ICAI 2023
12th International Conference on Applied Informatics
Eger, Hungary, 2023

Solving and Visualizing the Three-Dimensional Travelling Salesman Problem

Nándor Banyik[a], **Gergely Kovásznai**[b]

[a]Eszterházy Károly Chatolic University
banyiknandor1@gmail.com

[b]Eszterházy Károly Chatolic University
kovasznai.gergely@uni-eszterhazy.hu

Abstract

In our research, we investigated the Travelling Salesman Problem (TSP) in three dimensions. TSP is a well-known problem where you have n cities that you want to traverse, each of them only once, using the cheapest route. In the beginning of our research, we implemented a *generation system* to generate cities of a desired amount in 3D. After that, we implemented two algorithms: the *Nearest Neighbour*, and the *Branch and Bound*. Our system compares the performance of those algorithms visually in 3D.

While the Nearest Neighbour algorithm is quite fast, it does not typically find the cheapest route, thus it does not typically provide an optimal solution. It is a very light and easy algorithm: first of all, it defines a starting point (or node) and moves to its closest accessible neighbouring point, where it performs the same step. During this iteration, the points we have traversed are marked inaccessible, in order not to stuck in an infinite loop. The iteration runs until we cannot access any accessible points. Finally, the algorithm returns to the starting point from the lastly visited point. While this algorithm is light, its main problem is the following feature: while it always moves to the closest neighbour, it may detour from an optimal route, and the generated route will be more expensive. [1]

After analyzing this main problem, we started to develop a new algorithm, which is a modification (or extension) to the Nearest Neighbour algorithm. Our algorithm restricts the Nearest Neighbours' available points. Its concept is to

partition the set of available points into segments based on their coordinates. This is why we named the algorithm *Segmented Nearest Neighbour (SNN)*.

The Segmented Nearest Neighbour Algorithm

For partitioning the points to segments, we need to find a corner point for each segment. Then, we iterate through each non-corner point, find the closest corner, and include the point in the closest corner's segment.

Once we have our segments, we choose a starting point S, which will be one of the corner points. After that, we search for a point L in the same segment to traverse the latest in the algorithm. In three dimensions, there may be height differences between two points, so we need directed edges between them. So for finding point L, we need to find the farthest point in the initial segment.

Then, we find the exit point in the segment which is the closest point to the next segment's closest point. Afterwards, we search for a route inside the segment; thus, we only visit points that are in the same segment. If there is no more accessible point left in the segment, then we connect the lastly visited point to the segment's exit point. After that, we go to the next given segment, and repeat the steps; find the exit point in the current segment, and then search for a route in the segment.

After we have finished with all the segments, we connect the lastly visited point to L, and then we connect L to S.

In our experiments, we tested two segmentation methods; one of them is the *4-segmentation*. In this method, we only find the corners based on their x and z coordinates. We compared the performance of SNN against that of NN, and the results sufficiently proved that SNN was competent, but was in need for improvements.

The second segmentation method we tested is the *8-segmentation*. In this method, we find the corners based on their x, y, z coordinates. We compared SNN against NN yet again, but the results were quite the same. After several tests, we noticed that the order in which we traversed the segments mattered. We tested different orders and realised that each order were advantageous in different benchmarks. So, we collected 10 orders for traversing segments and, for each benchmark, we tested all of them to find the best order. This solution drastically improved the efficiency of our algorithm, as Figure 1 illustrates.

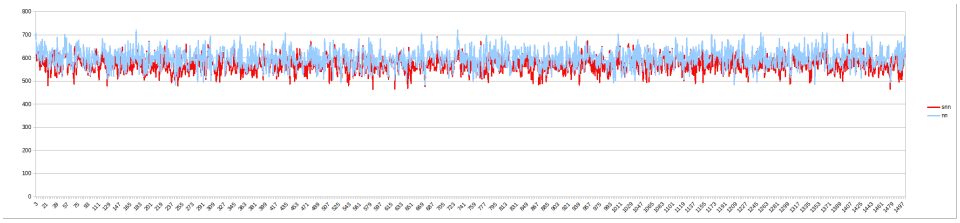

Figure 1. Comparison of the Segmented Nearest Neighbour's and the Nearest Neighbour's route cost.

References

[1] A. Z. Gregory Gutin Anders Yeo: *Traveling Salesman Should not be Greedy: Domination Analysis of Greedy-Type Heuristics for the TSP*, Discrete Applied Mathematics 117 (2002), pp. 81–86, doi: https://doi.org/10.1016/S0166-218X(01)00195-0.