# DESIGN & ANALYSIS OF ALGORITHEMS – CSC311
# - PROJECT-
# SOLVING TSP FOR METRIC GRAPHS USING MST HEURISTIC

Fall 2020


Participant Students

Ahmad Ali Al-Mosallam – 438103307

Faisal Abdullah Al-Duwayhi - 438102142

## Table of Contents

# 1. INTRODUCTION

## 1.1 Purpose

The goal of this project is to build a program that solves Travelling salesman problem using optimal solution and compare it with the approximation solution.

## 1.2 The Problem Definition

Travelling salesman problem (also called travelling salesperson problem or TSP) is an NP-hard problem in combinatorial optimization. TSP problem asks a question: given a list of cities and the distances between each pair of cities, what is the shortest route that visits each city and returns to the start city?

## 1.3 Deep Explanation of The Problem

TSP have various solutions, but not all of them get the optimal solution.

**Exact Algorithms:** are algorithms that always solve an optimization problem to optimality.

And we will try to solve the problem Using brute-force approach - it's an Exact Algorithm - that will find the optimal solution, but it takes $\Theta(n!)$, which is impractical even for 20 cities.

**Approximation Algorithms:** are efficient algorithms that find approximate solutions to optimization problems in polynomial time.

Approximation algorithms are faster than the Exact algorithms. following its name, approximation algorithms cannot get the optimal solution always.

And we will try to solve the problem Using Christofide's algorithm which gives at most 1.5 times the optimal.

Christofide's algorithm works as the following:

For making an Eulerian graph, we have to find a minimum spanning tree and combine it with a minimum-weight perfect matching graph from the MST`s odd vertices.

So, now we can find the Eulerian tour since every vertex in the graph has even degree.

Finally, convert the Eulerian tour to TSP using shortcuts by removing the repeated vertices.

## 2. EXPERIMENTS

-The graphs of the experiments have been attached with the project file.

<table>
<tr><th colspan="2">Experiment (1)</th></tr>
<tr><td>Original Graph (TSP Graph)<br>the Cities: 0, 1, 2, 3, 4, 5<br>Edge 0 -> city: 0, city: 1, weight: 906<br>Edge 1 -> city: 0, city: 2, weight: 259<br>Edge 2 -> city: 0, city: 3, weight: 430<br>Edge 3 -> city: 0, city: 4, weight: 156<br>Edge 4 -> city: 0, city: 5, weight: 550<br>Edge 5 -> city: 1, city: 2, weight: 759<br>Edge 6 -> city: 1, city: 3, weight: 531<br>Edge 7 -> city: 1, city: 4, weight: 785<br>Edge 8 -> city: 1, city: 5, weight: 388<br>Edge 9 -> city: 2, city: 3, weight: 420<br>Edge 10 -> city: 2, city: 4, weight: 111<br>Edge 11 -> city: 2, city: 5, weight: 481<br>Edge 12 -> city: 3, city: 4, weight: 371<br>Edge 13 -> city: 3, city: 5, weight: 142<br>Edge 14 -> city: 4, city: 5, weight: 462</td>
<td>Optimal solution Graph<br>Optimal Cost traverse = 1986<br>Optimal cost path = [ 0, 3, 5, 1, 2, 4, 0 ]<br>the Cities: 0, 1, 2, 3, 4, 5<br>Edge 0 -> city: 0, city: 3, weight: 430<br>Edge 1 -> city: 3, city: 5, weight: 142<br>Edge 2 -> city: 5, city: 1, weight: 388<br>Edge 3 -> city: 1, city: 2, weight: 759<br>Edge 4 -> city: 2, city: 4, weight: 111<br>Edge 5 -> city: 4, city: 0, weight: 156</td></tr>
<tr><td colspan="2">Approximation solution Graph<br>Approximation Cost traverse = 2123<br>Approximation cost path = [ 0, 4, 2, 3, 5, 1, 0 ]<br>the Cities: 0, 1, 2, 3, 4, 5<br>Edge 0 -> city: 0, city: 4, weight: 156<br>Edge 1 -> city: 4, city: 2, weight: 111<br>Edge 2 -> city: 2, city: 3, weight: 420<br>Edge 3 -> city: 3, city: 5, weight: 142<br>Edge 4 -> city: 5, city: 1, weight: 388<br>Edge 5 -> city: 1, city: 0, weight: 906</td></tr>
<tr><td colspan="2">Comparison between the optimal and approximation solution: 1.06</td></tr>
</table>

