



KTH ROYAL INSTITUTE OF TECHNOLOGY

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

---

# Project TSP Report

---

December 1, 2018

AUTUMN TERM 2018

NAVID FARHADI

FRANZ FUCHS

ARTURS KURZEMNIEKS

ALEXANDRE PEINOT

KATTIS SUBMISSION ID: 3520421 (SCORE: 35.02778)

# 1 Introduction

Given a set of  $n$  cities and the distances between them, the Travelling Salesman Problem (TSP) consists in finding the shortest circuit visiting all the cities only once. In this project, we study the Euclidian version of TSP in two dimensions. In this variant of the problem, all the cities are represented by their coordinates  $x$  and  $y$  and the distance between two cities is the Euclidian distance between them (rounded to the nearest integer in the case of the project).

This report presents the different algorithm we have implemented for obtaining the initial tour of this problem and for optimizing it. It shows the results obtained and compares the performance of the different solutions.

## 2 Implementation

### 2.1 Initial Tour

#### 2.1.1 Christofides' Algorithm

The algorithm of Christofides is not a brandnew invention. It is an approximation algorithm that achieves to find a tour with worst case length  $ALG \leq 1.5 \cdot OPT$ . The Christofides algorithm consists of major parts that are explained in the following text.

#### Construction of a Minimum Spanning Tree

A Minimum Spanning Tree (MST) is a subset of edges in a graph that produces a fully connected graph with minimum weight. It is important to note that there does not have to be a unique solution for the MST problem. There can be multiple sets of edges, which lead to equal total cost. There are several algorithms to solve that problem. The most common ones need  $O(m \cdot \log(n))$  time. One example is the algorithm of Prim. I decided to implement this one because I had previous experience with that algorithm. The algorithm of Prim has an initial node and creates the tree originating from that node. Once a vertex is processed the shortest path has been found and the incoming edge is added to the set of MST edges. Prim's algorithm is fairly simple to implement in C++ because the algorithm is based on a priority queue. The standard library for C++ offers various structs to use out of the box. I decided to use the `std::make_heap` function in combination with the basic data structure `std::vector<std::pair<int,double>>` and a comparator struct that puts the smallest reachable vertex on the top of the heap.

### Finding of Vertices with Odd Degree

After finding one solution for the MST problem, the algorithm of Christofides says to sort out all the vertices that do not have an odd degree in the MST edges set. This check can be done in  $O(m + n)$ . First we iterate over all MST edges and increase for the two vertices the degree. After that, we sort out all vertices with an even degree. Since the amount of MST edges is always less equal than the amount vertices, we can simplify the runtime of finding all odd vertices to  $O(n)$ . The formula  $E_{MST} \leq n$  holds because of the construction criterion of MST.

#### 2.1.2 Nearest Neighbor

An initial tour can also be found with the nearest neighbor approach. Starting from a random initial vertex, the next vertex is determined in the following way:

- Compute the distance to each vertex that has not been visited yet
- Choose the vertex with the minimum distance

It can be shown that this approach leads to a tour with worst case length  $ALG \leq 2 \cdot OPT$ . My implementation of the nearest neighbor algorithm has a runtime of  $O(n^2)$  because there is an iteration over all vertices and another nested iteration over all vertices to find the one with the minimum distance to the current vertex.

#### 2.1.3 Multiple Fragment

The Multiple Fragment (MF) approach also promises to give a good initial tour, but cannot hold as tight bounds as the algorithm of Christofides. The MF approach is based on calculating all edges and sorting them from the lowest weight to the highest one. Starting with the edge that has the lowest weight edges are picked and put into the solution set of edges if they do not create circle in the solution domain. Moreover, each vertex is only allowed to have one incoming and one outgoing edge. In that way, my implementation of the MF approach has a runtime of  $O(n^3)$  because it is possible to have to iterate through all edges and to check at the last edge a circle of length  $(n - 1)$ .

## 2.2 Optimization

After the obtention of the initial tour, we need to run some optimization algorithm to get as close as possible from the optimal solution. In our project, we have focus

our work on three special cases of k-opt: 2-opt, 2H-opt and 3-opt. All of these algorithms belong to the class of local search algorithms. The main issue with these algorithms is that they can easily fall in some local minimum and then never obtain the optimal path.

In practice, these algorithms must run until there is no new improvement. However, due to the Kattis time constraint, we have set a limit in the time execution in order to get the best possible path in the allowed time.

### **2.2.1 2-opt**

The concept of the 2-opt algorithm is really simple. It basically consists in breaking two different edges of the tour and reconnecting the two obtained parts of the tour in a way that minimizes the distances. In this case, there are only two ways of reconnecting the tour. The complete 2-opt algorithm iterates over all the possible pairs of edges cutting in the tour.

Our implementation of the algorithm is  $O(n^2)$  because it iterates through all pairs of edge cutting. However, it is quite fast as it only evaluates the distances which could change by reversing the path between the two cuts and does the reverse only if it improves the tour's distance.

### **2.2.2 2H-opt**

### **2.2.3 3-opt**

## **3 Evaluation**