TRAVELLING SALESMAN PROBLEM

COMPLEXITY OF THE PROBLEM:

The direct solution for this problem would be to try all possible permutations. For N cities, the time complexity would be O(n!). The brute force solution is non-polynomial. This solution is not feasible for cities > 15.

HELD – KARP ALGORITHM:

It is a dynamic programming algorithm. It avoids the repeated computation of sub-problems by storing them, thereby reducing the time complexity. The time complexity of this solution is O((n^2)*(2^n)). Though this is faster than brute force solution, it doesn't work for cities > 16. This algorithm is exponential. We need a better algorithm, which works in polynomial time.

NEAREST NEIGHBOURS:

It is a greedy algorithm which takes a random city as starting point and then proceeds by visiting the nearest unvisited city. This algorithm terminates when all the vertices are visited. It quickly yields a short tour, but not the optimal one. The time complexity of this problem is O(N). Here, N is the number of cities.

What's wrong with NEAREST NEIGHBOUR?

The solution depends on the initial city and the geometric distribution of cities. Even for the same geometric distribution, if we change the initial city we get a different solution.

Fixing NEAREST NEIGHBOUR algorithm:

To fix the problem, we can a fixed city as seed and then proceed with the algorithm. But, fixing a seed results in poor solutions because this algorithm depends on the geometric distribution of cities.

Let us consider each city as starting point and then proceed the algorithm by keeping track of the best solution. The time complexity of the problem is O(N^2). Here, N is the number of cities.

Refined Algorithm:

1. Take the first city(0) as starting node
2. Traverse the nearest unvisited city
3. Stop when there is no city to visit and record the tour cost
4. Set vertices 1 to n-1 as starting nodes and repeat steps 2 and 3
5. Chose the best tour cost of all the tours

After applying a fix to the **NEAREST NEIGHBOUR** algorithm, it is giving good results to all the cost functions with any number of cities. But, it won't give good results in practical scenario.

Under the assumption that each tour is a state, each time a city is taken as start city, we form a tour. The tour of N cities consists of N paths or links(it is a circular tour). To take each path we need to check all the paths from the city where the salesman is present(He/she examines N paths). So to take one path he checks N-1 paths. To complete the tour he needs to check N*(N-1) paths. Also, each city is considered as start point and the algorithm is repeated. So total number of links checked = N*N*(N-1).

We consider a tour as a state. When a city is taken as start state , we get a tour using Nearest Neighbors algorithm. Each city taken as start state, so N tours are formed. So number of states = N which is nothing but MEB. Here, N is the number of cities.

Number of links or edges explored = MEB^3 = O (N^3)

Results:

NN	**C1**	**MEB**	**C2**	**MEB**	**C3**	**MEB**
10	426	10	58	10	1050	10
20	446	20	101	20	9500	20
30	466	30	156	30	33350	30
40	486	40	192	40	80600	40
50	506	50	221	50	159250	50
60	526	60	247	60	277300	60
70	546	70	276	70	442750	70
80	566	80	304	80	663600	80
90	586	90	334	90	947850	90
100	606	100	364	100	1303500	100
110	626	110	396	110	1738550	110
120	646	120	424	120	2261000	120

A STAR APPROACH:

A star is an informed search algorithm, which tries to reach the goal state from start state. It explores all the possible paths from a given state and choses the one which is closest to the goal state. It uses heuristics to estimate the approximate distance from a given state to the goal state. The heuristic should be consistent or admissible. The heuristic should never over estimate the actual cost to get to the nearest goal state. This is also called best-first search.

Admissibility of the Heuristic:

The heuristic is problem specific. One should check the admissibility of the heuristic for the given problem before selecting it. The distance estimated by heuristic function should be less than or equal to the actual distance to the goal state. If estimated distance to the goal state is greater than actual distance, then we will end up getting worst solutions to the problem by choosing bad edges. For TSP, we can choose **Minimum Spanning Tree** as the heuristic. We can assume the distribution of cities as a complete graph as every city is connected to every other city. In any graph, the distance between any vertices is less than the minimum spanning tree distance connecting those vertices. This is because the minimum spanning tree is constructed by taking the edges in increasing order (Kruskal's algorithm).

