University of Bergen (UiB)

Master Thesis presented for the Master of Science in Energy

Optimization of the Offshore Wind Inter-Array Cable Layout problem using Heuristic based Algorithms

Sunney Fotedar

Geophysical Institute, UiB, Norway

May 29, 2018

Optimization of the Offshore Wind Inter-Array Cable Layout problem using Heuristic based Algorithms

Sunney Fotedar

May 29, 2018

ABSTRACT

The current work presents an in-exact solution method used to identify feasible, and less costly inter-array cable layout for offshore wind farms. The solution method developed has been built considering the interests of wind farm developers in mind, and to support them in the planning of large offshore wind projects. The objective of the current study is to develop a fast heuristic based algorithm able to find good (less costly), feasible solution, with a small optimality gap.

We are given the positions of the turbines, obstacles, and substations. The optimization problem is to find a cable layout such that there is a unique path from each turbine to one of the substations. All the turbines are connected in a series connection on a cable having a pre-defined capacity limitation. There are few additional constraints such as to prevent two or more cables from crossing each other, and cables from entering any restricted areas in the sea bed. This problem is quite similar to the well-known Capacitated Minimum Spanning Tree (CMST) problem.

The cable layout problem has been proved to be *NP hard*, thus, an exact algorithm is likely to have a running time that is an exponential function of the size of the input. Most of the available exact models require fast computers, and hours of computation time to find an optimal solution and still, in large instances of the problem, an optimal solution is not achieved. Although our heuristic does not guarantee an optimal solution, it has the ability to reveal good, feasible solutions in short time frame for large as well as small instances. We have implemented the heuristic in Java, and used in-built as well as customized data structures for improving the running time of the algorithm.

We have tested our solution method on 8 real wind farm instances with total number of turbines ranging from 30 to 160. We have compared the results of our heuristic with the optimal solutions available for 4 wind farm instances. We achieved near optimal solutions (<1%) in most of the instances. The solution method includes a construction heuristic, which is a modified version of a well established greedy heuristic for CMST problem called Esau Williams' heuristic. We have adapted this heuristic by introducing a procedure to find a crossing free layout, and also used a *shape factor* in the heuristic function to improve the solution quality.

We have utilized a multiple node exchange neighborhood structure, and used it in a local search framework. The local search method improves the solution quality, and finds locally optimal solutions. In the end , we have

used the algorithm on some of the larger instances with more than 100 turbines. These instances are practically impossible to solve using exact methods. We have compared our heuristic results in these instances with the actual installed layout.

ACKNOWLEDGEMENT

I would like to express my gratitude to the Institute of Informatics, University of Bergen for providing me good working environment to conduct my research and studies. I would like to thank my supervisor Prof. Dag Haugland for his constructive inputs, criticisms and constant feedback. I would also like to thank my co-supervisor Dr. Ahmad Hemmati from the Optimization group, UiB for his feedback and comments on my work. My work would not have been possible without their constant efforts. I would like to thank Hordaland fylkeskommune for supporting my thesis work. I am also thankful to all my teachers, and student advisors at UiB.

Lastly, I would like to thank my wife Dr. Indu Dhar for providing me constant support and encouraging me to perform well during my masters studies. This would not have been possible without her efforts and sacrifice.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CAPEX	Capital Expenditure
EWEA	European Wind Energy Association
TSP	Travelling Salesman Problem
MILP	Mixed Integer Linear Programming
CMST	Capacitated Minimum Spanning Tree
IEA	International Energy Agency
OWCLP	Offshore Wind Cable Layout Problem
MST	Minimum Spanning Tree
VLSI	Very Large Scale Integration

1 INTRODUCTION

Offshore wind energy is gradually emerging as a significant new source of electricity especially in European countries such as U.K., Germany, Belgium, Denmark and The Netherlands. Europe buys 40% of all wind turbines sold globally [EWEA, 2017a]. According to the International Energy Agency (IEA), wind energy should meet a quarter of Europe's power demand by 2030. The daily wind energy data accessed from WindEurope's website on 13 October 2017 showed the share of wind energy in total daily electricity supply in EU countries was 7%. This included onshore wind contributing $455GWh$, and offshore wind contributing $106GWh$. This supply could power approximately 138 million average EU households [EWEA, 2017a]. The total grid connected offshore wind capacity in Europe is $11.5GW$, and a total of $21.7GW$ [EWEA, 2017b] of new projects have been consented. This includes a total of $3,344$ offshore wind turbines fully grid connected in European waters in 82 wind farms across 11 countries.

Power is directly proportional to the cube of the upstream wind velocity, so placing wind turbines in sea having high wind velocities results in higher energy production. Although beneficial in terms of power output, offshore wind comes along with its own set of challenges. Offshore wind farms are generally more expensive than their onshore counterpart. Due to deflated electricity prices in Western Europe, it is difficult for the developers to get financing for offshore projects, and renders heavy reliability on government subsidies. Most of the developers look to cut costs during the planning stage. One of the options available to offshore wind developers is cutting down investments in expensive sub-sea cables. This can be achieved using mathematical optimization, which is the approach taken in the current work.

The thesis is organized as follows, in this chapter, we discuss the problem statement, and constraints involved in the cable layout problem. In Chapter 2, we discuss the literature, and introduce some of the exact, and inexact methods already developed. We also discuss some of the well-known heuristics for an optimization problem similar to our problem called Capacitated Minimum Spanning Tree (CMST). We also discuss some background required to understand algorithms developed in Chapter 4 and Chapter 5. In Chapter 3, we introduce a Mixed Integer Programming Model (MILP) representing the cable layout problem. In Chapter 4, we present a construction heuristic. In Chapter 5, we develop a local search method to improve the solution quality. In Chapter 6 , we present experimental results obtained by using the developed solution method, and check its effectiveness by comparing it with the available optimal solutions.

1.1 OFFSHORE CABLES

Whereas most of the traditional power generating sources are built around a few high rating generating units at a single location, wind farms collect the power generated by multiple wind turbines spread over a large area. The energy generated at each wind turbine is collected through inter-array cables, and sent to shore using an export cable. Due to increasing distance of offshore projects from the shore, most of the current offshore wind farms have either single or multiple substations. These substations are required to step up the voltage and send power to the onshore terminal using high voltage export cables. The medium voltage cables which collect and channel the power to the substations are called *collector system* or *inter-array cables*.

Designing a wind farm is a multi-step process, which involves identifying a potential site, developing wind resource map, identifying optimal turbine locations, selecting inter-array cable type, and finalizing the cable layout. The above mentioned steps are done with an objective of minimizing overall investment cost and maximizing power production. There is also need to maintain high reliability of the offshore wind farms.