## Experiment (2)

### Original Graph (TSP Graph):

**the Cities:** 0, 1, 2, 3

Edge 0 -> city: 0, city: 1, weight: 668
Edge 1 -> city: 0, city: 2, weight: 764
Edge 2 -> city: 0, city: 3, weight: 933
Edge 3 -> city: 1, city: 2, weight: 528
Edge 4 -> city: 1, city: 3, weight: 282
Edge 5 -> city: 2, city: 3, weight: 729

### Optimal solution Graph:

**Optimal Cost traverse =** 2443
**Optimal cost path =** [ 0, 1, 3, 2, 0 ]
**the Cities:** 0, 1, 2, 3
Edge 0 -> city: 0, city: 1, weight: 668
Edge 1 -> city: 1, city: 3, weight: 282
Edge 2 -> city: 3, city: 2, weight: 729
Edge 3 -> city: 2, city: 0, weight: 764

### Approximation solution Graph:

**Approximation Cost traverse =** 2443
**Approximation cost path =** [ 0, 1, 3, 2, 0 ]
**the Cities:** 0, 1, 2, 3
Edge 0 -> city: 0, city: 1, weight: 668
Edge 1 -> city: 1, city: 3, weight: 282
Edge 2 -> city: 3, city: 2, weight: 729
Edge 3 -> city: 2, city: 0, weight: 764

Comparison between the optimal and approximation solution: $\boxed{\mathbf{1.00}}$

## Experiment (3)

### Original Graph (TSP Graph)

**the Cities:** 0, 1, 2, 3

Edge 0 -> city: 0, city: 1, weight: 1014
Edge 1 -> city: 0, city: 2, weight: 575
Edge 2 -> city: 0, city: 3, weight: 825
Edge 3 -> city: 1, city: 2, weight: 978
Edge 4 -> city: 1, city: 3, weight: 645
Edge 5 -> city: 2, city: 3, weight: 440

### Optimal solution Graph

**Optimal Cost traverse =** 2674
**Optimal cost path =** [ 0, 1, 3, 2, 0 ]
**the Cities:** 0, 1, 2, 3
Edge 0 -> city: 0, city: 1, weight: 1014
Edge 1 -> city: 1, city: 3, weight: 645
Edge 2 -> city: 3, city: 2, weight: 440
Edge 3 -> city: 2, city: 0, weight: 575