Parameters of A star:

1. **States**: We use a node to store the information of state that includes city number, parent from which the present city is reached, distance from initial city, tour cost via the city and a list of the size of the number of cities. The entry in the list is set to 1 if it is visited and 0 if it is not visited
2. **Initial state**: For the first city the parent is NULL and all the entries in the list are 0's.
3. **Actions**: Picking an edge is the action.
4. **Transition model**: Picking an edge takes the system to another state.
5. **Goal test**: When all the entries in the list are set to 1, it means that the tour is completed.
6. **Path cost**: Path cost is given by the cost function.

Algorithm:

1. Create a priority queue which stores "PriorityQueueNode". This PriorityQueueNode has the state. These nodes are stored in priority queue based on "tourCostViaVertex".
2. Select a random city as start node and add it to the priority queue
3. Check whether the priority queue is empty or not and pop the first element from it.
4. Set the popped city as explored
5. Check for Goal test
6. If Goal state is reached stop the algorithm and print the results, else explore the neighbors
7. While exploring neighbors, calculate the tour cost using g (n) + h (n) and add them to the priority queue
8. g (n) is the distance of the city from the starting city and h (n) is the heuristic distance to the goal state.

9. h (n) = distance to the nearest unvisited city from the current city + estimated distance to travel all the unvisited cities (MST heuristic is used) + nearest distance from an unvisited city to the start city.
10. Repeat steps 3-9 until the queue is empty or the Goal state is reached.

Results:

ASTAR	**C1**	**MEB**	**C2**	**MEB**
10	**427**	**38**	**68**	**93**
20	**456**	**320**	**107**	**1302**
30	**475**	**1472**	**217**	**62982**
40	**496**	**8569**	**Time >4 minutes**	**> 100000**
50	**518**	**132414**	**Time> 4minutes**	**>100000**

MEB is exceeding 100000 for 50 cities and it is taking more than 5 minutes. So, let us try Genetic approach.

GENETIC APPROACH:

It is a meta heuristic inspired from evolutionary Algorithms. It is guaranteed to provide close to optimal solutions relying on bio-inspired operators such as mutation, cross over and selection. It avoids the problem of getting trapped in local optima and facilitates a way to find close to optimal solution.

The initial generation of population is generated by random shuffling of cities. It is our choice to decide the size of initial generation. After the population is generated, we assign fitness to each tour in the population and sort them in increasing order of fitness. After this, we normalize the fitness. The fitness of each tour lies between 0 and 1 after normalizing. The mathematical formula behind fitness calculation is given below:

$$\text{Fitness} = 1/(d+1)$$

The new generation is to be developed from the existing generation. We need to make sure that the new generation is stronger than the older one. To accomplish this, we always pick two tours with high fitness from the old generation by using crossover and mutation operators and generate a new tour for the present generation.

Algorithm:

1. Create the initial generation

2. Calculate fitness and normalize the fitness of each tour in the initial generation
3. Produce new generation from the older generation using crossover and mutation operators
4. Normalize the fitness of the new generation
5. Repeat steps 3, 4 until termination criteria is met.

Are crossover and mutation operators good?

We should ensure that the crossover and mutation operators produce a valid tour. Applying these operators shouldn't include visiting of a city twice in the city and make sure that all the cities are visited. In the crossover operation, we pick two parents that are highly fit and then randomly select a sub tour from the first parent and add it to the offspring. Now, we go through the second tour and add the cities to the offspring that are not included in it. The crossover operator always ensures a valid tour. We also shuffle the tour generated from crossover by swapping cities. The amount of shuffling depends on the mutation rate. Excessive shuffling leads to bad solutions, so mutation rate should be picked wisely.

We continue the process of producing new generations from old generations and stop the process when a close to optimal solution is found. In this project, the population size is 10 and the process is repeated for 1000 generations. The number of states or tours explored are exactly 10,000. We can vary the initial population size and the number of generations. After trying several values for these parameters, the best values are found to be 10 for population size and 1000 for number of generation. Genetic algorithm is very fast and gives good results. But, it is highly randomized. Due to excessive usage of random generators in picking the initial generation, crossover and mutation, it gives a different tour each time it is ran for the same parameters. To fix this problem random generators can be used by fixing a seed, but it is difficult to control this. So let's try Simulated Annealing.