Currently, the above is achieved by a combination of mathematical modelling, statistical analysis, and some practical ad-hoc methods. The power losses due to wake effect can be accounted for by using various analytic wake models, wind resources are assessed using historical data. Both wind conditions, and wake effects are utilized to identify optimal positioning of the wind turbines in the approved area. Once the position of turbines are fixed, then inter-array and export cable layout is finalized. Since export cables are high voltage cables that run a long distance from substation to the onshore location, there are few decision variables to be manipulated. However, inter-array cable layout optimization is a challenging mathematical task, especially in wind farms with large number of turbines and multiple substations.

1.2 PROBLEM DEFINITION

Following is the optimization problem that we solve in this work:

Given the positions of turbines and substations, find a cable layout such that there is a unique path from each turbine to one of the substations, and the overall cable length is minimized. The constraints to be satisfied are: No *two or more cables should cross, splitting of power cables leaving turbines is not allowed, a predefined maximum number of turbines can be connected on a single cable, and cables cannot enter any restricted areas in the sea bed.*

The cable layout has a tree structure similar to the one shown in Figure 1.2.1 with root in the substation. While developing an optimization model for the cable layout problem, one has to make few decisions related to the graph representation of the problem. Most of the available models consider only turbines and substations as nodes in the final solution, but some also use optional nodes (Steiner nodes) around turbines. The latter option as opposed to former, doesn't lead to sub-optimal solutions [Fischetti and Pisinger, 2016], [Klein and Haugland, 2017]. The restricted areas in the sea bed are modeled by placing optional nodes at the vertices of the convex hull of a restricted area. The addition of optional nodes results in increasing the size of the problem, and makes it even more harder to solve.

One of the crucial constraints which requires special attention while solving this optimization problem is the *cable crossing* constraint. The two main reasons behind such a constraint are the need for expensive bridge structures to allow cable crossing, and thermal interference between the two crossing cables resulting in reducing the cable capacity [Fischetti et al., 2015] [Bauer and Lysgaard, 2015].

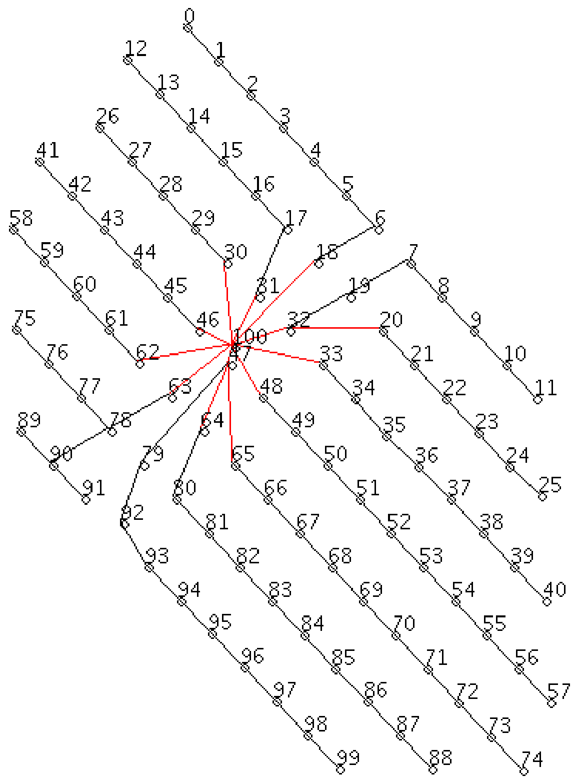

Figure 1.2.1: An illustration of a feasible cable layout for Thanet Offshore Wind Farm with 100 turbines. The red line is the cable going into the substation from each rooted tree. The cable capacity used is 10

2

LITERATURE REVIEW

Due to increased interest in offshore wind energy in Europe, various articles pertaining to offshore cable layout optimization have been published in previous few years. Most of the articles are using Mixed Integer Linear Programming (MILP) models to minimize the cost of cables by minimizing the cable length. The focus is mainly on the layout of inter-array cables. [Bauer and Lysgaard, 2015] have compared the problem to an open vehicle routing problem with unit demands and planarity constraint. The authors have used a route based structure. It is highlighted in [Klein and Haugland, 2017] , [Pillai, 2017], [Fischetti and Pisinger, 2016] that branching is allowed, and tree structures could be used for inter-array cable layout. There is an opportunity to reduce the cable length using branching, and parallel cables. [Pillai et al., 2015] have used a path finding algorithm to avoid non-convex as well as convex obstacles. All the models have used commercial MILP solvers to achieve optimal solutions of the cable layout problem.

[Fischetti and Pisinger, 2016] have not only included the cable costs, but also taken into consideration future revenue losses due to power flow in the cables. They have presented a MILP model to optimize the routing by considering both cable cost and power losses in the objective function. However, as opposed to [Pillai et al., 2015] and [Wędzik et al., 2016], where a non-linear objective function is used for including power losses, [Fischetti and Pisinger, 2016] uses a pre-computing strategy to avoid complicated quadratic models. In most of the above mentioned work, the authors have not included non-crossing constraints in the formulation, but added them dynamically after getting infeasible solution from the MILP solver.

2.1 SOLUTION METHODS

Authors [Chen et al., 2013], [Klein and Haugland, 2017], [Bauer and Lysgaard, 2015] and [Hou et al., 2016] have developed exact solution methods to solve the layout problem, and they have been reasonably successful in their approach. [Klein and Haugland, 2017] were able to solve all the instances for Barrow (30 turbines), Walney 1 (51 turbines), Walney 2 (51 turbines) optimally, except Sheringham Shoal (88 turbines and 2 substations). [Hou et al., 2016] have solved the problem using a dynamic minimum spanning tree algorithm for up to 80 turbines and achieved optimality.

[Gonzaélez-Longatt et al., 2012] have developed a genetic algorithm based solution method for the cable layout problem. Although they have achieved near optimal solutions for various test instances including more than 200 turbines, but it uses a fixed start open multiple TSP (mTSP) problem which doesn't allow for branching and thus, adds unnecessary constraints. [Pillai et al., 2015] have formulated a constrained version of the capacitated minimum spanning tree problem and used Gurobi 5.6 MILP solver that combines simplex solving approaches, cutting plane algorithm, heuristic algorithms and terminates when it reaches a threshold of optimality gap. [Macedo de Lacerda and de Medeiros Junior, 2006] have developed a genetic algorithm for a capacitated minimum spanning tree problem with unit demands. They have suggested a new method for cross-over and encoding the candidate solutions. [Dahmani et al., 2015] have developed a novel genetic algorithm based approach in which non-crossing constraints are taken into consideration. One major criticism of the model is its inflexibility. Each node can only have 8 neighbouring nodes with which it can form an arc, apart from one with the substation. This reduces the number of variables and makes non-crossing constraints manageable, but the model loses on providing flexibility.

