University of Bergen
Department of informatics

Routing of Offshore Survey Vessels

Author: Ingrid Næss Johansen
Supervisor: Martin Vatshelle

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

October, 2021

Acknowledgements

The work done in this master theses was done at the Department of Informatics at the University of Bergen, during the period of January 2020 to October 2021.

First, I would like to thank my supervisor Martin Vatshelle. I don't think I can ever thank you enough for everything you have done for me, and I don't think I could ever have done this without your help. Even with how the situation have been the last year you have always been there to help me whenever I needed it.

I would also want to thank my boyfriend Oskar Leirvåg for always being there for me and supporting me.

A huge thanks also needs to be given to all my friends, I can't name you all but you know who you are. Ragnhild Skretting Solberg for being with me since 1st grade. You are always there when I need someone to talk too, even though you now live over 400km away. Roger Wisnes for countless academic discussion both at the university and on Zoom, and Sindre Hole for always taking time of your work to join when i needed some time off.

I would also like to thank my Family. My mom and dad for always being there for me and helping me whenever I need it, and always believing that I have what it takes, even though I don't always believe that myself. My siblings for always supporting me and being there when I needed it. And lastly my grandparents. Even though you no longer are with me today I still want to thank you for everything you have done for me, and I know you would have been proud if you could have been here today.

Contents

1 Introduction

The study of vehicle routing is a field of its own, ranging from the vehicles driving on the road to drones flying in the air [Roberti and Ruthmair, 2021]. A lot of the work focus on routing vehicles along roads and not so much focus has been done on routing offshore vessels. Some other master theses that have been working on similar problems are "Simulation Model for Strategical Fleet Sizing and Operational Planning in Offshore Supply Vessels Operations" [Aneichyk, 2009], "Routing and Scheduling of Platform Supply Vessels" [Øyra Friedberg and Uglane, 2013], "Optimal offshore supply vessel planning: A case study of a Chinese offshore oil and gas production area" [Zeng, 2014].

Offshore vessels in the oil industry sometimes deploy equipment on various points of the seafloor, (see Figure 1) and need to plan the best order to deploy on all these locations. In "Monitoring the Ormen Lange field" [Vatshelle et al., 2017] they show how they perform scientific measurements over the oil field called "Ormen Lange". They have a set of measurement locations at the bottom of the sea where they monitor pressure and gravity. The normal mode of operation is that the vessel is positioned over the point of deployment, then the equipment is lowered down to and places on the seafloor using an ROV. Then for a period of time instruments are measuring before they are picked up and returned to the vessel which then proceeds to next location and repeats the process. To transport equipment between measurements locations is time consuming and expensive, mainly due to the cost of hiring the vessels, so finding an optimal path to lower the time it takes to make all the measurements is beneficial.

The survey vessel routing problem takes as input a complete graph generated from a set of points by computing a travel time between each pair of points. We assume the travel time is the same in each direction, hence the graph is undirected. We call this graph class *suvey vessel graphs*. When the travel time between two points is being calculated, acceleration and marching speed must also be taken into account. There are also measurements that takes place at platforms at the bottom of the seabed. The time it takes to lower and raise the equipment down to these platforms can vary, therefore I have decided to add this time as a constant at the end of the algorithm.

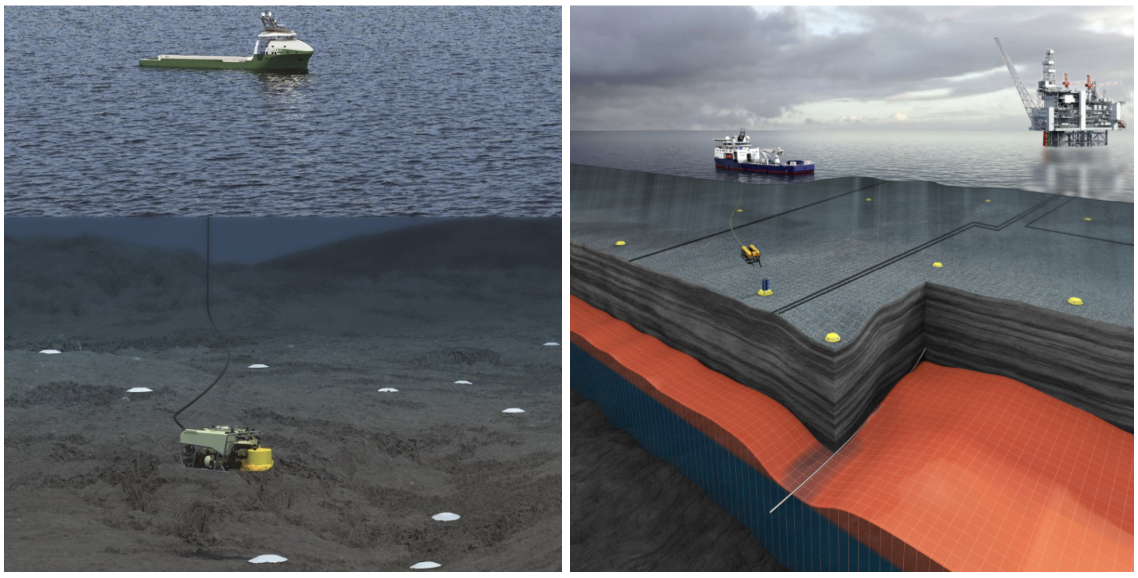
Figure 1: Illustrated Images of survey vessles and the platforms at the sea floor. Taken from [Octio, 2021]

When implementing the algorithm for this problem however there are things you can do to simplify the problem. The first we notice is that the instruments have to be raised and lowered at each of the points on the seabed. This is a process that takes the same amount of time, no matter where in the route it is done. Therefore I have decided to add all of these times together into one big number, and add this number to the result of the algorithm at the end. This way I do not have to think about this problem in the implementation of the algorithm and it still produces an accurate travel time. The acceleration for the vehicle also needs to be calculated into the travel time. Since the time it takes for the boat to reach its marching speed is constant, this can be used, together with the distance between two points to find the duration it takes to travel from one position to another.

The algorithm used in this thesis is called Christofides algorithm. This is an approximation algorithm that finds a good, but not exact solution for the Travelling salesman problem. The solution given by The Christofides algorithm will never be more than $\frac{3}{2} \cdot OPT_{TSP}$. I have also added two improvements to the algorithm. The first is to remove all crossing edges in the final path. This improvement will always make a shorter path for an Euclidean graph, but not for the graph used in this thesis, which is an Metric graph. The second improvement is to move a single vertex in the path, to see if this results in a shorter path.

In the first chapter you find information about the problem and the theme for this thesis. In the second chapter I have made a list of all the preliminaries used in this thesis. In the third chapter the algorithm that is implemented in this thesis is described. The fourth chapter is the biggest part of this theses, the implementation of the algorithm, which is publicly available on GitHub [Johansen, 2021] In the last chapter you will find the results from running the implementation of the algorithm, implemented in chapter four, using the data from the "Ormen Lange" field.

2 Preliminaries

Definition 2.1 (Graph)**.** *A Graph $G(V, E)$ is a data structure, where V is a set of vertices and E is the edges. Edges are pairs of distinct vertices in V. A graph is called a weighted graph if each edge $e \in E$ is associated with a weight $w(e) >= 0$ (in this thesis we only consider positive weights).*

Definition 2.2 (Degree)**.** *The degree of a vertex v is the number of edges connecting to v.*

Definition 2.3 (Weight $w()$)**.** *Given a graph $G(V, E)$, the weight $w(e)$ is the length of an edge $e \in E$*

Definition 2.4 (Weight $Sum_w()$)**.** *Given a graph $G(V, E)$, the weight $sum_w(S)$ is the length of all the edges in $S \subseteq E$ added together.*

Definition 2.5 (Walk)**.** *Given a graph $G(V, E)$, a walk W is a sequence of vertices $v \in V$, connected by edges $e \in E$*

Definition 2.6 (Trail)**.** *Given a graph $G(V, E)$, a trail T is a walk where every edge $e \in T$ is unique.*

Definition 2.7 (Path)**.** *Given a graph $G(V, E)$, a path P is a trail T where every vertex $v \in P$ is unique.*

Definition 2.8 (Cycle)**.** *Given a graph $G(V, E)$, a Cycle C is a walk where the first and the last vertex is the same vertex, and the set of vertices $v \in C \backslash v_{start}$ is unique.*

