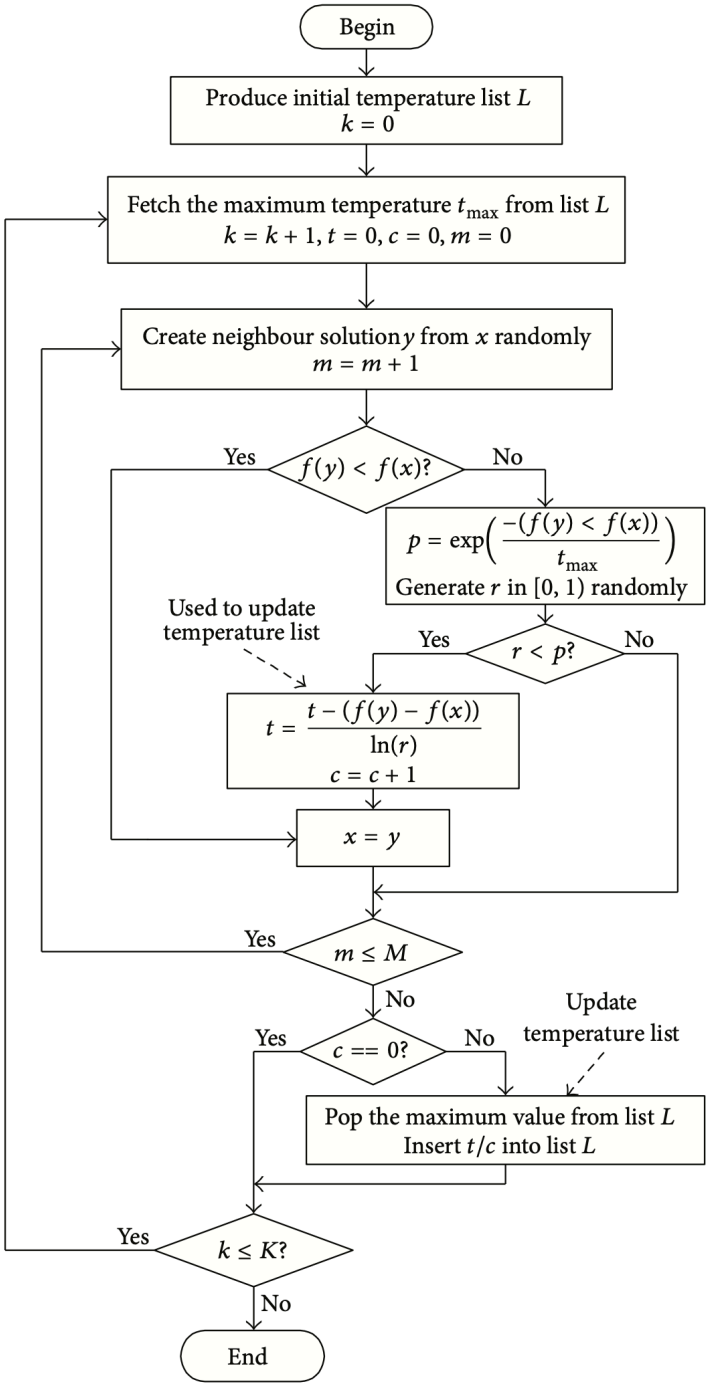
One of the initial observation is that, GA is slow when compared to SA. It performs better if we give it more time.

The crossover method which I used is the Cyclic Crossover. The mutation process I used is, randomly swapping/inserting/inverting and two given nodes on the path.

Method: The parameters I defined were: ***mutation_prob***, ***crossover_prob*** which are useful in defining the mutation probability and crossover probability respectively. Given a pool of parents (mating pool), we randomly pick two parents according their fitness, and then we may decide to crossover them with a prob - ***crossover_prob***. Then after the crossover process, we can randomly pick the new generation off-springs with prob - ***mutation_prob***, and the process of mutation is, we randomly perform swapping/inserting/inverting on two given random nodes on the path. (*Swap, Insert, Invert are defined below, see Fig: 2, 3, 4*)

To Conclude, in general, this algo takes more time to arrive at the optimal solution. One point to mention is combination of Cyclic crossover and SA will also yield slightly better results in time aspect.

List Based Simulated Annealing:



Simulated annealing (SA) algorithm is a popular algorithm but its parameters' setting is a key factor for its performance, depending on the problem statement at hand. In order to make the parameters adaptable to the current best solution, I maintained a list of temperatures which then will be used for simulated annealing to solve traveling salesman problem (TSP). This is called LBSA algorithm. It uses a list-based cooling schedule to control the decrease of temperature. Specifically, a list of temperatures is created first, and then the maximum temperature in list is used by Metropolis acceptance criterion to decide whether to accept a candidate solution. The

temperature list is adapted iteratively according to the topology of the solution space of the problem.

