Good afternoon everyone, I am Ryan Goh. Today, I would like to share my findings about the Travelling Salesman problem.

Firstly, what I know.

TSP is about finding the shortest possible route that visits each node exactly once and return to the starting point. For example, the left route shows the best possible route, and the right route is a possible unoptimal route.

Now we explore exact algorithms that can fully solve the problem.

We can brute force all N factorial routes. However, that takes O(N\*N!) time, which is only practical for N <= 12.

Secondly, we can use dynamic programming, known as the Held-Karp algorithm. The details are on the slide, with the time complexity being O(N^2\*2^N). This is only practical for N <= 20 at best.

Unfortunately, exact algorithms, as for now, can only be exponential in time complexity, which is too slow practically. What if we can solve the problem less accurately, but much faster? Welcome to heuristic algorithms.

We would explore the following: what are the heuristic algorithms for TSP, and how accurate they are on average?

Firstly, the nearest neighbour algorithm. We greedily visit the next unvisited node closest to us. Time complexity is O(N^2) if we fix a starting point, O(N^3) if we try all starting points, which is good for N <= 500. It is 25% worse than exact solutions though.

Next, shortest edge algorithm. We greedily add the shortest distances between 2 nodes as long as it does not form an invalid route. Time complexity is O(N^2logN), but 20% worse than exact.

We can also try tweaking a valid route to form another shorter route, like the k-opt algorithm. This algorithm repeatedly removes k edges from the original route and rejoin them differently to form a shorter path, until no more moves can be made. For 2-opt, it is 5% worse than exact, 3-opt, 3%; but returns diminish over larger k.

Time complexity is O(cN\*N^2) where c is the number of operations. O(cN\*MN) exists also, but may be less optimal.

From this, I learnt that we may need to sacrifice accuracy to improve time, or vice versa. For example, in my test run with n=400, the O(N^2) nearest neighbour algorithm performed the worst in optimality whereas the 2-opt nearest edge algorithm performed the best, even though the former has the best time complexity. So, the algorithm we choose has to be very dependent on the context of the problem we are solving.

Now, a demonstration.

Here is my source code for the algorithms mentioned. It has a brute force algorithm, dynamic programming, nearest neighbour, shortest edge and 2-opt. Let us run the program. While we wait, I would also like to say that I had written “check answers” to check whether my construction of the path matches the distance found, which is asserted here.

As aspected, the brute force and dynamic programming solution worked optimally, with the other heuristics having a worse solution.

This is the end of my presentation. Thank you and any questions?