Definition 2.9 (Minimum spanning tree)**.** *Given a graph $G(V, E)$, a minimum spanning tree (MST) is a set of edges $S \subseteq E$ such that:*

1. *S connects V which means that for all pairs of vertices $u, v \in V$ there must exist a path from u to v only using edges in S.*
2. *The $sum_w(S)$ is minimum.*

Lemma 2.10. *Let $G(V, E)$ be a graph and S a MST of G then there is no cycle using only edges in S.*

Proof. The criteria for a MST is that it is connected and that it is minimum. Assume for contradiction that S contains a cycle C. If you remove one edge $x \in C$ the set $S \setminus x$ will still connect V, and $sum_w(S \setminus x)$ will be less than $sum_w(S)$ (Note that definition 2.1 assumes all weights are positive). This contradicts that S is a MST since $sum_w(S)$ is not minimum. □

Lemma 2.11. *Let $G(V, E)$ be a graph and S a MST of G then $|S| = |V| - 1$.*

Proof. Start by assuming every vertex $v \in V$ is a separate component. Adding one edge between two vertices $v \in V$ will connect them, making them the same element. By doing this $|V| - 1$ times, you end up with one element containing every vertex $v \in V$. Since all the vertices in V are connected with $|V| - 1$ edges, by adding one more edge you make a cycle. This means that a MST $S \subseteq V$ containing more than $|V| - 1$ edges will not be an MST, and an MST $S \subseteq V$ containing less than $|V| - 1$ edges will not be fully connected, there fore not be an MST. This means that every MST $S \subseteq V$ have $|S| = |V| - 1$ edges.

□

Definition 2.12 (Matching)**.** *Given a graph $G(V, E)$, a matching M is a set of edges $M \subseteq E$ such that:*

1. *Every vertex $v \in V$ is part of at most one edge $e \in M$*
2. *(perfect matching) Given a $G(V, E)$, a perfect matching of all vertices in G, is a matching M such that $|M| = \frac{|V|}{2}$*
3. *(Minimum matching) a minimum matching is a perfect matching M such that $Sum_w(M)$ is minimum*

Definition 2.13 (Eulerian cycle). *Given a graph $G(V, E)$, a Eulerian cycle EC is a path $P \subseteq E$ such that:*

1. *P contains every edge $e \in E$*
2. *P starts and end with the same vertex $v \in V$*

Definition 2.14 (induced subgraph). *Given a graph G the induced sub graph $G[S]$ is a graph such that the set of vertices vertices $v \in G[S]$ is a subgraph of G, and the edges $e \in G[S]$ is every edge from the original graph G where both endpoints of the edge e is in S*

3 TSP/Hamiltonian cycle

The Traveling Salesman Problem (TSP) is a well-known problem where the objective is to find the shortest walk through all vertices of a weighted graph. The TSP problem has been studied since 1930 [Zambito, 2006] The mathematician Merrill M. Flood used a version of TSP to solve a school buss rout buss problem.

Definition 3.1 (Traveling Salesman Problem)**.** *The Traveling Salesman Problem (TSP) is a well-known problem where the objective is to find the shortest walk through all vertices of a weighted graph, ending at the starting vertex.*

Definition 3.2 (Hamiltonian cycle)**.** *Given a graph $G(V, E)$, a Hamiltonian cycle H is a cycle that goes through every vertex $v \subseteq V$ exactly once.*

For the traveling Salesman problem no polynomial time algorithm for finding the optimal solution is known. This takes us to the P vs NP problem. P is every problem that there exist an algorithm for in polynomial time. NP is every problem where there exist a certificate for the solution which can be checked in polynomial time. All of the problems in P exist in this bigger group of problems called NP. The problems in $NP \setminus P$ are problems that does not have a polynomial time algorithm.

Inside NP there is also a group of problems called $NP - Complete$. The special thing about these problems in $NP - Complete$ is that they can all be reduced to every other problem in $NP - Complete$. This means that if you found an algorithm to solve one of these problems in polynomial time, you would also be able to solve every other problem in $NP - Complete$. The problems in NP that are not in either P or $NP - Complete$ are called $NP - Intermediate$ problems.

There also exist another group of problems called $NP - Hard$ problems. These are problems outside of NP that do not have a polynomial time solution, and a solution for any of these problem cannot be checked in polynomial time either. For these problems there does not exist a reduction to make them similar to any problem in $NP - complete$. However, if you find a polynomial time solution for any of the problems in $NP - Hard$, that means you can solve every problem in $NP - complete$ in polynomial time also.

3.1 Earlier work

The Traveling Salesman Problem is a problem that has been studied for a long time, but no exact solution has been found to the problem. The problem was posted by Hassler Whitney in 1934 at Princeton University during a Seminar talk, where he talked about the "48 States Problem" which later got the name "Traveling Salesman problem". The person who first popularized the Traveling Salesman Problem was the American mathematician Merrill M. Flood, who in 1956 he published his work "The Traveling Salesman Problem"[Flood, 1956].

One article for the traveling salesman that was posted in 1963 called "An Algorithm for the Traveling Salesman Problem" [Little et al., 1963], used an algorithm called "branch and bound". It breaks the set of all possible tours into smaller sets, using a method called branching. This algorithm calculates lower bound for each subset, to find a tour that is less than every lower bound. This method is based on ideas that are frequently used in assignment problem solving. Another approach to the problem was posted a few years later by non-mathematicians. This was in an article called "An Engineering Approach to the Traveling Salesman Problem" [Roberts and Flores, 1966], posted in 1966. In this article they talked about a more intuitive way with a list of operations for the algorithm to execute. This method generates all circuits from the smallest to the largest, and has been tested on sets of 20, 48 and 52 vertices.

There has also been a lot of work on different versions of the TSP. "The Black and White Traveling Salesman Problem" [Ghiani et al., 2006] posted in 2006, is a version where the vertices in G are partitioned into black and white, and the goal is to design a shortest Hamiltonian tour on G. The method used in this algorithm has been applied in both telecommunications and airline scheduling, and testing of the algorithm shows that it can solve exact instances of up to 100 vertices. A dynamic problem has also been applied in "A Dynamic Traveling Salesman Problem with Stochastic Arc Costs" [Toriello et al., 2014].

One that might also be interesting is "Exact Methods for the Traveling Salesman Problem with Drone" [Roberti and Ruthmair, 2021], where a delivery truck and a drone works together on deliveries. Here they use the benefit of both, the delivery truck can drive long distances, and the drone can drive fast in urban areas since the roads is not an obstacle.

3.2 Metric TSP

Definition 3.3 (Metric Graph). *A graph $G = (V, E)$ is a metric graph if the triangle inequality holds for all triples a, b, c of vertices. That means $w((a, b)) + w((b, c)) \geq w((a, c))$.*

Metric TSP is TSP where the input is a Metric graph. The survey vessel graphs is a subset of Metric graphs. Hence any algorithm for metric TSP can be used to solve the survey vessel routing problem. Another graph class which is a subset of metric graphs is Euclidean graphs.

Definition 3.4 (Euclidean Graph). *A graph $G = (V, E)$ is a Euclidean graph if each vertex can be associated with a point in the plane, and the edges are assigned weights equal to the Euclidean distance between the two endpoints.*

Euclidean TSP is TSP where the input is a Euclidean graph.

Definition 3.5 (Survey vessel graph). *A graph $G = (V, E)$ is a survey vessel graph if each vertex can be associated with a points in the plane, and the edges are assigned weights equal to the time it takes to travel between the two points, assuming constant acceleration a and maximum speed v. The time is computed from the Euclidean distance between the end points by following the formula:*

$$t(d) = \begin{cases} \frac{d}{v} + \frac{v}{a}, & \text{if } d \geq \frac{v^2}{a} \\ 2\sqrt{\frac{d}{a}}, & \text{if } d < \frac{v^2}{a} \end{cases}$$

Lemma 3.6. *A survey vessel graph is not a Euclidean graph.*

Proof. Given a graph $G = (V, E)$ and tree vertices $v_1, v_2, v_3 \in V$. Start by assuming there is a vertex v_c in the center of v_1, v_2, v_3, and being the same distance from v_1, v_2, v_3. This set of vertices is possible to draw in the plane if the graph is a Euclidean graph, but not if the graph is a survey vessel graph. Given the $acceleration = a$, $distance = d$, and $marching\ speed = v$. If the distance from v_1 to v_c is shorter than $\frac{v^2}{a}$ the vessel will not reach the march speed. When looking at the edges (v_1, v_c) and (v_1, v_2) in the Euclidean graph and the survey vessel graph, the difference between these two edges will not be equal. This means that when trying to draw the survey vessel graph in the plane, the circle drawn with radius $w((v_1, v_c))$ around v_1 will not intersect whit the point v_c in the center of the tree vertices v_1, v_2, v_3. This shows that the survey vessel graph can not be drawn in the plane. □

3.2.1 Crossing edges

One of the ways you can improve the length of a walk in an Euclidean graph is by removing crossing edges. By crossing edges we mean two edges that intersect when lines between their endpoints are drawn in the plane.