Given the cable capacity, final task after identifying the location of turbines, substations, and obstacles is to find optimal cable connections. Since the MILP model cannot be solved to optimality in larger instances, we develop fast heuristics which can provide good solutions in less time. We will compare our solutions with optimal solutions achieved in [Klein and Haugland, 2017] for 4 wind farm instances. We have chosen this MILP model for comparison, because it includes all the relevant constraints, and is the most accurate representation of the layout problem. Similar to [Klein and Haugland, 2017], we also allow parallel cables and branching at turbine locations.

The above mentioned problem has resemblance to the classical Capacitated Minimum Spanning Tree (CMST) problem with additional constraints. In the next section, we describe the CMST problem , and related exact and in-exact solution methods.

2.2 CAPACITATED MINIMUM SPANNING TREE

The Capacitated Minimum Spanning Tree (CMST) problem is a generalization of the well known Minimum Spanning Tree (MST) problem with a pre-defined root r, and a capacity limitation of K. The capacity limitation means that each sub-tree can have a total demand K or less. The sub-tree can be defined as the maximal sub-graph of the tree connected with the root using a single edge. In Figure 2.2.1, a capacitated minimum spanning

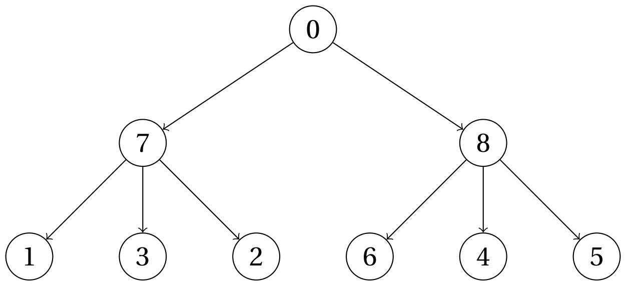

Figure 2.2.1: An illustration of capacitated minimum spanning tree rooted at 0

tree rooted at 0 and capacity limitation $K = 4$ is shown.

Now, we develop some notation to be used in succeeding section. We consider a connected graph $G = (V, A, f, c, K)$, with node set $V = \{0, 1, 2, .., n\}$, A is the set of arcs , f_i is the non-negative demand at each node $i \in V$, and c_{ij} represents the cost of the arc between node i and node j (Note: $c_{ij} = c_{ji}$). The root node is represented by 0. The arc set $A = (V \times (V \setminus \{0\}))$

A rooted sub-tree is represented by T_i, which is the rooted sub-tree connected with the center node 0 using an arc $(0, i)$. The sum of the demands at each node in any rooted sub-tree T_i, where $i \in V \setminus \{0\}$ cannot exceed K. Thus, if all nodes have unit demand $f_i = 1$, there can be at most K nodes in each rooted sub-tree. The minimum spanning tree problem has an efficient or polynomial algorithm, but the capacitated minimum spanning tree problem has been proved to be *NP hard*, even with uniform demand at each node [Sharma and Bardai, 1970]. Since the CMST is *NP hard*, exact algorithms are likely to have an exponential running time. Due to the intractability of the problem, numerous heuristics as well as meta-heuristics have been proposed for use in various fields, such as telecommunication design and network design [Voss, 2008].

Although there are many similarities between the offshore wind cable layout problem (OWCLP) and the CMST problem, there are also quite significant differences. Unlike CMST problem, in the OWCLP, the demand is at the root nodes (substations), and cable crossing constraints are applicable. The latter makes OWCLP even more harder to solve than CMST.

2.2.1 *Mathematical formulation*

In this section, we discuss one of the mathematical formulation for CMST problem, which forms the basis for a MILP model for OWCLP. A great variety of mathematical formulations have been suggested in the literature and a detailed review can be found in [Voss, 2008]. Assuming $f_i = 1$ for all $i = \{1, \cdots, n\}$, and $f_0 = 0$, the CMST problem can be formulated as a mixed integer linear programming (MILP) model. We define $x_{ij} = 1$, if arc (i, j) is in the final solution and $x_{ij} = 0$, otherwise. We introduce a continuous non-negative flow variable y_{ij}, which represents the amount flowing through the arc (i, j) for all $i = \{0, \cdots, n\}$ and $j = \{1, \cdots, n\}$.

Let us use the graph: $G = (V, A, f, c, K)$ with $f_i = 1$ for all non-central nodes, and $f_0 = 0$. A MILP model is formulated for the CMST problem:

$$\min \sum_{i=0}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} \tag{2.2.1}$$

$$s.t \qquad \sum_{i=0}^{n} x_{ij} = 1 \qquad \forall j \in \{1, ..., n\} \tag{2.2.2}$$

$$\sum_{i=0}^{n} y_{ij} - \sum_{i=1}^{n} y_{ji} = 1 \qquad \forall j = \{1, .., n\} \tag{2.2.3}$$

$$x_{ij} \leq y_{ij} \qquad \forall (i, j) \in A \tag{2.2.4}$$

$$y_{ij} \leq (K-1) \cdot x_{ij} \qquad \forall (i, j) \in A \tag{2.2.5}$$

$$x_{ij} \in \{0, 1\}, y_{ij} \geq 0 \qquad \forall (i, j) \in A \tag{2.2.6}$$

The constraint (2.2.2) ensures that only one arc reaches every non-root node. The constraint (2.2.3) makes sure that the difference between the flow units going in and leaving out is 1, that is, a unit demand at each node $i \in V \setminus \{0\}$.

The flow variable y_{ij} has been introduced to prevent the formation of sub-tours and ensure connectivity of the solution. [Toth and Vigo, 1995] have presented an exact model using a branch and bound based algorithm. [Gouveia and Martin, 1999] have proposed a hop-indexed generalization of the above formulation, [Malik and Yu, 1993] have used Lagrangian sub-gradient optimization, [Malik and Yu, 1996] have used cutting plane algorithm for more than 200 nodes. Although there are plethora of exact algorithms available, still the interest has been on fast heuristics which can solve large instances of the above problem in less time. [Chandy and Russel, 1972] have shown experimentally that Martin's [Martin, 1967] and Esau-Williams' [Esau and Williams, 1966] heuristics provide near-optimal solutions with Esau Williams' heuristic having a slight edge over the others.

