AI Assignment Report

Name: Akshay G Rao (CS21S002)

TSP is a NP-hard problem and has exponential time complexity. Hence I have used multiple algorithms which give an approximate solution.
I have used four algorithms to solve this problem:-

1. **Nearest Neighbour heuristic** algorithm
    - Start at a random starting point, add to tour and visit it.
    - Select the nearest (in terms of provided distance matrix) unvisited neighbor
    - Add this neighbor to tour and visit it
    - Consider this neighbor as the starting point and repeat step 1 till step 4 until all nodes are visited

The Nearest neighbor output may depend on the random starting point. Hence, repeatedly call this algorithm with different starting points. The starting point was chosen by dividing the number of cities into intervals of size two and picking a starting point randomly in each of these intervals.

The reason for choosing this algorithm is its much faster compared to genetic algorithms and gives a reasonable initial estimation generally. https://www.ijser.org/researchpaper/Solving-TSP-using-Genetic-Algorithm-and-Nearest-Neighbour-Algorithm-and-their-Comparison.pdf was a helpful resource among several others to help me move in this direction. **Also all the tours produced by this algorithm will later be part of initial population in genetic algorithm.**

2. **Savings heuristics** algorithm:
    - Start by selecting a random base vertex. All other vertices would be anchored around this vertex.
    - Remove each pair of vertices from base vertex and establish link between these two vertices only when there is a net saving of distance.
    - Repeat this until (numberOfCities – 2) number of vertices are unlinked from base vertex.

Savings heuristics output also depend on randomly selected base vertex. Therefore similar to earlier nearest neighbor approach, I called this algorithm at various base vertex points. The starting point was chosen by dividing the number of cities into intervals of size two and picking a starting point randomly in each of these intervals. Here interval was little more compared to nearest neighbor since it is little more computationally expensive compared to nearest neighbours. Again **all the tours produced by this algorithm will later be part of initial population in genetic algorithm.**

3. **Greedy heuristics** algorithm:
    - Sort all the edges based on provided distance matrix
    - Add each least cost edge into the tour as long as they don’t form a cycle prematurely

I selected this algorithm since its time complexity is quite less and also its tour would contribute to the initial population of genetic algorithms. Among some resources, https://www.ijeat.org/wp-content/uploads/papers/v4i6/F4173084615.pdf was a notable one. This paper infact prefers greedy heuristic over nearest neighbor and genetic algorithms.

4. Genetic algorithm:

a. Initial population of tours:

   Initially the population size would be (numberOfCities * 5). The tours generated by nearest neighbors, savings heuristics and greedy heuristic would be part of this initial population. The remaining part of population is randomly generated. While doing all of this, it is made sure that there is no repetition of any tour in entire population.

b. Calculate fitness for each tour in the current population:

   The cost of the tour is considered to be the fitness of that member(tour) of population. So unlike general convention wherein fitness is maximized, in my implementation, I would be minimizing the fitness.

c. Select population based on fitness:

   The size of selected population would be half the size of current population. The current population is sorted and (currentPopulationSize/4) best fitness tours would be selected. The remaining part is selected randomly each from an interval of size=(currentPopulationSize/(2*numberOfPopulationSelectedRandomly). The reason behind such a selection is to allow best few set of tours to remain and randomly inject other tours from current population across different range of fitness. This is done hoping that, there are representatives of multiple tour costs as opposed to preserving just best tour representatives solely. **This helps increase the diversity in the selected population.**

d. Cross-over population selected in previous step:

   Here, I have used ordered crossover operator. Each consecutive pair of tours in the sorted order of selected population are crossed over to generate the off-spring population. The fitness scores(tour cost) are injected into each member of offspring population.

e. Merge the off-spring population with current population:

   The size of the new population would remain constant over iterations(that is, size would be same as initial population). Entire off-spring population would be part of the new population. The remaining part is filled with best tours from current population.

f. The new population generated in previous step would act as current population. **Repeat from step c to e until best tour of off-spring population remains same in two consecutive iterations**.

g. When iteration is broken due to no increase in best tour from off-spring(step f), mutate the current population:

   This mutation steps tries to randomly mutate 25% of population randomly(during first half of the times, the mutation was invoked) and 50% of population(during second half of the times, the mutation was invoked). **Mutation tries to increase diversity of population. Also,**

here the extent of mutation also depends on the number of times the population was mutated.

h. Step g(mutation step) will run for steps=numberOfCities. Step g runs only when step f iteration was broken

So, in short the mutation is tried only when there is no increase in off-spring population best tour cost. Also it is tried only a certain number of times before restarting the algorithm with a bigger population.

i. Once mutation cycle reaches maximum in step h, increase the newPopulation size. Size of new population = current population size * 2. With this new population size, go back to step a.

So, essentially we try mutation for certain number of steps. Later we restart entire genetic algorithm with a bigger population size(equal to new population size)

https://www.researchgate.net/publication/258031702_A_Comparative_Study_between_the_Nearest_Neighbor_and_Genetic_Algorithms_A_revisit_to_the_Traveling_Salesman_Problem was a helpful resource for setting few hyper-parameters related to genetic algorithm.

Dataset used for testing was mainly from https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html

I observed that the general trend (atleast for this dataset) was **greedy heuristic > nearest neighbor heuristic > genetic algorithm >= savings heuristics** in terms of optimal solution.

However sometimes genetic algorithm overtook others too in terms of optimality.

Hence, the greedy heuristic is tried first, followed by nearest neighbor, followed by savings heuristic. As a last resort the genetic algorithm is executed considering tours of all the three algorithm in its initial population. The genetic algorithm is executed by increasing the population size in each iteration indefinitely until timeout or memory exceeds.

Further optimizations possible:

Swarm optimization techniques like ant colony optimization, African buffalo optimization would be good to try out.

I also felt christofides algorithm would be a good option.