Lemma 3.7. *Given a Euclidean graph $G = (V, E)$ and a walk $W = \{v_1, v_2, \ldots, v_n\}$, two crossing edges $e_1 = (v_i, v_{i+1})$ and $e_2 = (v_j, v_{j+1})$, where $i < j$. Replacing these two edges by $e_3 = (v_i, v_j)$ and $e_4 = (v_{i+1}, v_{j+1})$ will always give a shorter walk.*

Proof. Given a Euclidean graph $G = (V, E)$, and the two crossing edges $e_1 = (v_1, v_2), e_2 = (v_3, v_4) \in E$. Let v_c be the point where these two edges are crossing. In an Euclidean graph, the direct edge from v_1 to v_3 will always be shorter than the edge from v_1 to v_c + the edge from v_c to v_3, unless the point e is located on the edge from v_1 to v_3. □

For a survey vessel graph however, this does not hold. Given a survey vessel graph G and two crossing edges edges $e_1 = (v_i, v_{i+1}) \in G$ and $e_2 = (v_j, v_{j+1}) \in G$. If the length of the edge e_1 is shorter than $\frac{v^2}{a}$(the distance it takes for the boat to accelerate up to match speed and down again) and e_2 is considerably longer, replacing these edges with two new edges both longer than $\frac{v^2}{a}$ means less time spent in marching speed.

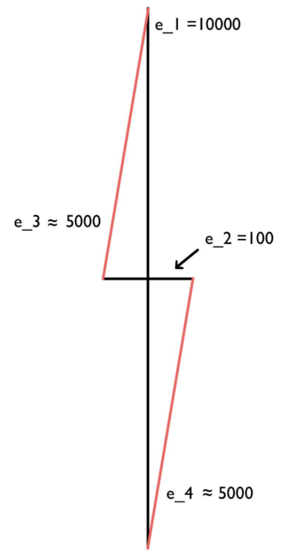

Figure 2: This example shows a case with two crossing edges in a path, and two alternate edges that doesn't cross.

Given the example in figure 2, and given acceleration $= 0.05m/s$, march speed $= 5m/s$, and the distances in the figure given in m . Calculating the travel times you get $e_1 = \frac{10000}{5} + \frac{5}{0.05} = 2100s$, $e_2 = 2 * \sqrt{\frac{100}{0.05}} = 90s$, and $e_3 = e_4 = \frac{5000}{5} + \frac{5}{0.05} = 1100s$. Adding the two crossing edges together you get $2100s + 90s = 2190s$. Adding the two none crossing edges together you get 1100*2 = 2200. As you can see, the travel time for the crossing edges is shorter than the one for the none crossing edges.

3.3 Blossoming algorithm

The Blossoming Algorithm is an algorithm for finding a perfect maximal matching for a set of vertices in any graph. This algorithm is an improvement of the Hungarian matching algorithm that finds maximum weight matching in bipartite graphs. The idea behind the algorithm is that you take cycles of an odd length and combine them to one single vertex. By repeating this until all the odd cycles in the graph has been removed and then run the Hungarian matching algorithm on the new graph made.

The input of the algorithm is a list of every edge in the graph, given by a list of n_1, n_2 and the length between the two vertices. Since the algorithm uses the length of each edge and not the points in the plane, the algorithm can be used by graphs that are not Euclidean. Since the blossoming algorithm outputs a maximum matching and the Christofides algorithm needs a minimum matching, some modifications were made. Since I did not implement the Blossoming algorithm I decided to modify the input to the blossoming algorithm rather than the algorithm itself. What was done was every length between two vertices was multiplied with -100. This Inverts the graph by switching the longest and the shortest distances. This makes the algorithm believe that it returns a maximum matching on the inverted graph, but in reality it returns a minimum matching on the original graph.

3.4 Exact Algorithms

The most naive way to find the optimal/best solution is to start in one vertex and try every single possible route through the graph, and then back to the start. Since you are going to go back to the start it does not matter where you start, and you only have to start from one vertex. This means from vertex v_1, try ever $v-1$ connecting vertex except where you came from. Then from all on these $v-1$ vertices, try every $v-2$ connecting vertices, except the ones where you already have been. Continue this until you have tried every single option.

For the graph I am using which contains n = 115, the number of different walks you can go is $(v-1)*(v-2)*...*(v-(v-1)) = 114*113*...*1 = 114! = 2.5*10^{186}$. If you try to test all of these options on a super computer and started when the universe was made, you would not even have tested 0.000000000000000001 percent of all the possible paths for the graph. The amount of time it takes to test the graph is growing exponentially with the number of vertices in the graph. Which means for 10 vertices it takes less than a second to test all possible walks, for 21 vertices it takes a year, for 28 vertices it takes about the life of the universe.

Another way to find the exact shortest path is by using dynamic programming. For every $S_i \subseteq S$ of length i, you save the shortest path from $v_1 \in s$ to $v_2 \subseteq s$, where $v_1 \neq v_2$. To find for length $j = i+1$, choose a start vertex s_1, then choose two vertices s_2, s_3 to be the last vertices, in the respective order. To find the order of the last $j-3$ vertices, set s_1 to be the start vertex, s_2 to be the end vertex, and find the shortest path between these two vertices. This means $j-3+2 = j-1 = i$ vertices. Since you already have the length of every set of length I, just set the length to be the length of the set between s_1 and s_2 of size I, plus the length between s_2 and s_3. Do this for every start vertex $s_1 \in S$ and every pair of vertices $s_2, s_3 \subseteq S$

3.5 Christofides algorithm

The Christofides algorithm is an approximation algorithm for the Travelling Salesman Problem, that guaranties that the factor of the solution will be within 3/2 of the optimal solution. The graph used in this algorithm is a Euclidean graph which means that the points given to the vertices are there actual points in the plane, and the length between two vertices are the actual length in that plain.

The Christofides algorithm is divided into different algorithms that combined makes an Approximation Algorithm for the Travelling Salesman problem. The Algorithm 1 shows each of the steps for the Approximation Algorithm Christofides algorithm. This is a bit simplified but these are the most crucial steps.

The algorithm starts by making a minimum spanning tree of all the vertices in the graph. After this it goes through the minimum spanning tree and looks at the degree of the vertices and chooses all the vertices with an odd degree. It than makes a perfect minimal matching with all the vertices with an odd degree. The algorithm than combines the minimum spanning tree and the perfect minimal matching to make an Eulerian circuit through every edge in these two combined, see figure 3. Since the Eulerian circuit most likely will contain duplicates of some vertices the algorithm than makes a Hamiltonian cycle by replacing all duplicates of edges with short cuts that skips the vertex.