2.3 BACKGROUND

In this section, we briefly discuss some necessary background required to develop the algorithms in later sections. Some of the key topics covered are Esau Williams' heuristic, Convex hull, Graham Scan algorithm and a brief review of commonly used local neighborhood structures for the *CMST* problem.

2.3.1 *Esau-Williams*

[Esau and Williams, 1966] proposed a greedy heuristic based on a real valued reduction function which maps each non-root node to a potential cost reduction value(see equation (2.3.4)). We use the same connected graph G as discussed in the previous section with unit demand at each non-root node and no demand at the root node 0. In the first iteration of the Esau-Williams' heuristic, we start with a costly, but feasible star layout. In the star layout, every node i is connected to the root node 0 forming an arc $(0,i)$. However, in each iteration of Esau-Williams' heuristic a central arc $(0,i)$ is traded off for a better link with any of the neighboring node satisfying certain conditions to be discussed later.

The equation (2.3.1) states that there can be at most K nodes in each rooted sub-tree T_i. The rooted sub-tree T_i is a component of a tree spanning V as its maximal sub-graph uniquely connected to the root by an arc (0, i). We define $N(T_i)$ as the set of nodes in the rooted sub-tree T_i. We define X_i as the set of nodes in the same sub-tree as node $i \in V \setminus \{0\}$. The equation (2.3.4), gives the the reduction value for each node $i \in V \setminus \{0\}$, calculated in each iteration of the Esau-Williams' heuristic. We define M as the set of $|V|-1$ arcs constituting a spanning tree of the graph G.

The reduction value (R_i) is the maximum cost reduction that is achieved by removing the central arc $(0,i)$ from M and still maintaining the connectivity of the spanning tree M by adding an arc with a node $j \notin X_i$, such that c_{ji} is minimized. We define a set $S(i)$ containing all the feasible neighboring nodes of the node i using the equation (2.3.2). We also define node $j(i)$, which is one of the nearest, feasible neighboring node to i, and is identified using the equation (2.3.3).

We have also shown a pseudo-code in the Algorithm (1) to give an intuition of how it works. The Algorithm (1) continues until at least one of the value is non-zero in the reduction vector $R \in \mathbb{R}^{|V|-1}$. This approach can also be easily generalized for multiple-root CMST problems.

$$\sum_{z \in N(T_i)} f_z \leq K \tag{2.3.1}$$

$$S(i) = \{j \in V \setminus \{0\} : j \notin X_i, (j,i) \in A, |X_i| + |X_j| \leq K\} \tag{2.3.2}$$

$$j(i) = \begin{cases} \text{one of the } j \in \operatorname{argmin}_{j \in V \setminus \{0\}} \{c_{ji} : j \in S(i)\}, & S(i) \neq \emptyset \\ 0, & S(i) = \emptyset \end{cases} \tag{2.3.3}$$

$$R_i = \begin{cases} c_{0i} - \min\{c_{ji} : j \in S(i)\}, & S(i) \neq \emptyset \\ 0, & S(i) = \emptyset \end{cases} \tag{2.3.4}$$

Algorithm 1: Esau Williams

```
Data: G=$(V, A, c, K)$
Result: $M$: set of $|V| - 1$ arcs spanning $G$
$M \leftarrow \emptyset$ ;
for node $i \in V \backslash 0$ do
    $M = M \cup \{(0, i)\}$ ;            // set of arcs in spanning tree
    $X_i = \{i\}$;
    $R_i = .1$ ;                         // any value except 0
while $(\exists i \in V : R_i > 0)$ do
    for node $i \in V$ do
        $j(i)$ = nearest neighbouring node ;       // using (2.3.3)
        compute $R_i$ ;              // Reduction value using (2.3.4)
    find $i^* \in \arg\max_{i \in V} R_i$ ;
    if $R_{i^*} > 0$ then
        $M = M \cup \{(j(i^*), i^*)\}$;
        $M = M \backslash \{(0, i^*)\}$;
        $X_{j(i^*)} = X_{i^*} = X_{i^*} \cup X_{j(i^*)}$;
return $M$;
```

Esau Williams' heuristic (Alg. 1) is a simple greedy heuristic which creates a feasible capacitated minimum spanning tree. It has been proved [Karnaugh, 1976] with an example that a modified version of the well-known minimum spanning tree algorithm called Kruskal's algorithm, with an additional capacity limitation cannot guarantee a better result than Esau-Williams' heuristic [Voss, 2008].

2.3.2 *Convex hull, and Graham Scan*

Definition 2.3.1. The convex hull of a set of points S in n dimensions is the intersection of all convex sets containing S. For N points $p_1, ..., p_N$, the convex hull C is then given by the expression:

$$C = \{\sum_{j=1}^{N} \lambda_j p_j : \lambda_j \geq 0 \text{ for all j and } \sum_{j=1}^{N} \lambda_j = 1\}. \tag{2.3.5}$$

There are numerous algorithms to find the convex hull of a finite set of points in 2 dimensions. We will be using Graham Scan algorithm to find the convex hull of the set of N points in $n = 2$ dimensions. It is an efficient algorithm that has a time complexity of $O(N \cdot logN)$, where N is the number of points and all the points $P_1 \cdots P_N \in \mathbb{R}^2$. We define the points as $P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3), \cdots P_N = (x_N, y_N)$. Following is an overview of the algorithm:

1. Find an anchor point with the smallest y coordinate
2. Sort all the points by comparing angles between the anchor and the point which is being assessed

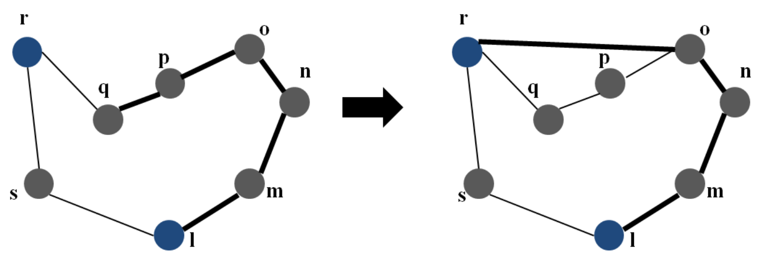

Figure 2.3.1: An illustration of Graham Scan

3. Add each point to the list *H*
4. While assessing each point in the list, we check whether 3 consecutive points are turning left or right . If there is a right turn then the algorithm removes the second to last point in the list H.

A simple example is given in Fig. 2.3.1. We first analyze points p,q,r,. These three points form a right turn, so we remove second last point q from the list H. In the next step, we notice that the last point r also forms a right turn with o and p, so we remove p. Then we analyze r,o and n which turns left, r is added to the list H. Continuing in the same way, we get convex hull of the set of points. We do not mention the pseudo-code here as we are merely interested in the results.

