Delft University of Technology

Practical assignment for Transportation, Routing and Scheduling
WI4062TU

Project Report

Authors:
Ivaylo Ganchev (5135001)
Goerge Tzanetos (4667638)

Date:
February 22, 2023

Contents

1 Introduction

Travelling Salesman Problem (TSP) is the challenge of finding the shortest and optimal route for a person to travel through several cities. The definition of such problem is that from a certain number of cities a salesman need to visit each of them exactly once and finish his trip in the initial city, with the order of going through the cities not being important. It is a very common method applied in scheduling deliveries since in this businesses time and cost efficiency is very important. Its wide applicability is contributing a lot for the significance of the algorithm but also the difficulty can rise a lot with the number of cities rising as well. It is considered as an NP-hard problem and there is not any algorithm that is able to solve all the instances of the problem. In order to produce a relatively good solution that can serve its purpose heuristic methods come in practice. There exist different heuristics such as Nearest insertion, Furthest insertion, Cheapest Insertion, etc. depending on the way how we insert the next node into the path. Non of them guarantees that an optimal path will be provided but they guarantee that a good path will be proposed for a short amount of time.

2 Problem formulation and methods for solving it

For solving the Euclidean TSP (ETSP) problem, where distances between cities obey the triangle inequality, we are going to implement nearest neighbour method which will be explained later. In order to compare the results of this algorithm we are going to compare it with the Local Search algorithm in term of finding a sorter path.

2.1 Mathematical representation of TSP

Here the general representation of the TSP is given to give a better insight of the procedure in TSP.

$$\begin{aligned}
\min \quad & \textstyle\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.} \quad & \textstyle\sum_{j=1}^{n} x_{ij} = a_i \quad 1 \le i \le m \\
& \textstyle\sum_{i=1}^{m} x_{ij} = b_j \quad 1 \le j \le n \\
& x_{ij} \ge 0 \quad 1 \le i \le m, 1 \le j \le n \\
& \textstyle\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1 \quad \forall S \subset V, S \ne \emptyset, \bar{S} \ne \emptyset
\end{aligned} \tag{1}$$

Here x_{ij} represent the decision variables which is equal to 1 if a city j follows directly after city and 0 otherwise and c_{ij} represent the cos for including a city. The first two constraints tells us that each node should be visited by exactly one node and is left again exactly once and the last constraint prohibits sub tours.

2.2 Local search LS

In the local search method we start from some initial solution and we start tweaking it in order to make it better. The algorithm by which the local search method works is the following:

- Initialize a solution found by any method (eg. Greedy algorithm)
- Look at the neighbouring solutions and search for a better one
- If a better solution is found it is becoming the current one
- The procedure is repeated until no better solution can be found and the current one is called a local optimum.

As a disadvantage of this algorithm can be noted that if the initial solution is very bad we may need to explore many new solutions in order to find a local optimum. However, we can stop the method at any point since even at the beginning we already have a solution so when we stop the algorithm for sure we will have a better one than that in the in the starting point.Having this in mind we are going to compare the Local Search method with the Nearest Neighbour algorithm and see which of the two will provide a better result in finding the shortest path.

2.3 Nearest Neighbour NS

Assuming that the TSP is symmetric means that the costs of traveling from point A to point B and vice versa are the same. With this property in effect, we can use a heuristic that's uniquely suited for symmetrical instances of the problem. It's known as the nearest neighbor approach, as it attempts to select the next vertex on the route by finding the current position's literal nearest neighbor.

2.3.1 Algorithm

We can explain this method with the following easy steps:

- Randomly select a node as initial;
- Find the next node that is nearest to the previous(has the lowest cost) that is not yet in the route and add it to the new route;
- Repeat until all the nodes are included in the route.

3 Results

This section presents the results of the Euclidean Travelling Salesman problem, using the Nearest Neighbour heuristic, and compares it to the Local Search heuristic. For the latter, an algorithm is already provided as well as the datasets. The algorithm for the Nearest Neighbour heuristic is explained by pseudo-code in subsection 2.3, and is shown in MATLab script in section 5. Below, in figures 1 to 4, the comparison of route plots are visible, and so the higher efficiency of the Nearest Neighbour heuristic can already be distinguished. Specifically, Figure 1 and Figure 2 concern the first provided dataset that contains 50 cities.