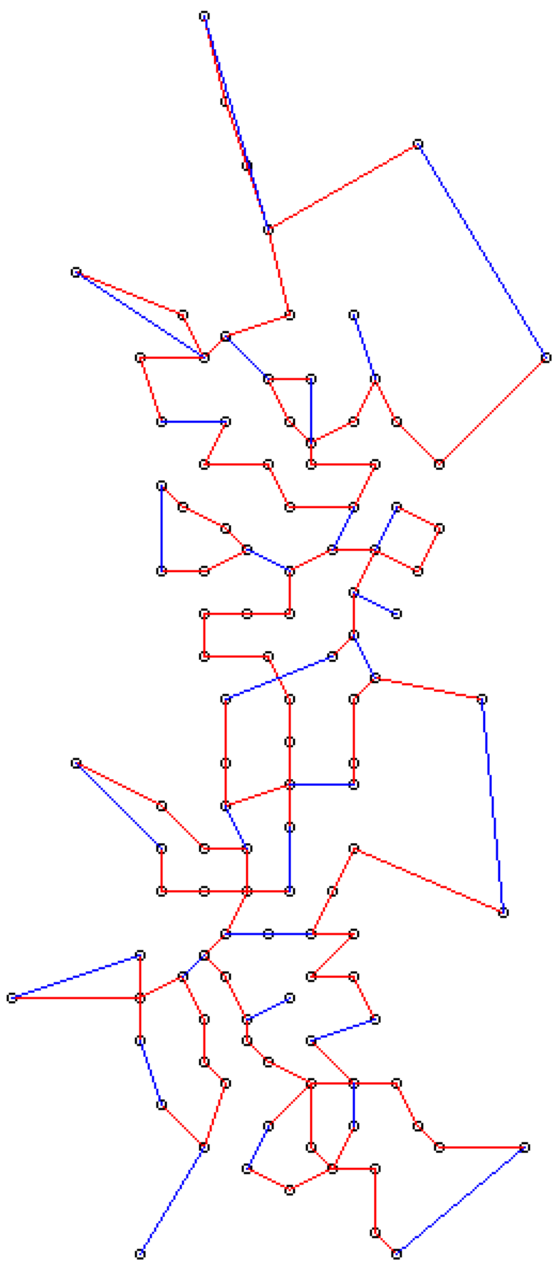

Figure 3: The Eulerian Cycle made from the Perfect matching(the blue lines), and the minimum spanning tree(the red lines). Note that in this example, two edges can have the same endpoints, but only one of the edges gets drawn.

Algorithm 1: Christofides algorithm

```
Data: A graph $G$.
Result: A Hamiltonian cycle
1 $T \leftarrow$ a minimum spanning tree of $G$.
2 $S \leftarrow$ the vertices of $T$ with odd degree.
3 $M \leftarrow$ a perfect matching in $G[S]$.
4 $C \leftarrow$ an Eulerian Circuit containing the edges $T \cup M$.
5 $H \leftarrow$ a Hamiltonian cycle from $C$ by skipping repeated vertices.
6 return $H$;
```

I have also added a few improvements to the algorithm to hopefully get a better result. These added improvements will not grantee a better result, but the walk will be less than or equal to the solution given by the Christofides algorithm.

3.5.1 Correctness

Lemma 3.8. *The algorithm will be able to make an MST T*

Proof. Since the graph G is a connected graph, in fact the graph G is a complete graph, this means that that the Kruskal algorithm used in Christofides algorithm will always be able to produce a MST from G. □

Lemma 3.9. *Given a MST T there will always be an even number of vertices with an odd degree.*

Proof. A graph on V has $1/2 \sum_{v \in V} d(v)$ edges, so $\sum d(v)$ is an even number [Diestel, 2017]. In order for $\sum d(v)$ to be an even number, the number of vertices with an odd degree must be an even number. □

Lemma 3.10. *The algorithm will always be able to make a perfect minimal matching from the vertices with odd degree in the MST*

Proof. Given a graph G and an MST T on the graph, the $|S|$ is even, and that the graph is a complete graph, the Blossoming algorithm used in Christofides algorithm will be able to make a perfect matching. □

Lemma 3.11. *The algorithm will always be able to make an Eulerian Cycle, using the MST and the Perfect matching*

Proof. Given Lemma 3.9 and Lemma 3.10, all the vertices $v \in T \cup M$ has an even degree. All the vertices in $v \in T \cup M$ is also connected, given that the graph is an Euclidean graph. This means that by starting at an arbitrary vertex $v \in T \cup M$, you can go through every vertex $v \in T \cup M$ and end where you started, thus making an Eulerian Cycle. □

Lemma 3.12. *The Algorithm 2 always produces a Hamiltonian cycle.*

Proof. Since the graph G is a complete graph, you can always remove duplicates of noes. If you have two edge $e_1 = (a, b)$ and $e_2 = (b, c)$, you can replace these two edges with a new edge $e_3 = (a, c)$ □

3.5.2 Approximation

In this section we will prove that Christofides algorithm always produces a solution which is at most 3/2 of the optimal solution. We define $OPT_{tsp}(G)$ to be the minimum weight of a Hamiltonian cycle in G. Likewise we define $OPT_{mst}(G)$ to be the weight of a minimum spanning tree and $OPT_{match}(G)$ to be the weight of a minimum perfect matching.

Lemma 3.13. *Let G be a metric graph then $OPT_{tsp}(G) \leq Sum_w(H) \leq Sum_w(C)$.*

Proof. Given a complete metric graph G, $OPT_{tsp}(G) \leq Sum_w(H) \leq Sum_w(C)$. When you have $Sum_w(C)$, $Sum_w(H)$ will always be shorter or at least as short as $Sum_w(C)$ since any short cut you make between two vertices will always make $Sum_w(H)$ shorter, and you will never need to add any edges to $Sum_w(H)$ that will make it longer. $OPT_{tsp}(G)$ is also the same as $Sum_w(H)$. Since the graph is metric and complete, the shortest path between two vertices will always be the direct path between the two vertices. This means that making a $Sum_w(H)$ will also result in making a $OPT_{tsp}(G)$. □

Lemma 3.14. *Let G be a metric graph then $OPT_{mst}(G) < OPT_{tsp}(G) \leq 2 * OPT_{mst}(G)$.*

Proof. Assume for contradiction that $OPT_{mst}(G) \geq OPT_{tsp}(G)$. Since $OPT_{tsp}(G)$ is a cycle we can remove one edge from $OPT_{tsp}(G)$ and obtain a path P. Clearly P is a spanning tree and since all edge weights are positive the weight of P is less than $OPT_{tsp}(G)$. By definition $OPT_{mst}(G) \leq Sum_w(P)$. Hence we get a contradiction.

Assume for contradiction that $OPT_{tsp}(G) > 2 * OPT_{mst}(G)$. Given an MST $OPT_{mst}(G)$, you can start at one arbitrary vertex and move through every vertex in the MST using a DFS, and make an Eulerian cycle. This is done by moving back the edge when you reach a leaf vertex or have no new vertices to visit for current vertex. This means you are using every edge in $OPT_{mst}(G)$ twice, resulting in $Sum_w(C) = 2 * OPT_{mst}(G)$. Given the lemma 3.13, $OPT_{tsp}(G) \leq Sum_w(C)$. Hence we get a contradiction.

□

Lemma 3.15. *Let G be a graph with an even number of vertices, then $OPT_{match}(G) \leq \frac{OPT_{tsp}(G)}{2}$.*

Proof. Assume for contradiction $OPT_{match}(G) > \frac{OPT_{tsp}(G)}{2}$. $OPT_{tsp}(G)$ is a cycle with n edges. We can split the Hamiltonian cycle corresponding to $OPT_{tsp}(G)$ into two matchings by letting every second edge be in M_1 and the other edges be in M_2, we select M to be the matching from M_1, M_2 with lowest weight and get that $M \leq \frac{OPT_{tsp}(G)}{2}$ hence we get a contradiction as M can not be smaller than the smallest perfect matching. □

If we apply the above lemma on the graph $G[S]$ we get that $OPT_{match}(G[S]) \leq \frac{OPT_{tsp}(G[S])}{2}$ clearly this implies that $OPT_{match}(G[S]) \leq \frac{OPT_{tsp}(G)}{2}$ since we can take a Hamiltonian path in G and shortcut past any vertex not in S. This Hamiltonian path of $G[S]$ will always be shorter since the graph is Euclidean and then all shortcuts will make the cycle shorter.

Lemma 3.16. *Let G be an Euclidean graph and H be the Hamiltonian cycle produced in line 5 of the Algorithm 1, then $Sum_w(H) < \frac{3}{2} * OPT_{tsp}(G)$*

Proof. Let C be the Eulerian cycle produced in line 4 of the Algorithm 1, Then $Sum_w(H) \leq Sum_w(C) = OPT_{MST}(G) + OPT_{match}(G)$ by construction and the fact that the graph G is metric. When making C you add all edges from $OPT_{MST}(G) + OPT_{match}(G)$, which means $Sum_w(C) = OPT_{MST}(G) + OPT_{match}(G)$. When making the Hamiltonian cycle you make shortcuts in the walk by removing two edges e_1 and e_2, and adding one new edge e_3, so that $|e_1| + |e_2| \geq |e_3|$. From lemma 3.13 we get that the length of the path can only either stay the same or get shorter, this shows that $Sum_w(H) \leq Sum_w(C)$. Therefore $Sum_w(H) \leq Sum_w(C) = OPT_{MST}(G) + OPT_{match}(G)$. Given lemma 3.14 $OPT_{MST}(G) \leq OPT_{TSP}(G)$ resulting in $Sum_w(H) \leq Sum_w(C) = OPT_{TSP}(G) + OPT_{match}(G)$. Given Lemma 3.15 $OPT_{match}(G) \leq \frac{OPT_{tsp}(G)}{2}$ resulting in $Sum_w(H) \leq Sum_w(C) = OPT_{TSP}(G) + \frac{OPT_{tsp}(G)}{2}$. Adding this together you get $Sum_w(H) < \frac{3}{2} * OPT_{tsp}(G)$. □

3.5.3 Runtime

The runtime for the algorithm is is calculated from the lines in the Christofides algorithm, algorithm 1.