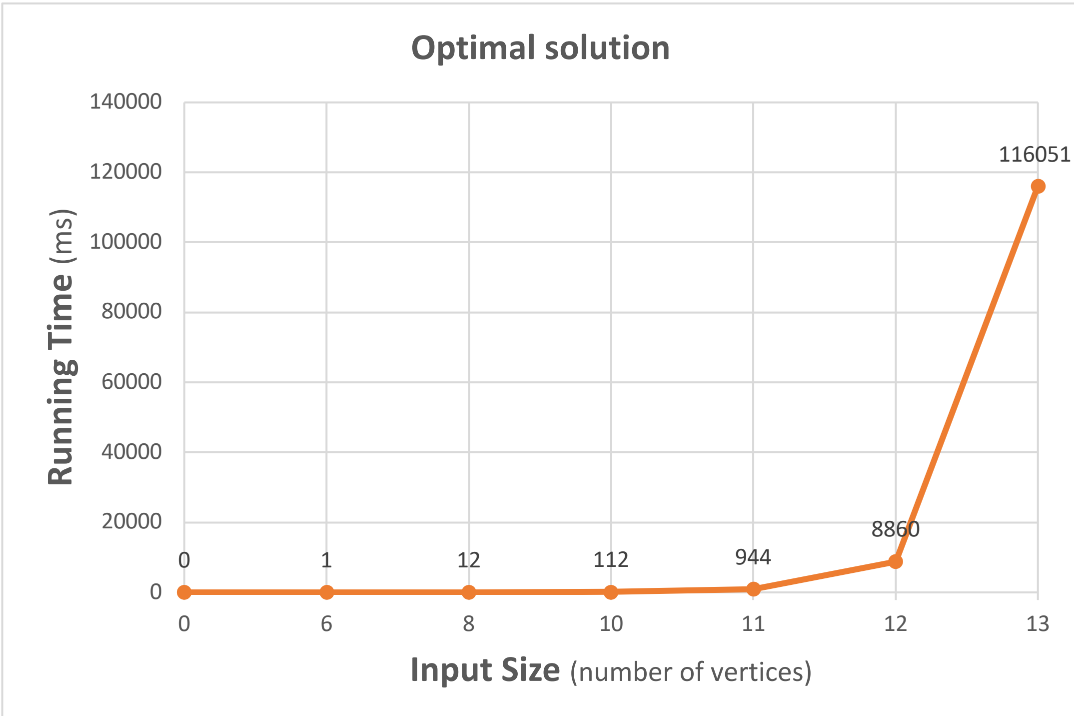
### Approximation solution Graph

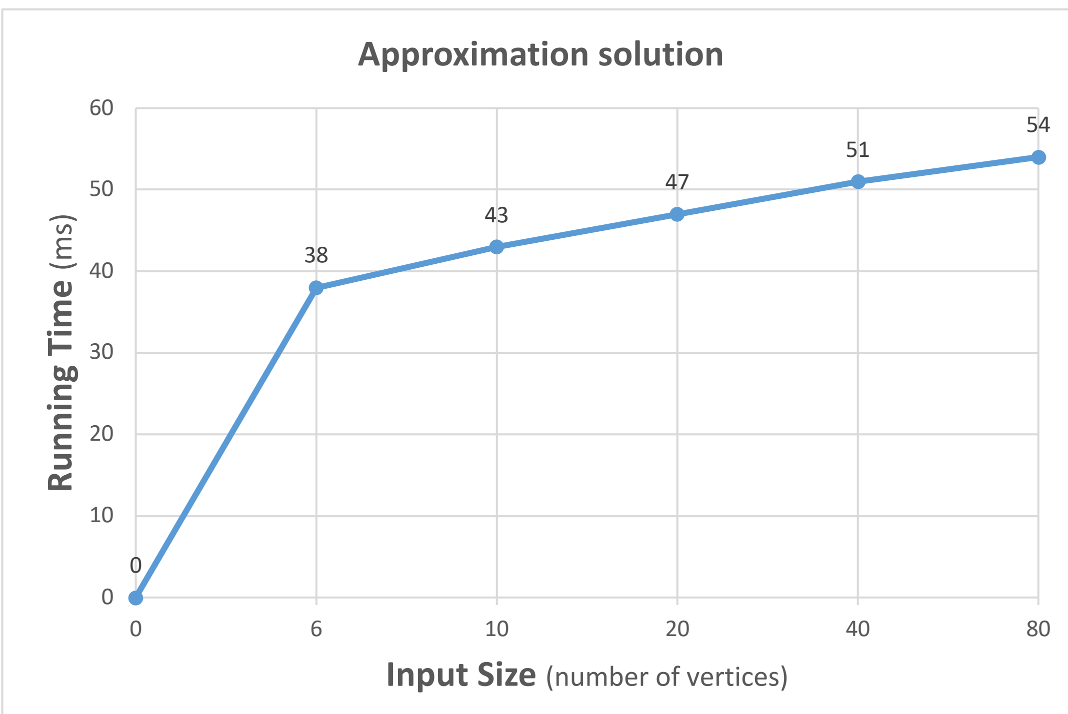
**Approximation Cost traverse =** 2674
**Approximation cost path =** [ 0, 2, 3, 1, 0 ]
**the Cities:** 0, 1, 2, 3
Edge 0 -> city: 0, city: 2, weight: 575
Edge 1 -> city: 2, city: 3, weight: 440
Edge 2 -> city: 3, city: 1, weight: 645
Edge 3 -> city: 1, city: 0, weight: 1014