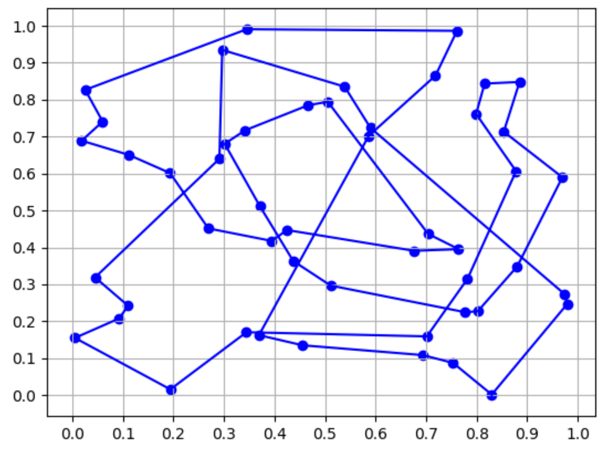

Figure 1: Local Search method for `tsp_50_0.csv`

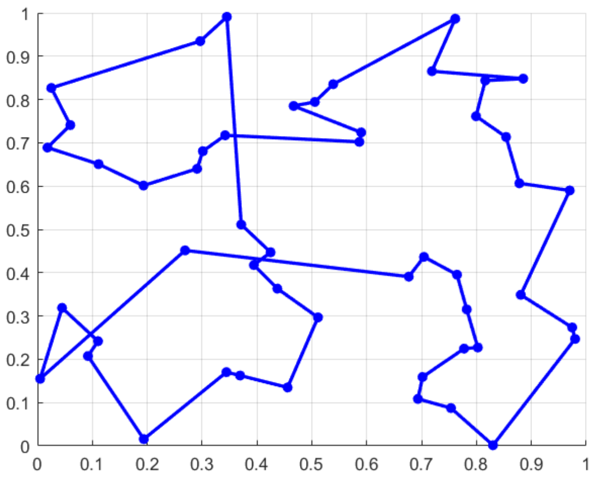

Figure 2: Nearest Neighbour method for `tsp_50_0.csv`

Figure 3, and Figure 4 concern the sixth provided data set that contains 75 cities. These two data sets were previewed as one for each size with the corresponding road maps for the travelling salesman. However, there have been a total of 10 data sets provided. The resulting lengths travelled and running times for all of these are listed in Table 1. As highlighted, the Nearest Neighbour heuristic produces significantly more efficient routes than the Local Search one for all sets and. This becomes even more obvious in Figure 5, where all objectives are plotted for the two different heuristics. The dashed line in the center of the graph also separates the 50-city data sets (left) and the 75-city data sets (right), with the largest improvement in travelled length being present at the sixth database, at 5.8186. Also from the table we can see that the NS is around 3 times faster than the LS while executing the task for finding a route.