Line 1. The MST uses the Kruskal algorithm and have a runtime of $O(n \cdot log(n))$.

Line 2. This part goes through all the vertices and checks if the degree is odd, since finding the degree of a vertex can be done in $O(1)$ time this line has runtime $O(n)$

Line 3. The perfect matching uses the algorithm Blossoming algorithm and has a runtime of $O(n^2 \cdot |E|)$. In worst case all nodes of the MST has odd degree and since G is a complete graph, $|E|$ can be $O(n^2)$ giving a total runtime of $O(n^4)$

Line 4. The Eulerian Cycle can be found in $O(|E|)$ time, and since the subgraph we are searching through is a spanning tree + a matching, the runtime is $O(n)$

Line 5. The Hamiltonian is found by short cutting whenever a vertex appears twice in the Eulerian tour. To find all repeated vertices can be done in $O(n)$ time. Selecting one of the repeated nodes to keep and short cutting all occurrences can be done in $O(degree)$ time, since sum of degree is $2n$ (ref lemma) the total runtime of this line is $O(n)$

Line 6. The fixes I have made for the algorithm is $O(n^2 * m)$. Since the Hamiltonian cycle the fixes are applied to have the same number of edges and vertices, the runtime will be $O(n^3)$

4 Programming

In this section we describe the programming done in association with this thesis. Since Christofides algorithm has an approximation ratio of $\frac{3}{2}$ we know that for any metric graph the output will be somewhat reasonable. If we want to improve on this there are two main approaches, improve either the approximation ratio for all graphs or the results for typical inputs of the problem. As we regarded the task of improving the approximation ratio of Christofides algorithm for quite hard we have instead focused on testing out various ideas on a real world dataset from the Ormen Lange field. Since the graphs obtained from converting point sets into times is no longer Euclidean it was not clear to us which of the rules applying to Euclidean graphs would also apply to these graphs.

4.1 The data set

The dataset of points used in this algorithm is extracted from the map of "Ormen Lange" obtained from [Vatshelle et al., 2017], see figure 4. A grid was drawn over the image of the field and each point were given coordinates corresponding to this grid. These coordinates are not exact coordinates, but are read with one decimal. To get the exact distance between two points the scale at the bottom of the image were used to calculate the actual distance between two points, and then calculate the difference in percentage from measured distance to actual distance.

Input to the algorithm is information about the Oil Field "Ormen Lange". The first is all the points at the bottom of the sea floor. This is given as X, Y coordinates in a .txt file named points.txt. Here there are 115 lines with coordinates that the algorithm uses to make vertices for the graph. There are no edges given to the graph, since this is a boat driving out on the sea, which means this is a complete graph. The algorithm is also getting information to scale the field, since the coordinates in the file are not true to size. This is a number that combined with the distance between two points will give the actual distance in the field.

Since there is a boat that drives between these points the algorithm is also getting information about this boat. This is the max speed at which the boat can drive, the acceleration for the boat and the time it takes to lower and raise the measuring equipment.

4.2 Programming environment

The implementation for this algorithm is written in the Java programming language version 15, using IntelliJ as the integrated development environment (IDE). The full implementation for this algorithm is now also public on GitHub. [Johansen, 2021]

4.3 Algorithms and data structures

For the minimum spanning tree, I implemented the Kruskal's algorithm. Since Union Find is something that does not exist in Java, but is something that this algorithm requires, I also implemented this.

For the perfect matching I did not write the Blossoming algorithm. For this I found an already written code that I have used, written by Lucas Siemond. [Siemond, 2018] To implement this code into my project I had to convert the input and output for the blossoming algorithm to match the code that I have written. Also due to the fact that the blossoming algorithm is a maximum spanning tree, and the Christofides algorithm requires a minimum spanning tree, I did have to convert the input to get the required output.

To be able to see the path that the algorithm is producing I used JFrame and JPanel together with graphics to make a visualizer that shows the path as an image. To be able to convert the data from the input graph to the vessel routing graph, I also made a Singleton instance named "DataConverter".

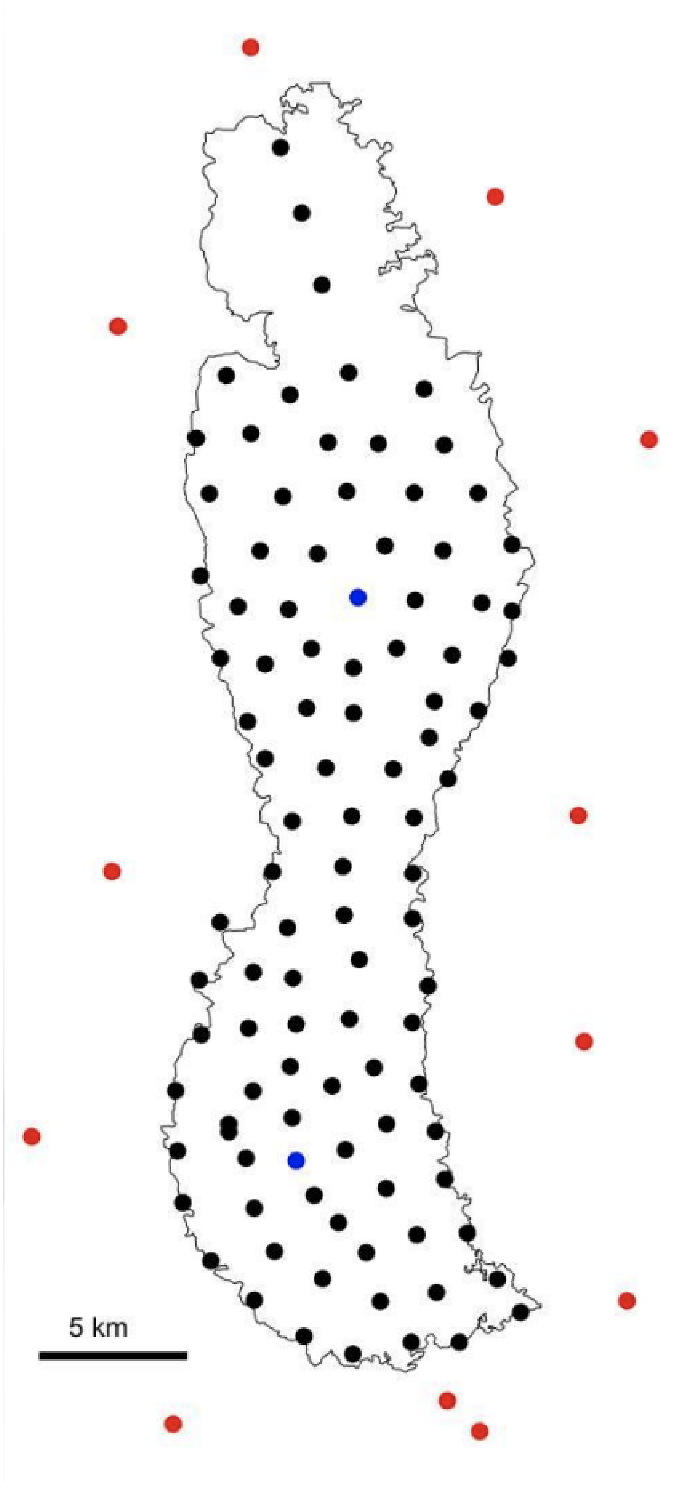

Figure 4: This is an illustration of The Ormen Lange field, taken from [Vatshelle et al., 2017]

4.4 Visualizing

To be able to analyse the results better I have made a visualizer in the program that shows the graph and the different paths as images. This visualizer print out one image of the map for each of the important steps in the algorithm. The MSP, perfect matching, Eulerian cycle, and every version of the Hamiltonian cycle. The Visualizer uses JFrame and JPannel as frameworks for the image, and then draws the edges and vertices with graphic.