In this method the way of producing the neighbour is mentioned as follows: Let's π be the current path (current "state", *although strictly speaking state is the wrong term to use in these kinds of solution space problems*) and π' be the neighbour produced from the current path π. Now, I produce three paths from the existing path and choose the best path among them in-order to produce a neighbour π'.
The three methods are:

- **Swap**: This operation swaps two cities in a given path. Thus, in general a total of 4 edges are to be changed. (*See Fig-2*)

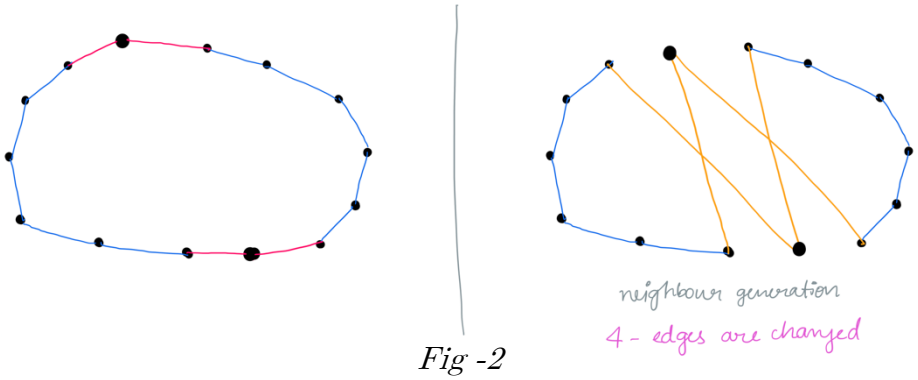


Fig -2

- **Inverse**: This operation swaps any two random edges, and thus two edges are changed. (*See Fig-3*)

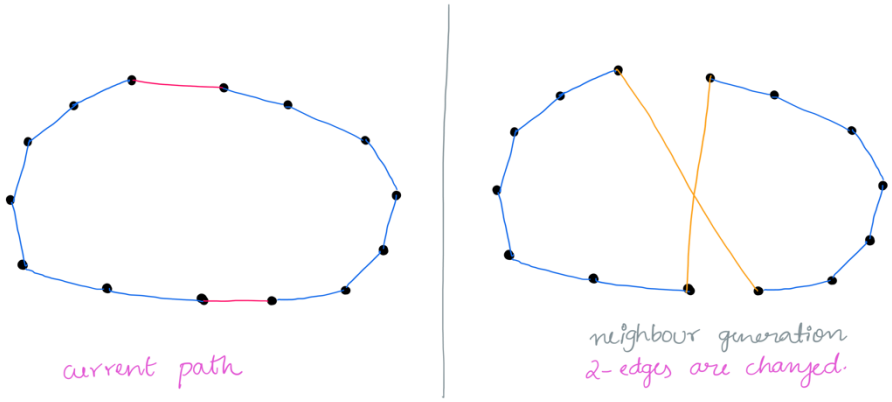


Fig -3

- **Insert**: This operation inserts a city 'i' in place of city 'j'. In general, three edges can be changed. (*See Fig-4*)

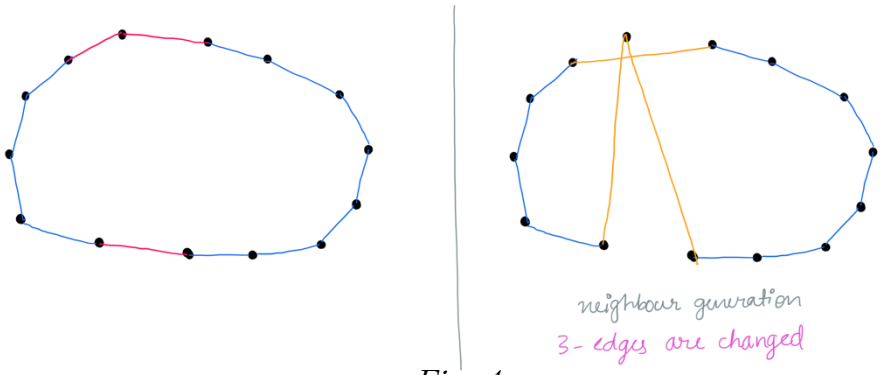


Fig -4

(The above three methods of creating neighbours is also used in the Genetic Algorithm implementation)

More the length of the temperature list, slower is the cooling of the temperature. Instead of the initialising the temperature in SA, here in List Based-SA we initialise the probability. The inner loop's iterations are controlled by the parameter "M".

Method: As the name suggests, we maintain a list of temperatures here, initially they are quite high which aids in random exploration. With each iteration, we reduce the temperature probabilistically, depending on how better the solution is. Once a candidate is selected according to Metropolis acceptance criterion, if the selected candidate is better, then we reduce the temperature and if the selected candidate is worse than current best solution, then we intentionally increase the temperature hoping that there is still more exploration to do...

So, unlike in SA, here the temperature won't always decrease monotonically, thus performing better than naïve SA implementation. One more **advantage** when compared to SA is that, the nature of generation of the neighbour candidates (paths). Here we have a hybrid generator, which choses the best among the three produced neighbours by the above three mentioned methods.

By performing a grid search, I've arrived at optimal parameters for the problem statement, there is still a scope of improvement here to adjust the parameters accordingly given the size of the problem (number of cities).

Out of the above three mentioned methods - SA, GA, LBSA; LBSA performed well in my experimentation.

END