2.3.3 *Local neighborhood structures for the problem*

A heuristic gives us a good, and feasible solution of an optimization problem without guaranteeing optimality. After attaining a feasible solution, we try to reduce the optimality gap or improve the solution quality. In this section, we look into the various improvement methods which improves the solution quality by exploring neighboring candidate solutions.

Once the initial feasible heuristic solution is available, a local search algorithm can be used to change the solution, and reduce the optimality gap. In this step, local neighborhoods, and exploration policies are defined. The performance of any neighborhood search algorithm mainly rely upon the neighborhood structure it exploits. Two of the well known neighborhood structures for the CMST problem are mentioned in [Amberg et al., 1996] and [Sharaiha et al., 1997]. The neighborhood structure in [Amberg et al., 1996] is based on exchanging one node between two sub-trees.

The neighborhood structure defined by [Sharaiha et al., 1997] moves a part of the sub-tree from one sub-tree to another or directly connects it to the root node. Both of these neighborhood structures can be considered as two-exchange neighborhood structure, because they exchange nodes between only two sub-trees. As it is quite evident from their definition, the number of neighborhood solutions in two-exchange neighborhood cannot

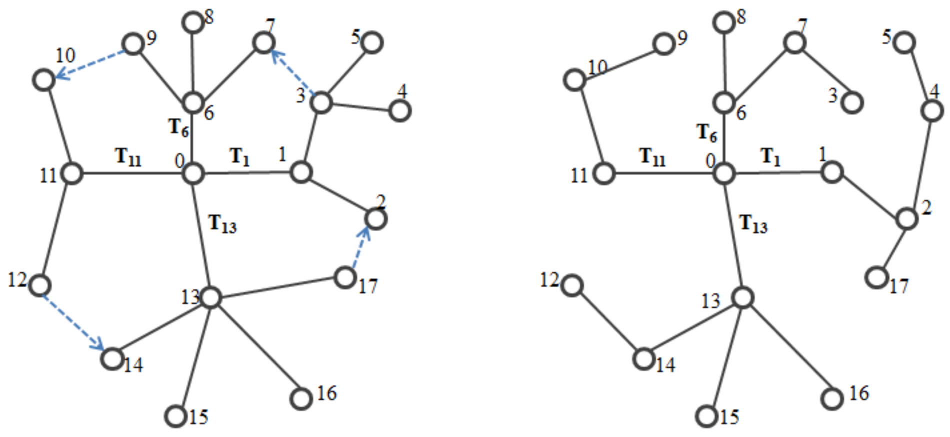

Figure 2.3.2: An illustration of Cyclic Multi-exchange (from left to right)

be more than n^2, where n is the number of non-root nodes.

In article [Ahuja et al., 2001], a multiple node exchange (multi-exchange) neighborhood structure has been proposed. This neighborhood structure allows exchange of nodes spanning multiple sub-trees. There are two types of multi-exchange methods : *cycle* and *path-based* exchanges as depicted in Figures 2.3.2 and 2.3.3. The cardinality of each sub-tree remains the same in cycle-based exchange, but in the path-based multi-exchange, cardinality of one of the subtree increases and one reduces. The arcs on the left side of the Figure 2.3.2 represent node 17 leaving sub-tree T_{13} and entering T_1, whereas node 3 leaves T_1 and enters sub-tree T_6. Similar is the case for the path-based exchange.

The multi-exchange neighborhood structure finds large number of neighbors of a solution. A capacitated minimum spanning tree problem with n nodes and capacity K, may contain as many as $\Omega(K^{\frac{n}{K}}(n-K-1)!)$ solutions [Ahuja et al., 2001]. Therefore, the authors have developed a heuristic based method which involves formation of a directed improvement graph of the solution. The improvement graph maps each multi-exchange to a subtree-disjoint cycle. Thus, every negative sub-tree disjoint cycle in the improvement graph represents a profitable multi-exchange neighbor of the current solution. The computational experiments have proved the superiority of this neighborhood structure as compared to two-exchange neighborhood.

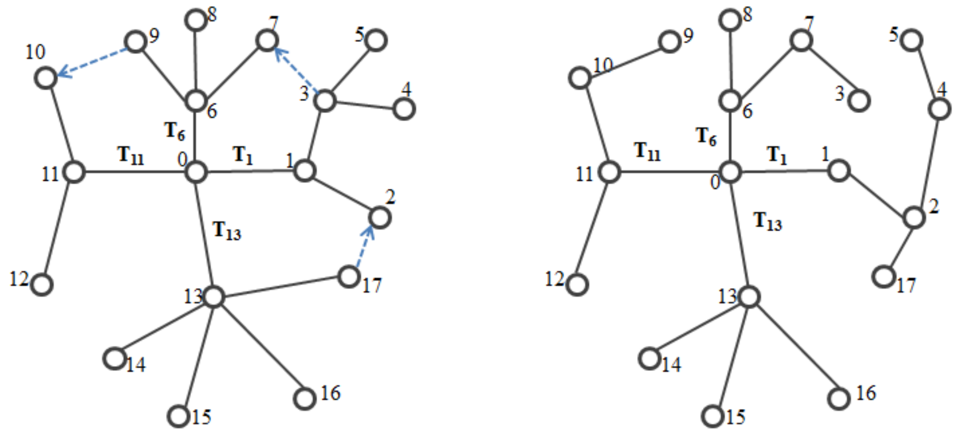

Figure 2.3.3: An illustration of Path Multi-exchange (from left to right)

3

MATHEMATICAL DESCRIPTION

In order to evaluate the results attained from heuristics, it is always a good idea to benchmark heuristic solutions for some of the reference instances with the solutions from a state of the art exact/MILP model. It can be used to find the optimality gap, and benchmark the performance of different heuristics. In this work, we compare our heuristic results with results published in [Klein and Haugland, 2017] for four wind farms Walney 1, Walney 2, Barrow and Sheringham Shoal. We have deliberately chosen this model as it allows branching, parallel cables, and obstacle avoidance. So, we believe that this exact model is an accurate representation of the actual problem. However, different project developers may have different additional restrictions which can be addressed by an ad-hoc fix of the heuristic or adding new constraints to the exact model.

Now, we briefly describe the engineering problem, followed by mathematical description of the problem.

3.1 ENGINEERING DESCRIPTION

We are provided as an input; positions of the turbines, substations and regions of the seabed not accessible to cable installations. We are also given the maximum cable capacity to be used for the cable connections. Multiple turbines are connected to one of the substations in series circuit without any loops. Two or more cables can enter a turbine , but only one cable containing all the power leaves. This is done using switch gears at the turbines. The cost of the cable is given in per unit length, and we assume that the overall cost depends on the total cable length.