It was also a good help to have the map when trying to find some improvements for the algorithm. When you visualize the map it is easier to see where improvements can be done, and also what needs to be done in order to improve the algorithm. The images of the different results from the algorithm shown in this theses are made with this visualizer.

4.5 Improvements for the program

Since the Christofides algorithm does not give an optimal solution for the Traveling Salesman Problem, there are small fixes you can make to shorten the walk even more. The two I have chosen to implement in my program is to remove all crossing lines 4.5.1 and to change the path of some vertices 4.5.2 to try to find an shorter path.

4.5.1 Remove all crossing edges

First start with going over every pair of edges in the walk and removing all crossing edges, replacing them with new none crossing edges. Since replacing two edges may make new crossing edges in another part of the walk, the program are doing this operation until there are no more crossing edges in the walk.

The runtime of checking every pair of edges to see if they are crossing is N^2. The number of times you need to go over the graph to check for new crossing edges are depending on the graph, stopping if there are no more crossing edges, but in worst case the runtime is N. So the worst case runtime for this improvement is N^3.

Algorithm 2: Improvement 1

```
  Data: A hamiltonian cycle $H$.
  Result: A shorter hamiltonian cycle $H'$
1 for $e_1 \in edge$ do
2     for $e_2 \in edge$ do
3         if crossing($e_1, e_2$) then removeCross($e_1, e_2$);
4     end
5 end
6 return $H'$;
```

4.5.2 Change the placement of a vertex in the walk

After getting the Hamiltonian cycle from the Christofides algorithm, and doing the additional improvement of removing any crossing lines. There are still one more thing you can do to try to improve the path by making it even shorter. This is to try an find vertices in the walk, that can be moved to another position in the walk, making the overall result shorter.

Since the graph G is an Euclidean graph the idea for this improvement will also hold. Given an Hamiltonian H, five vertices in H, v_1, v_2, v_3, v_4, and v_5, and tree edges $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, and $e_3 = (v_4, v_5)$, you can check if moving the vertex v_2 will make the H shorter. Let $L_1 = w(e_1) + w(e_2) + w(e_3)$ and $L_2 = w((v_1, v_3)) + w((v_4, v_2)) + w((v_2, v_5))$. If $L_1 > L_2$ the path will be shorter if you change the placement of the vertex. By doing this you also create a new edge from v_1 to v_3 in H, replacing the once going through v_2.

By doing this you can end with a little dilemma. By changing the placement of one vertex in the walk, to another position, this change can make a new set of crossing edges. The overall length of the walk will be shorter by doing these replacements, but the previous improvement of removing crossing edges can now be done over again. This means that the program will be going back and fourth between these two improvements till one of them no longer have a positive impact on the result of the walk.

Lemma 4.1. *If a vertex v in a Hamiltonian is closer to an edge e than the edge connecting its neighbours, then moving n to between the vertices in e will shorten the length of H.*

Proof. Given a Hamiltonian H, five vertices v_1, v_2, v_3, v_4 and v_5, and four edges $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, $e_3 = (v_4, v_5)$ and $e_4 = (v_1, v_3)$ in H. If v_2 is closer to e_3 than e_4 in H, than H will become shorter if you move the placement of v_2 in H to be between v_4 and v_5. This is because H will have to walk both e_3 and e_4 regardless, but having v_2 closer to the corresponding edge means the detour through v_2 will make the walk shorter overall. □

Algorithm 3: Improvement 2

```
Data: A hamiltonian cycle $H$.
Result: A shorter hamiltonian cycle $H'$
for $e \in edge$ do
    for $v \in vertices$ do
        if shorterIfMoved($e, n$) then move($e, n$);
    end
end
return $H'$;
```

5 Results

After implementing the Algorithm 1, the program can produce a Hamiltonian Cycle, see Lemma 3.12. After making the tour, the Algorithm then prints out the different results it has found to the console.

Since there is no efficient way to find an exact solution to the travelling salesman problem, there does not exist any exact result that I can compare to, so seeing how it compare to the actual shortest path is difficult. Instead you can set a lower and a upper bound to see how the result compare to these. A lower bound for this problem is the $OPT_{mst}(G)$. Since $\frac{Sum_w(E) \cdot 2}{3} \leq OPT_{TSP}(G)$, see Lemma 3.14. This means that the result from the algorithm can be no lower that $OPT_{mst}(G)$. There is also an upper bound to the problem. The result given by the algorithm can be no greater that the Eulerian cycle that the algorithm provides. This is because the algorithm is making a Hamiltonian, from the Eulerian, and given the Lemma 3.13.

The last step of the original Christofides algorithm is to make the Hamiltonian, using the Eulerian cycle from the previous step. For this part the algorithm removes duplicates of vertices at random. Instead of using this approach, my algorithm is making a quick calculation of the path to see which edge makes the shortest path, and remove the one that makes the longest. By using this calculation the result will be no greater than the random removal, but has the potential to make a shorter path.

5.1 Results from the algorithm

The algorithm produces both the time it takes for the offshore vessel to travel the distance, and the actual distance of the tour. It gives the distance in both meters and km, and the time in both seconds and hours. The results calculated are the length of the minimum spanning tree and the perfect matching, and the length and duration for the Eulerian circuit, Hamiltonian with random path chosen, Hamiltonian with bets path chosen, and the Hamiltonian with improvements.

	Length	Time
MST	232.05 km	
Matching	100.27 km	
Eulerian	332.32 km	39.09 h
Hamiltonian Random	313.42 km	34.01 h
Hamiltonian	298.83 km	33.20 h
Hamiltonian with Improvements	280.97 km	32.21 h

I have also calculated the percentage between different results. The first is the difference between the minimum spanning tree and the Hamiltonian with Improvements. The second is the difference between the Hamiltonian with Improvements and the Eulerian, third is the difference between the Hamiltonian with Improvements and Hamiltonian with random chosen path, and last is difference between Hamiltonian with Improvements and Hamiltonian with best chosen path.

Percentage:

	percent
Bigger than MST	17.41
Less than Eulerian	15.45
Less than Hamiltonian Random	10.35
Less than Hamiltonian	5.98

5.2 The map

Figure 5, 6, and 7, is maps made by the visualizer, that shows the actual path that the algorithm is taking. Figure 5 shows the path made by the christofides algorithm where the the removal of vertices going from the Eulerian to Hamiltonian is chosen at random. Figure 6 shows the path made by the christofides algorithm where the removal of vertices going from the Eulerian to Hamiltonian is calculated to choose the edges that results in the shortest path. Figure 7 shows the path made in figure 6, but here both the improvements are also applied.