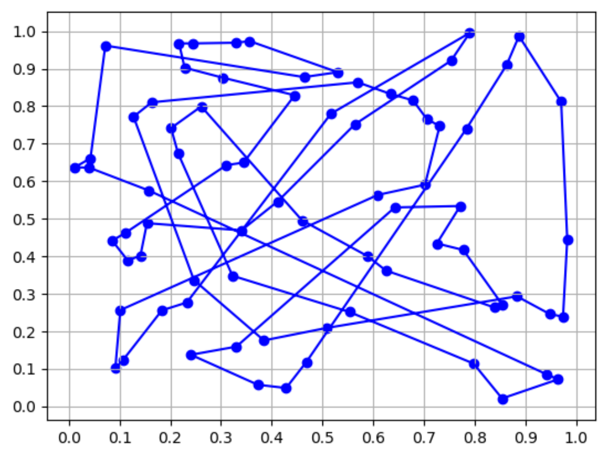

Figure 3: Local Search method for `tsp_75_0.csv`

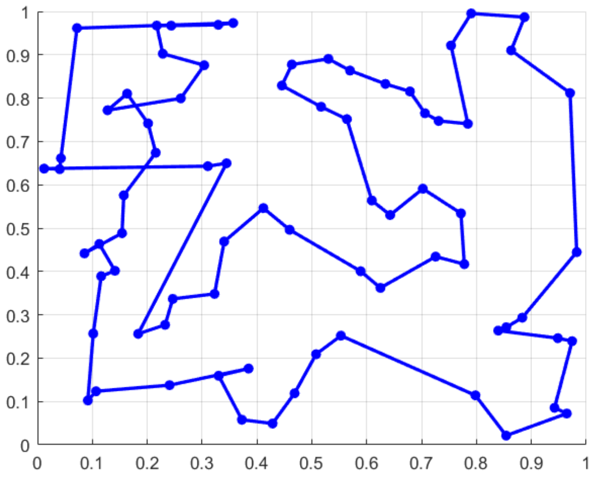

Figure 4: Nearest Insertion method for `tsp_75_0.csv`

Table 1: Objective Results and Runtimes

	Dataset	**LS Objective [-]**	**NN Objective [-]**	**LS RunTime [s]**	**NN Runtime [s]**
1.	`tsp_50_0.csv`	9.2705	6.3738	0.4063	0.05
2.	`tsp_50_1.csv`	8.7469	6.4584	0.375	0.0427
3.	`tsp_50_2.csv`	8.1090	6.8916	0.4219	0.0382
4.	`tsp_50_3.csv`	7.6924	5.8805	0.6875	0.0469
5.	`tsp_50_4.csv`	9.9183	6.1336	0.3218	0.0495
6.	`tsp_75_0.csv`	13.4083	7.5897	0.5469	0.0738
7.	`tsp_75_1.csv`	10.8061	7.8169	0.9531	0.0784
8.	`tsp_75_2.csv`	12.7600	7.9136	0.9688	0.0625
9.	`tsp_75_3.csv`	10.5202	7.3443	0.9688	0.0699
10.	`tsp_75_4.csv`	11.2374	8.0144	0.9375	0.0529

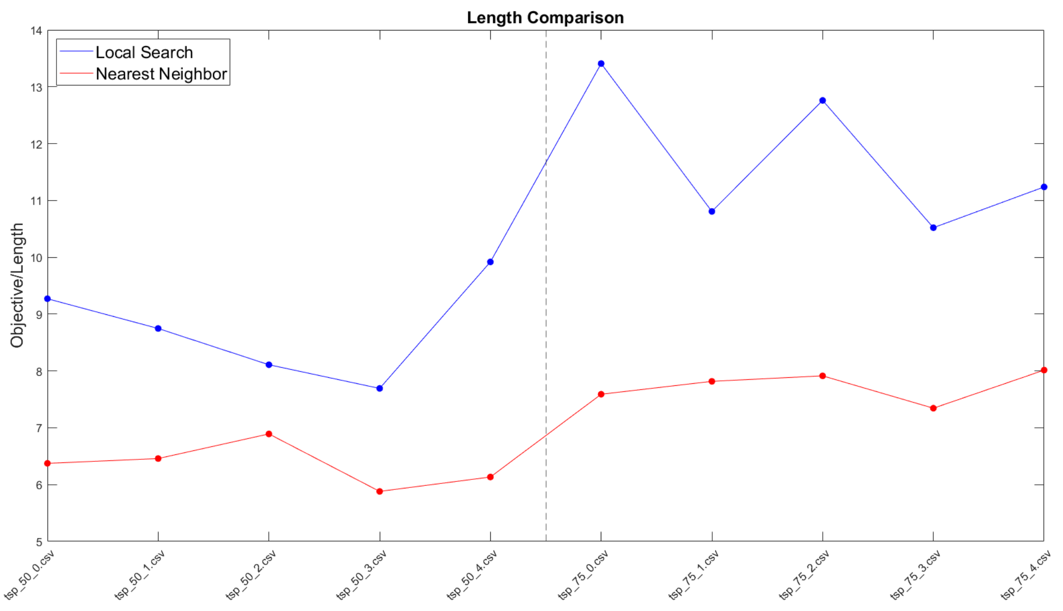

Figure 5: Length Comparison for the two heuristics, with all datasets

4 Conclusion

To conclude with, the different variations of the Travelling Salesman problem, like the Euclidean Travelling Salesman one, are a set of problems that produce a high-level computational task, related not only to vehicle routing, but also other areas such as computer wiring and X-ray crystallography. Clearly, the initial task at hand is to create an algorithm to visit all the cities and return to the start, and secondly to do it more efficiently. There are numerous heuristics to get through the initial one, just like the two that are used in this report. The efficiency task is achieved here, with two different heuristics producing a road map for a specific set of cities, but at significant length differences. The first four figures of section 3, show an obvious improvement in routes, when applying the Nearest Neighbor method. The computational task however, is something the Travelling Salesman problem is known for, as the solution usually ends up requiring supercomputer capabilities when applying complexity such as a set of an actual network of cities in a country. Therefore, it can also be seen here, in Figure 5, that the difference generally becomes larger after adding 25 nodes to the data. As a final note, it is valuable to repeat this project with other heuristic methods that are widely available, and observe the new results, to potentially arrive to an optimal solution.