Comparison between the optimal and approximation solution: $\boxed{\mathbf{1.00}}$

| Experiment (4) | |
|---|---|
| **Original Graph (TSP Graph)**<br>**the Cities:** 0, 1, 2, 3, 4, 5<br>Edge 0 -> city: 0, city: 1, weight: 591<br>Edge 1 -> city: 0, city: 2, weight: 601<br>Edge 2 -> city: 0, city: 3, weight: 367<br>Edge 3 -> city: 0, city: 4, weight: 591<br>Edge 4 -> city: 0, city: 5, weight: 707<br>Edge 5 -> city: 1, city: 2, weight: 10<br>Edge 6 -> city: 1, city: 3, weight: 591<br>Edge 7 -> city: 1, city: 4, weight: 254<br>Edge 8 -> city: 1, city: 5, weight: 260<br>Edge 9 -> city: 2, city: 3, weight: 600<br>Edge 10 -> city: 2, city: 4, weight: 258<br>Edge 11 -> city: 2, city: 5, weight: 260<br>Edge 12 -> city: 3, city: 4, weight: 433<br>Edge 13 -> city: 3, city: 5, weight: 555<br>Edge 14 -> city: 4, city: 5, weight: 124 | **Optimal solution Graph**<br>**Optimal Cost traverse** = 1785<br>**Optimal cost path** = [ 0, 1, 2, 5, 4, 3, 0 ]<br>**the Cities:** 0, 1, 2, 3, 4, 5<br>Edge 0 -> city: 0, city: 1, weight: 591<br>Edge 1 -> city: 1, city: 2, weight: 10<br>Edge 2 -> city: 2, city: 5, weight: 260<br>Edge 3 -> city: 5, city: 4, weight: 124<br>Edge 4 -> city: 4, city: 3, weight: 433<br>Edge 5 -> city: 3, city: 0, weight: 367 |
| **Approximation solution Graph**<br>**Approximation Cost traverse** = 1795<br>**Approximation cost path** = [ 0, 3, 4, 5, 1, 2, 0 ]<br>**the Cities:** 0, 1, 2, 3, 4, 5<br>Edge 0 -> city: 0, city: 3, weight: 367<br>Edge 1 -> city: 3, city: 4, weight: 433<br>Edge 2 -> city: 4, city: 5, weight: 124<br>Edge 3 -> city: 5, city: 1, weight: 260<br>Edge 4 -> city: 1, city: 2, weight: 10<br>Edge 5 -> city: 2, city: 0, weight: 601 | |
| Comparison between the optimal and approximation solution: **1.005** | |

## 2.1 Running time versus input size



## 3. Conclusion

Noticing that the approximation algorithm (Christofide's algorithm) does not have an exact pattern for the approximation solution, and that depends on such things like the minimum spanning tree of the graph and the Eulerian tour ….

However, using the brute-force approach (optimal solution) is not the best choice though, because when we see the chart above, we found out that the optimal solution is growing so fast almost exponentially (the difference in time between input size 11 and 12 is so big), but when we see the chart of the approximation solution, we notice that the approximation solution is faster than the optimal solution even for large data and its time grows slower than the optimal solution.

Finally, we can figure out that the approximation algorithm is a good choice but it's also a double-edged sword algorithm, since it gives a faster running time but not always giving the correct solution.