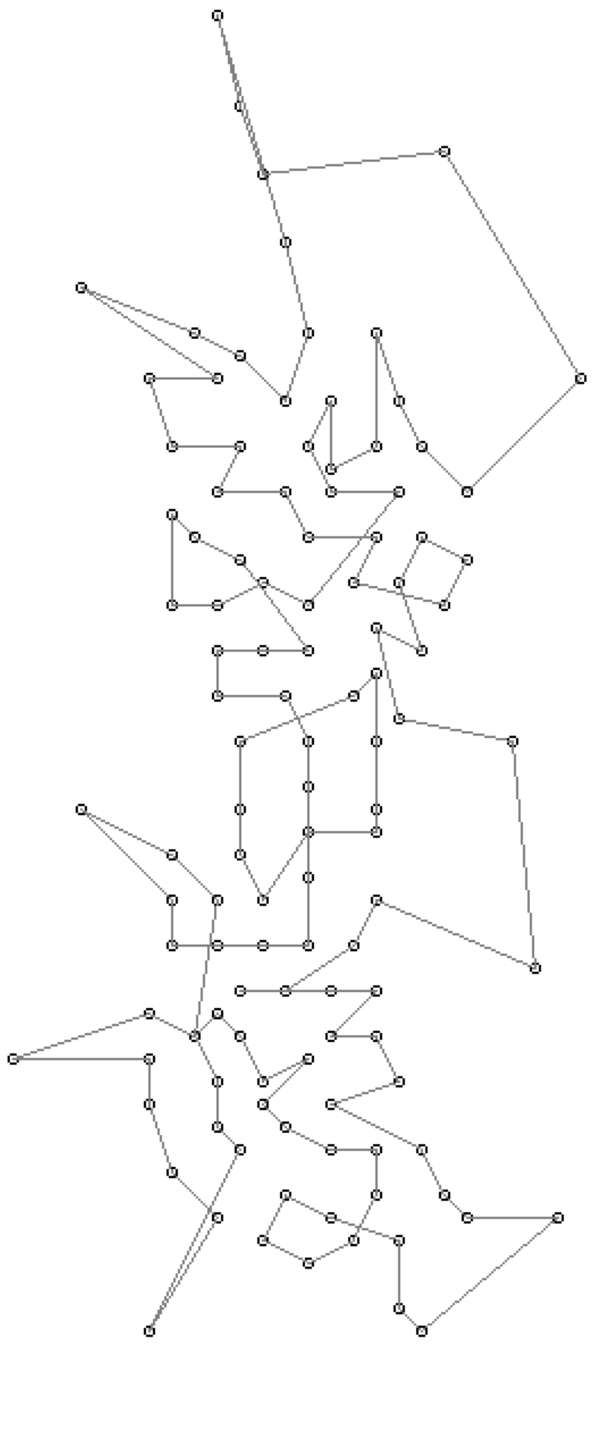

Figure 5: This figure shows the Hamiltonian cycle where the edges removed when going from Eulerian to Hamiltonian is chosen at random.

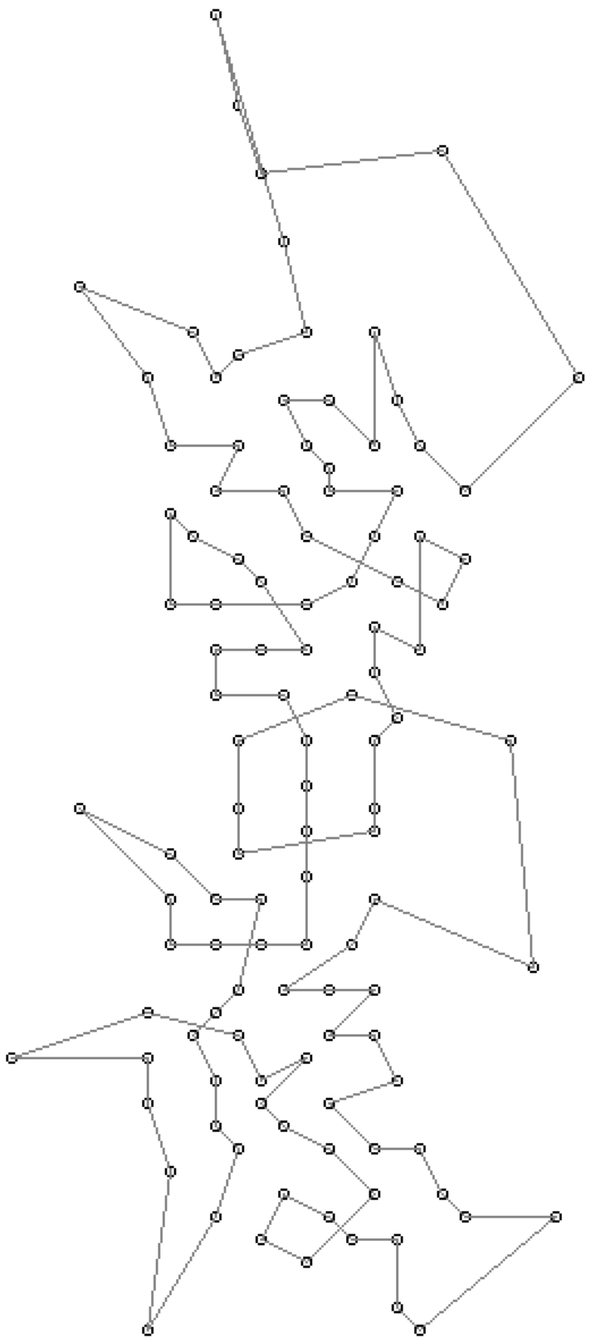

Figure 6: This figure shows the Hamiltonian cycle where the edges removed when going from Eulerian to Hamiltonian is calculated to choose the shortest path.

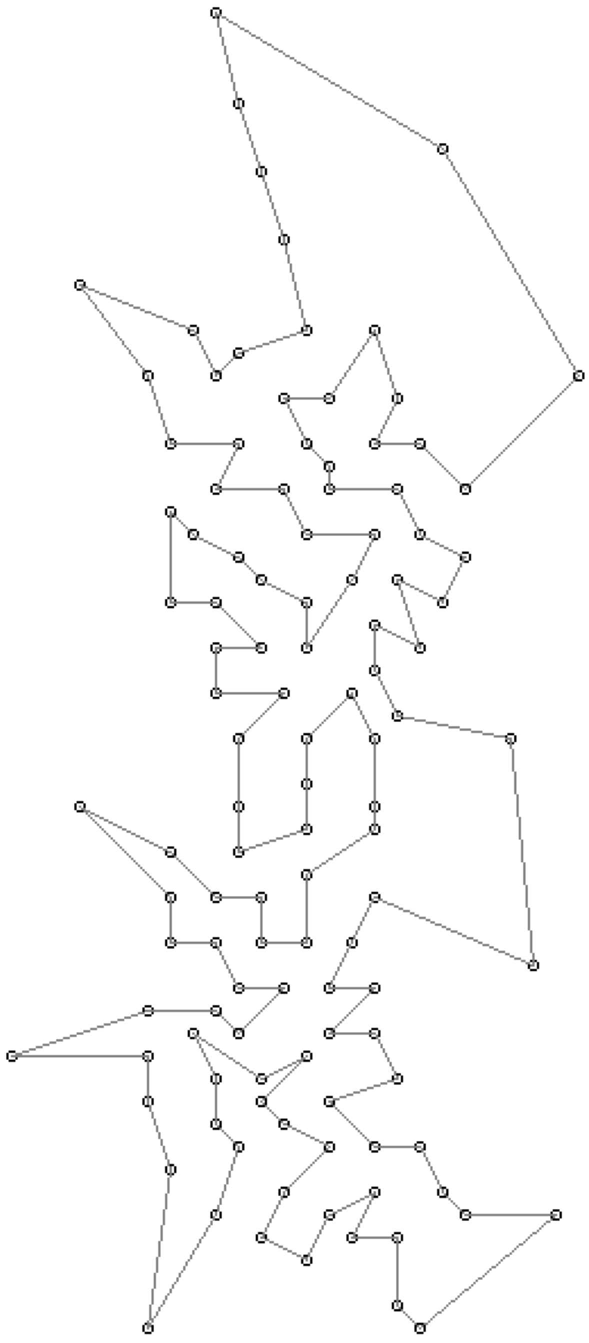

Figure 7: Hamiltonian with Improvements. This figure shows the results from 6 where the extra improvements have been added.

5.3 Comparison of different methods

I have also tried to see if one of the improvements I have added to the algorithm are better than the other. Here only one of the improvements are added to the algorithm before the results is printed. As you can see in the table below both improvements given by the algorithm result in a shorter walk than the original Christofides Algorithm, but both are also shorter than both improvements combined. The table also shows that the changing the place of the vertex actually gives a slightly better result than the removal of crossing edges. Figure 8 shows the path made in figure 6, but only the improvement where the placement of one vertex is applied. Figure 9 shows the path made in figure 6, but only the improvement where the crossing edge is removed is applied.

The reason both of these are applied to the algorithm as the improvements is that after removing crossing edges you might end up with a vertex that can change place in the walk and therefore make the length shorter. the same holds for the other, after changing the placement of one vertex in the walk you might make two new edges crossing, and replacing them with none crossing edges will make the walk shorter. To get the best result here you need to apply these two improvements to the walk until one of them do not make the walk any shorter. This is when no vertex can change place in the walk to make it shorter, or no edges are crossing after a vertex has changes place.

There are a lot of other improvements that could have been added to this algorithm, but I decided to only implement the two mentioned above. One of the extra improvements that could have been added is to change a subsection $S \subset H(E)$, such that the length $Sum_w(H)$ becomes shorter. Sometimes changing the placement of each vertex $v \in S$ individually does not make $Sum_w(H) shorter$, but changing the placement of S in the graph will.