The installation of parallel cables between two turbines in a shared trench may reduce the overall cable length. Parallel cables are used as a cost saving measure, as they help in avoiding any potential cable crossing. This point can be understood by looking at the example highlighted in Fig. 3.1.1, where parallel cable are used in (*c*), and the layout is feasible as opposed to (*a*), which is infeasible. This layout is also less costly than (*b*). Therefore, allowing parallel cables gives better results in some cases.

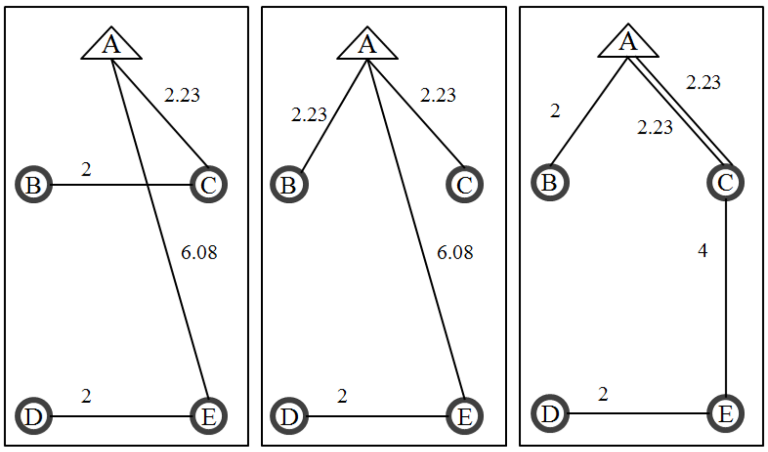

Figure 3.1.1: An illustration of importance of Parallel cables. *a*: Infeasible *b*: cost = 12.54 *c*: cost = 12.46

One of the main constraints to be tackled is node crossing. In Fig. 3.1.2, there is a node crossing at node B . The cable (green) from A is connected to F via B. This cable layout is not feasible according to our cable crossing constraints. An alternate feasible path which can be used is $A-G-F$, where no power is deposited at node G. In order to tackle these issues, some additional optional/steiner nodes are placed in an orbit around each turbine node. So, a connection with a Steiner node located in an orbit of turbine node i means no power is deposited through that cable at node i.

Due to growth in the offshore wind industry, it is expected that many offshore wind farms have location shared with some restricted zones in the sea. Thus, tackling obstacles becomes crucial. Although restricted areas can come in all shapes and sizes, it is safe to assume that we can identify a convex polygon around each convex/non-convex obstacles. Vertices of the convex polygon can be the location of optional nodes, which can be used to avoid these obstacles. These additional nodes are necessary for providing flexibility to the cable path.

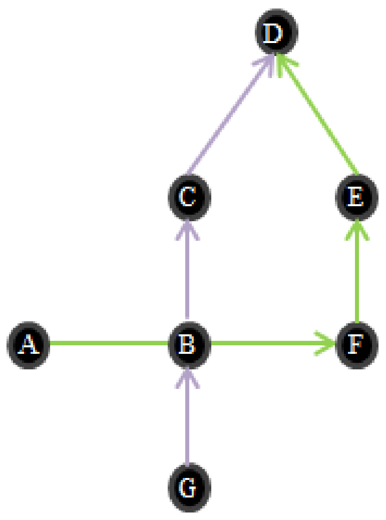

Figure 3.1.2: An illustration of node crossing

3.2 GRAPH DESCRIPTION

As we are dealing with a combinatorial optimization problem, a good place to start is first establishing the graph.

We have a directed graph $G = (V, A)$, with V as the set of nodes in the graph and A is the set of arcs in the graph. The node set V contains T,O and S, the set of turbine nodes, optional nodes and substation nodes respectively ($V = T \cup O \cup S$). The optional nodes are placed near each turbine, and also at the vertices of the convex polygons containing the obstacles. The location of the optional nodes (O) are not part of the problem statement, but used as a design parameter for improving the solution quality of the model. The set of arcs A consists of arcs originating from either T or O and reaching all the nodes in V. There are no arcs originating from the substation nodes in S. Thus, the set of arcs $A = (V \setminus S \times V)$. The set ω consists of all the edge pairs ({i,j},{h,k}) crossing each other, where both the edges {i,j} and {h,k} $\in E$. Here E is the undirected edge set defined as $(\{i, j\} : (i, j) \in A)$. The power produced at each node $i \in V$ is given by M_i. We are assuming that each turbine has same power production, $M_i = 1, \forall i \in T$ and $M_i = 0, \forall i \in O$. The function $P : V \mapsto \mathbb{R}^2$ embeds the nodes in the plane and line segment between i and j, where $i,j \in V$ is embedded by $P(i, j) = \{P(i) + \theta(P(j) - P(i)) : \theta \in [0, 1]\}$. Summary of the notations used in developing MILP are in Table 3.2.1.

Name	Description	Type
V	Vertices in the graph	Set
T	Turbine nodes	Set
O	Optional nodes (around turbines and obstacles)	Set
S	Substation nodes	Set
A	Set of arcs, A= $(T \cup O) \times$ V	Set
$P(i)$	Coordinates of $i \in V$	Parameter
$P(i,j)$	Closed line segment between between $P(i)$ and $P(j)$	Parameter
c_{ij}	Euclidean distance between P(i) and P(j)	Parameter
x_i	x-Coordinates of $i \in V$	Parameter
y_i	y-Coordinates of $i \in V$	Parameter
K	Maximum number of turbines on a cable	Parameter
x_{ij}	If the arc between vertex i and j is formed $x_{ij} = 1$, otherwise 0	Binary variable
y_{ij}	Flow variable between nodes i and j	Continuous variable
E	undirected edge set {{i,j}:(i,j) $\in$ A}	Set
ω	$\{\{i,j\},\{h,j\} \in E \times E : P(i,j) \cap P(h,j) \setminus \{P(i),P(j),P(h),P(k)\} \neq \emptyset\}$	Set
M_i	Power production at each node	Parameter

