# AI: Programming Assignment 1

Anirudh S CS17B002 and Mahima Raut CS17B112

Travelling salesman problem

## 1 Abstract

We have explored various ways to approach the famous NP complete problem of Travelling Salesman posed as a part of the course CS6380 : Artificial Intelligence. Given the coordinates of every city as well as a distance matrix indicating the inter-city distances and were asked to solve 2 versions of the problem : Euclidean and Non-Euclidean.

## 2 Problem Description

Travelling Salesman Problem: Given a set of cities and distance between every pair of cities, we have to find the shortest possible route that visits every city exactly once and returns to the starting point. It is basically the lightest closed Hamiltonian path in the graph system of cities and routes.

In the Euclidean version of the given TSP, cities are given as coordinates on a two-dimensional plane and the salesman can move freely from one city to any other. In this form of the problem, distances between cities are calculated using the Euclidean distance formula, thus we can directly use the distance matrix given to us, no additional computation is needed.

Although TSP is usually studied on a Euclidean plane, when obstacles are placed on the routes, the distances are no longer Euclidean, but they still satisfy the metric axioms. This means that we cannot derive actual route-distances between the cities by euclidean distance formula, so we'll have to use the matrix given to us, and coordinates of cities stand independently for us to determine triangle inequality in the future.

## 3 Considered Approaches

For this project, we tried tackling the Euclidean version of the problem first. The two algorithms considered for the same were the savings heuristics and the greedy heuristics. Nearest Neighbour was not considered as we'd seen many cases for which it gave extremely sub-optimal tours in the class. Additionally, since the last best path printed would be only judged, we considered looping

through the algorithms with until the best possible optimal path was achieved. In case of Savings, we would initialise the starting point of the route randomly over subsequent iterations and run the whole algorithm again to capture best possible route. We also considered if the answer given by savings doesn't improve even after N/10 iterations, we'd switch to 2-opt with whatever resulting min tour we get through repeatedly iterating.

# 4 Current State and Implementation

Currently, the code only uses savings heuristic to generate tours.
After generating a tour with the the savings algorithm, we tried performing a series of edge exchanges to improve upon the original tour by using the 2-opt/2-edge-exchange heuristic. Our original approach was to run the savings algorithm repeatedly with randomly initialising the starting vertex to find the starting point that resulted in the best initial tour, but we found that there isn't much improvement in cost by changing base vertex. Due to time constraints, we were unable to make the 2-opt code work, but you can find our implementation of 2-opt function in the comments.

# 5 Further Scope for improvement

To improve on the runtime of the naïve 2-opt algorithm, there is a comparison that exploits the triangle inequality to determine if a swap will reduce distance. This is the method I used to improve the speed of our algorithm. Using the same notation as above, the comparison is that dist(A, C) must be less than dist(B, D) for an exchange to be distance-reducing. Using this comparison, we reduced the number of exchanges needed to achieve the same result. My algorithm stops once the distance stops improving, indicating a locally optimal tour.

There is another method which can further improve the efficiency of the 2-opt algorithm. This method is to build a neighbor list for each city, sorted by increasing distance of the neighbors from the original city. Using this strategy, the algorithm requires even fewer iterations possibly. We made an attempt to use this method and implemented all the code that generates sorted neighbor lists for each city, but we haven't been able to get it to work in practice.

# References


- https://www.slideshare.net/sandpoonia/lecture28-tsp
- https://dm865.github.io/assets/dm865-tsp-ch-handout.pdf
- https://link.springer.com/chapter/10.1007/978-0-387-77131-1_14