Results:

GA	**C1**	**MEB**	**C2**	**MEB**	**C3**	**MEB**
10	430	10000	60	10000	822	10000
20	494	10000	297	10000	7594	10000
30	555	10000	769	10000	26418	10000
40	623	10000	1822	10000	62978	10000
50	760	10000	3842	10000	124726	10000
60	923	10000	6110	10000	218488	10000
70	925	10000	12725	10000	349794	10000
80	1146	10000	13144	10000	521696	10000
90	1373	10000	14002	10000	743770	10000
100	1663	10000	27616	10000	1011182	10000
110	1765	10000	42509	10000	1368456	10000

120	2252	10000	62687	10000	1773266	10000

The above results are not fixed. Each time the program is executed, it will give different results. This is because of the random generators we are using.

SIMULATED ANNEALING :

It is inspired from the process of Annealing in metallurgy which involves heating and cooling of a material to change its internal structure. First we have to make sure whether it is appropriate to apply Simulated Annealing on Travelling Salesman Problem.

How is Simulated Annealing applied for TSP?

Here, we use a temperature variable to simulate this heating process. The most vital step in this algorithm is how one chooses the values of initial temperature and cooling rate. We need to experiment with different temperatures and cooling rates and measure their results to decide upon the final values.

Initially when the temperature is high, the algorithm accepts worst solutions than the current solution. This prevents us from getting trapped in local optimal solution. When the temperature is reduced, the algorithm accepts worst solutions reluctantly thereby keeping focus on an area of search space in which a nearly optimal solution is found. This gradual cooling makes simulated annealing algorithm effective in finding near optimal solution from a large search space.

Acceptance Function:

Hill climbing algorithm fails because it gets struck in local optima. It doesn't accept a solution, which is worse than the current solution. Sometimes we need to pass through worse solutions to achieve better solution than the current optimal solution. This is achieved in Simulated Annealing with the help of Acceptance function. The acceptance function always gives a value between 0 and 1. Higher the value of acceptance probability, higher is the acceptance of new solution. The mathematical formula behind Acceptance Function is given below:

exp ((currentTourCost – newTourCost)/Temperature)

Simulated Annealing algorithm:

1. Set initial temperature and cooling rate and select a random tour

2. Create a new solution by making small changes in the current solution. This change is typically a swap operation of two cities.

3. Check the feasibility of new solution with the help of Acceptance function and decide whether to move to it or not.

4. Repeat steps 2, 3 until stopping criteria is met. The stopping criteria might be a stopping temperature or the number of states explored.

Challenges associated with Simulated Annealing:

We get a different solution each time the algorithm is ran. The reason behind this problem is the selection of random tour initially. To fix this problem, we need to give a seed instead of the random tour. The game is not yet over. There is one more place we need to fix randomization. We swap the cities to get a new solution. Here the seed is taken randomly by the programming language implementation. Generally, all languages follow Fisher-Yates algorithm.

We can fix this problem by choosing the seed. This is achieved in java with the help of setSeed() function. Also, revert back the new solution if it is not better than the current optimal solution. With these changes, the Simulated Annealing algorithm gives the same solution every time you run it.

Results:

SA	**C1**	**MEB**	**C2**	**MEB**	**C3**	**MEB**
10	430	38372	63	38372	884	38372
20	470	38372	169	38372	7628	38372
30	510	38372	259	38372	26466	38372
40	546	38372	429	38372	62597	38372
50	590	38372	608	38372	123168	38372
60	628	38372	578	38372	213638	38372
70	670	38372	680	38372	337835	38372
80	710	38372	745	38372	507278	38372
90	750	38372	847	38372	725098	38372
100	790	38372	939	38372	992346	38372
110	826	38372	1074	38372	1323205	38372
120	870	38372	1274	38372	1719105	38372

The above results are fixed.

Time:

All the strategies took milli seconds to seconds to produce output except the A star algorithm.

Conclusion:

Travelling Salesman Problem is an NP-Hard problem. One cannot find the optimal solution the problem. But, close to optimal solutions can be obtained by following certain strategies. Each approach discussed above has its own merits and demerits. The nearest neighbor gave good results for all the cost functions given. But, Nearest neighbors might not work in practical scenarios because of its greedy nature. The A star algorithm with MST as heuristic is giving good results, but it is not working properly for N > 50. This can be improved by taking a better heuristic.

The Genetic algorithm gives good results. It is hard to come up with population size and the number of generations. Also, it is highly randomized . It is difficult to fix the seed and initial population. It gives different results each time it is ran. The Simulated Annealing algorithm is the best algorithm for solving TSP. We need to experiment with Temperature and cooling rate to find the right values. We can fix a seed and get the same output each time it is ran.