Table 3.2.1: Notations for Cable Layout Optimization Problem

3.3 MILP MODEL

The model is simplified for ease of understanding and conveying the core ideas behind the cable layout problem. However, for detailed understanding of some of the common MILP models, readers are referred to [Klein and Haugland, 2017], [Fischetti and Pisinger, 2016]
MILP Model:

$$\min \sum_{(i,j)\in A} c_{ij} \cdot x_{ij} \tag{3.3.1}$$

$$\sum_{i\in V: i\neq k, (k,i)\in A, (i,k)\in A} (y_{ki} - y_{ik}) = M_k \qquad \forall k \in T, O \tag{3.3.2}$$

$$K \cdot x_{ij} \geq y_{ij} \qquad \forall (i,j) \in A \tag{3.3.3}$$

$$\sum_{j\in V: j\neq i, (i,j)\in A} x_{ij} = 1 \qquad \forall i \in T \tag{3.3.4}$$

$$\sum_{j\in V: j\neq i, (i,j)\in A} x_{ij} = 0 \qquad \forall i \in S \tag{3.3.5}$$

$$\sum_{j\in V: j\neq i, (i,j)\in A} x_{ij} \leq 1 \qquad \forall i \in O \tag{3.3.6}$$

$$\sum_{i\in V: j\neq i, (i,j)\in A} x_{ij} \leq 1 \qquad \forall j \in O \tag{3.3.7}$$

$$x_{ij} + x_{ji} + x_{hk} + x_{kh} \leq 1 \qquad (\{i,j\},\{h,k\}) \in \omega \tag{3.3.8}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{3.3.9}$$

$$y_{ij} \geq 0 \qquad \forall (i,j) \in A \tag{3.3.10}$$

The objective (3.3.1) minimizes the cable length. The constraint (3.3.2) are used to balance the flow at each turbine, and optional nodes. It is not a valid constraint for the substations as they absorb all the power entering it. The constraint (3.3.3) is necessary to avoid flow through any arc more than its capacity. The flow variable y_{ij} is used in the model to prevent formation of any sub-tours and ensure connectivity. The continuous variable y_{ij} is the amount of power flowing in the cable between i and j. The binary variable x_{ij} is 1, if the arc (i,j) is in the final tree solution. The constraints (3.3.4), (3.3.5) and (3.3.6) all put constraints on the number of outgoing arcs from turbines, substations and optional nodes respectively. There must be one arc leaving each turbine, no arcs leaving substations and at most one arc can leave optional nodes. There is an additional limitation on the in-degree of the optional nodes as mentioned in the constraint (3.3.7). The constraint (3.3.8) refers to the non-crossing constraint.

One could choose to add more restrictions in the MILP model by limiting the in-degree of each turbine or substations. However, we restrict our description to the above mentioned simple MILP model for the cable layout problem. One of the drawbacks of the MILP model mentioned above is equation (3.3.8), as it is weak and its number can be very large $O(|V|^4)$.

4 GREEDY HEURISTIC AND MODIFICATIONS

In this chapter, we present a modified version of Esau Williams' algorithm (Alg. 1). We call the new heuristic presented in this chapter as Obstacle-Aware Esau Williams heuristic (or Obstacle-Aware). As discussed previously, MILP exact models exist, and provide optimal solutions in few hours for smaller instances, but they are unable to solve larger instances in a reasonable time frame. There is a need to develop a fast heuristic revealing good, and feasible solutions for instances with large number of turbines.

4.1 PRELIMINARIES

The abstract problem definition and input graph to the heuristic remain similar to the MILP model discussed in previous section 3.2.1. The only difference is in the set of optional nodes O. The set O now contains only the extreme points of the convex hull of the obstacles, and optional nodes near turbines are not included. Unlike the model in [Klein and Haugland, 2017], we have not used optional nodes around each turbine. This is done to reduce the computational efforts. We have used another method to achieve the same flexibility. The arc set A has all the arcs directed from $T \cup O$ to all the other nodes V. Therefore, we define $A = (T \cup O) \times V$, where $V = T \cup O \cup S$.

The ambition of the heuristic is to first find a set Z of rooted trees spanning all the turbine nodes $i \in T$. The trees are rooted at a turbine node having least distance to one of the substations among all the turbine nodes in their respective tree. We partition the nodes in each rooted tree $\lambda \in Z$ into two sets, the set a_λ of active nodes and set p_λ of passive nodes. Both active (a_λ) and passive node (p_λ) sets are defined with respect to the rooted tree λ. We define $N(\lambda) \subseteq V \setminus S$, as the set of nodes in the rooted tree λ. We use $LR(\lambda) \subseteq N(\lambda)$, to denote the set consisting of all the leaf nodes and the root of the tree λ. All the rooted trees $\lambda \in Z$, are eventually connected to one of the substations such that there is a unique path from each turbine node to one of the substations. Now, we define the active and passive nodes formally:

Definition 4.1.1. A node $i \in T$ in the rooted tree λ is an active node with respect to λ, if it receives power from other nodes $j \in N(\lambda)$, that is, flow variable $y_{ji} > 0$, or it sends power to some other node $j \in N(\lambda)$, that is,

$y_{ij} > 0$. The set of active nodes of the rooted tree λ is denoted as $a_\lambda \subseteq T$. The node set $N(\lambda) \setminus a_\lambda$ is referred to as the passive node set (p_λ) of rooted tree λ.

It follows from the Definition 4.1.1, that for any rooted tree $\lambda \in Z$, we have

$$a_\lambda = \left\{ i \in N(\lambda) \cap T : | \sum_{j \in N(\lambda):(i,j) \in A} y_{ij} - \sum_{k \in N(\lambda):(k,i) \in A} y_{ki} \quad | = 1 \right\}. \tag{4.1.1}$$

We have used continuous flow variables y_{ij}, where $(i,j) \in A$. These variables have been discussed in previous Section 3.2.1. We define $A(\lambda) \subset A$, as the set of arcs in the rooted tree λ. Any feasible set (Z) of rooted trees spanning all the turbine nodes T must be both *arc crossing free* and *node crossing free*. These concepts are described next:

Definition 4.1.2. If all the arc pairs $(i,j) \in A(\lambda), (h,k) \in A(\lambda'); \lambda, \lambda' \in Z$, where $\lambda \neq \lambda'$, satisfy $\{\{i,j\},\{h,k\}\} \notin \omega$ (Table 3.2.1), then Z is said to be an *arc crossing free* set of rooted trees.