5 Appendix

```
%%
% Transportation , Routing and Scheduling Assignment
% Prepared by Ivaylo Ganchev and George Tzanetos
%%
clc
clear all
close all
%%
t1 = readtable('tsp_50_0.csv');
T1 = table2array(t1);
t2 = readtable('tsp_50_1.csv');
T2 = table2array(t2);
t3 = readtable('tsp_50_2.csv');
T3 = table2array(t3);
t4 = readtable('tsp_50_3.csv');
T4 = table2array(t4);
t5 = readtable('tsp_50_4.csv');
T5 = table2array(t5);
t6 = readtable('tsp_75_0.csv');
T6 = table2array(t6);
t7 = readtable('tsp_75_1.csv');
T7 = table2array(t7);
t8 = readtable('tsp_75_2.csv');
T8 = table2array(t8);
t9 = readtable('tsp_75_3.csv');
T9 = table2array(t9);
t10 = readtable('tsp_75_4.csv');
T10 = table2array(t10);

%% Nearest Insertion Code
tic

Nodes = T1; %Cities with distances
N_nodes = size(Nodes,1);

d = pdist(Nodes);
d = squareform(d);%Creating square matrix with distances
d(d==0) = realmax;

shortLength = realmax;

for i = 1:N_nodes

    stCity = i; %Initialization

    p = stCity; %Path

    pTr = 0;
    NewDist = d;

    curCity = stCity; % City in the current moment

    for j = 1:N_nodes-1

        [minDist,nextCity] = min(NewDist(:,curCity));
        if (length(nextCity) > 1)
```

```
                nextCity = nextCity(1);
            end

            p(end+1,1) = nextCity;
            pTr = pTr +...
                        d(curCity,nextCity);

            NewDist(curCity,:) = realmax;

            curCity = nextCity;

        end

        p(end+1,1) = stCity;
        pTr = pTr +...
            d(curCity,stCity);

        if (pTr < shortLength)
            shortLength = pTr;
            shortPath = p;
        end

end

toc

figure
x_min = 0;
x_max = 1;
y_min = 0;
y_max = 1;
plot(Nodes(:,1),Nodes(:,2),'bo');
axis([x_min x_max y_min y_max]);
axis equal;
grid on;
hold on;

% plot the shortest path
xd=[];yd=[];
for i = 1:(N_nodes+1)
    xd(i)=Nodes(shortPath(i),1);
    yd(i)=Nodes(shortPath(i),2);
end
line(xd,yd,'color','b','LineWidth',2);
title(['Path length = ',num2str(shortLength)]);
hold off;

%%
% Length Comparison
figure
filenum = 1:10;
obj1 = [9.2705, 8.7469, 8.1090, 7.6924, 9.9183, ...
        13.4083, 10.8061, 12.7600, 10.5202, 11.2374];
obj2 = [6.3738, 6.4584, 6.8916, 5.8805, 6.1336, ...
        7.5897, 7.8169, 7.9136, 7.3443, 8.0144];

plot(filenum,obj1,'b',filenum,obj2,'r')
hold on
```

```
scatter(filenum,obj1,'b','filled')
hold on
scatter(filenum,obj2,'r','filled')
hold on
xline(5.5,'--');
ylabel('Objective/Length','FontSize',15);
xticklabels({'tsp\_50\_0.csv','tsp\_50\_1.csv','tsp\_50\_2.csv', ...
             'tsp\_50\_3.csv','tsp\_50\_4.csv','tsp\_75\_0.csv', ...
             'tsp\_75\_1.csv','tsp\_75\_2.csv','tsp\_75\_3.csv', ...
             'tsp\_75\_4.csv'});
xtickangle(45);
legend({'Local Search','Nearest Neighbor'},'Location','northwest','FontSize',15);
title('Length Comparison','FontSize',15);
```