	Length	Time
Change place of vertex	288.20 km km	32.61 h
Removing crossing edges	287.35 km	32.56 h
Both Improvements	280.97 km	32.21 h

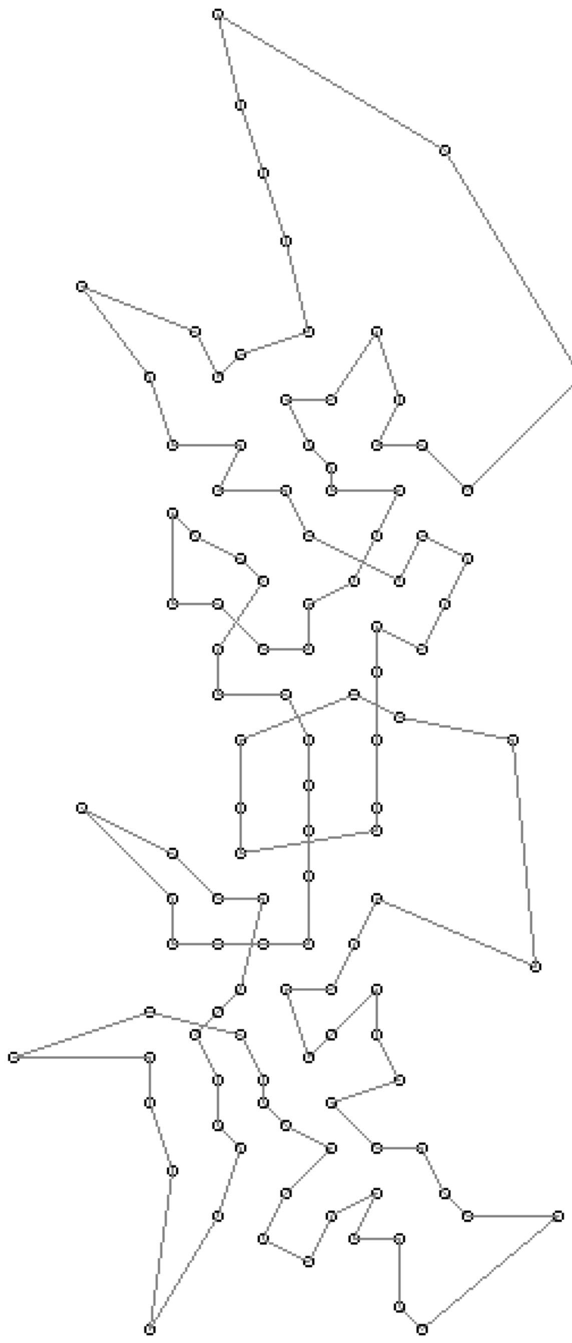

Figure 8: This figure shows the result where the algorithm have only applied the improvement where one vertex changes its place in the path to get a shorter path.

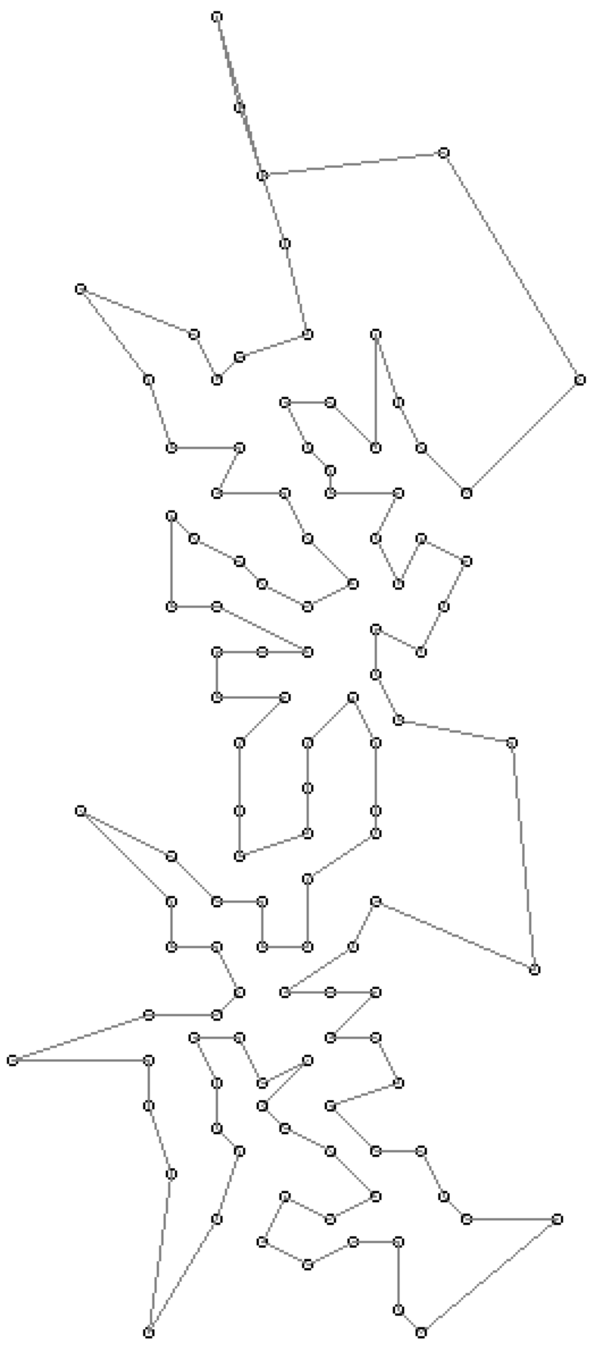

Figure 9: This figure shows the result where the algorithm have only applied the improvement where every two crossing edges have been replaced with none crossing edges.

References

[Aneichyk, 2009] Aneichyk, T. (2009). Simulation model for strategical fleet sizing and operational planning in offshore supply vessels operations. Master's thesis, Høgskolen i Molde.

[Diestel, 2017] Diestel, R. (2017). *Graph Theory*. Springer.

[Flood, 1956] Flood, M. M. (1956). The traveling-salesman problem. *Operations Research*, 4.

[Ghiani et al., 2006] Ghiani, G., Laporte, G., and Semet, F. (2006). The black and white traveling salesman problem. *Operations Research*, 54.

[Johansen, 2021] Johansen, I. (2021). The traveling salesman problem. https://github.com/Ingrid97/TSP. git commit: 3e3317f.

[Little et al., 1963] Little, J. D. C., Murty, K. G., Sweeney, D. W., and Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations Research*, 11.

[Roberti and Ruthmair, 2021] Roberti, R. and Ruthmair, M. (2021). Exact methods for the traveling salesman problem with drone. *Operations Research*, 55.

[Roberts and Flores, 1966] Roberts, S. M. and Flores, B. (1966). An engineering approach to the traveling salesman problem. *Operations Research*, 13.

[Siemond, 2018] Siemond, L. (2018). Maximum weighted matching - edmonds blossom. https://github.com/simlu/EdmondsBlossom. git commit: 8e52ab4.

[Shoemaker et al., 2016] Shoemaker, A. and Vare, S. (2016). Edmonds' Blossom Algorithm Stanford University, CME 323

[Toriello et al., 2014] Toriello, A., Haskell, W. B., and Poremba, M. (2014). A dynamic traveling salesman problem with stochastic arc costs. *Operations Research*, 62.

[Vatshelle et al., 2017] Vatshelle, M., Glegola, M., Lien, M., Noble, T., and Ruiz, H. (2017). Monitoring the ormen lange field with 4d gravity and seafloor subsidence.

[Zambito, 2006] Zambito, L. (2006). The traveling salesman problem: A comprehensive survey. https://www.semanticscholar.org/paper/The-Traveling-Salesman-Problem%3A-A-Comprehensive-Zambito/a5cdc315936617eb0e41ad54095950dba04b9a84.

[Zeng, 2014] Zeng, C. (2014). Optimal offshore supply vessel planning: A case study of a chinese offshore oil and gas production area. Master's thesis, University of Stavanger.

[Øyra Friedberg and Uglane, 2013] Øyra Friedberg, D. and Uglane, V. T. (2013). Routing and scheduling of platform supply vessels. Master's thesis, Norwegian University of Science and Technology. With: SINTEF, http://www.sintef.no.

[Octio, 2021] Octio AS. (2021). Gravimetry and seafloor subsidence surveys. https://www.octio.com/gravimetry-and-seafloor-subsidence-surveys/.