Arc crossing free is a necessary, but not sufficient condition to attain a feasible layout. We also need to prevent node crossing (see: Fig. 4.1.1) and arcs from entering in restricted areas. We present an observation which forms the basis of our definition of node crossing. We define a set D of all the common nodes present in at least two distinct rooted trees. The nodes in the set are neither leaf nodes nor root node in their respective tree. We call this set *common node set*, formally given as

$$D = \left\{ i \in T \cup O : i \in \big(N(\lambda_x) \setminus LR(\lambda_x)\big) \cap \big(N(\lambda_y) \setminus LR(\lambda_y)\big), x \neq y, \lambda_x, \lambda_y \in Z \right\}. \tag{4.1.2}$$

Observation 4.1.1. We assume that $\vec{u}$ and $\vec{v}$ are two vectors in the Cartesian plane. We know that there is a unique parallelogram having $\vec{v}$ and $\vec{w}$ as its two sides. The area of the parallelogram is given by $|\det[\vec{v}\ \vec{w}]|$. The sign of the determinant indicates the orientation of $\vec{w}$ with respect to $\vec{v}$. The vector $\vec{w}$ is anticlockwise to $\vec{v}$ if and only if the determinant is positive, and $\vec{w}$ is clockwise if and only if the determinant is less than zero. The two vectors are linearly dependent if the determinant is zero.

We make use of Observation 4.1.1, and common node set D (4.1.2) to identify node crossing in the set of rooted trees Z. Now, we define a set of nodes $F(Z) \subseteq D$, which contains all the nodes violating the node crossing constraint in the set of rooted trees Z. We use function P to embed nodes in the plane $\mathbf{P} : i \mapsto \mathbb{R}^2$, where $i \in V$. We represent $P(a) - P(b)$ with $\vec{ba}$, where $b, a \in V$. We define a set $F(Z)$ with respect to the set of rooted trees Z, containing all the nodes where node crossing takes place as:

$$\begin{aligned} F(Z) = \{ d \in D : (\det[\vec{kd} \quad \vec{ki}] \times \det[\vec{dl} \quad \vec{dj}]) < 0; (k,d),(d,l) \in A(\lambda_m), \\ (i,d),(d,j) \in A(\lambda_n), m \neq n, \lambda_m, \lambda_n \in Z \}. \end{aligned} \tag{4.1.3}$$

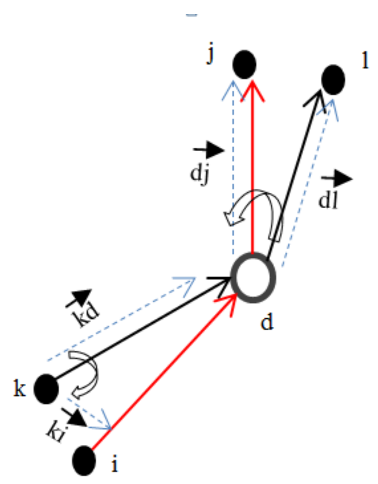

Figure 4.1.1: An illustration of Node Crossing at Node d. (Bold lines in red and black represent two cables which cross at node d.)

This is highlighted in the Fig. 4.1.1, the node crossing takes place at node d. The two parallelogram formed by $\vec{kd}$ with $\vec{kl}$, and $\vec{dl}$ with $\vec{dj}$ have opposite orientation. Thus, the node d gets added to the set $F(Z)$ mentioned in equation 4.1.3.

Definition 4.1.3. If all the arc pairs $(i, j) \in A(\lambda), (h, k) \in A(\lambda'); \lambda, \lambda' \in Z$,, where $\lambda \neq \lambda'$, satisfy $\{\{i, j\}, \{h, k\}\} \notin \omega$, and $F(Z) = \emptyset$, then Z is said to be a *crossing free (both node crossing and arc crossing free)* set of rooted trees

Following is the summary of the conditions required to be satisfied to attain a feasible cable layout:

1. Each rooted tree λ contains at most K active nodes, that is, $|a_\lambda| \leq K$, where K is the cable capacity
2. The set Z is *crossing free* and none of the arcs in $\lambda \in Z$ enter the convex hull of the obstacles
3. $N(\lambda) \cap S = \emptyset$, for each rooted tree λ in Z.
4. $\bigcup_{\lambda \in Z} a_\lambda = T$ and $a_\lambda \cap a_{\lambda'} = \emptyset$, for $\lambda \neq \lambda'$
5. Each rooted tree $\lambda \in Z$ is connected to S by exactly one path between nodes i and s, where $i \in a_\lambda$ and $s \in S$. Thus, a feasible set of trees spanning $T \cup S$ is obtained. (We do not use the term Forest, since there is a possibility that the trees are not disjoint)

4.2 OBSTACLE-AWARE ESAU WILLIAMS HEURISTIC

We had discussed Esau Williams' heuristic in section 2.3.1, which finds good, and feasible solutions to the CMST problem. We have now estab-

Nodes	Coordinates	Reduction(1st Iteration)	Reduction(2nd Iteration)
B	(-2,-1)	2.236 - 1.414 = 0.822	0.822
C	(-1,-2)	2.36 -1.414 = 0.822	0.822
D	(1,-1)	1.414-2.236 = -.822	0
E	(-1,-3)	3.162-1 = 2.162	2.162
F	(-2,-3)	3.6-1 = 2.6	2.6
G	(-3,-3)	4.24-1 = 3.24	0

Table 4.2.1: Sample calculations in Esau-Williams heuristic

lished additional constraints, such as node and arc crossing to be incorporated to the CMST problem. We discuss a simple example to give an intuition about need for modification of Esau-Williams' heuristic and later on, we present the modifications to this heuristic.

4.2.1 *Sample Calculations*

The reduction value mentioned in equation (2.3.4) is calculated for each turbine node, reflecting the potential savings achieved by removing the central arc $(i,s) \in A$, where $s \in S$. We start with a feasible star layout, where every turbine node is connected to one of the closest substations. In every iteration, the central arc to one of the substations is removed from a node i^* having the maximum reduction value. The node i^* is connected with a nearest neighboring node $j(i^*)$. An example is illustrated in Fig. 4.2.1. The capacity limitation of each cable in this example is 3. In Fig. 4.2.1, from a to c, overall cost is reduced. The calculations for two iterations are shown in Table 4.2.1. In the first iteration, node *G* has maximum reduction value, so central arc *(G,A)* is removed and node *G* joins node *F* using a direct arc *(G,F)*. Once the central arc of a node is removed, its reduction value is set to 0. In the final iteration, an infeasible layout is achieved (Fig. 4.2.1.c). Thus, Esau-Williams' heuristic does not guarantee a feasible solution to the offshore cable layout problem.

4.2.2 *Key challenges*

Esau Williams' reduction function is used to partition the turbines nodes in active node disjoint rooted trees. We develop some procedures to make sure that the set of rooted trees are crossing free. Therefore, instead of adding direct arcs between turbine nodes, we use paths which do not cross any existing arcs or enter any restricted areas. Following are the key challenges addressed by our heuristic:

1. Identifying arc or node crossing if it exists
2. Preventing cables from entering restricted areas in the sea bed
3. Finding shortest paths while avoiding obstacles

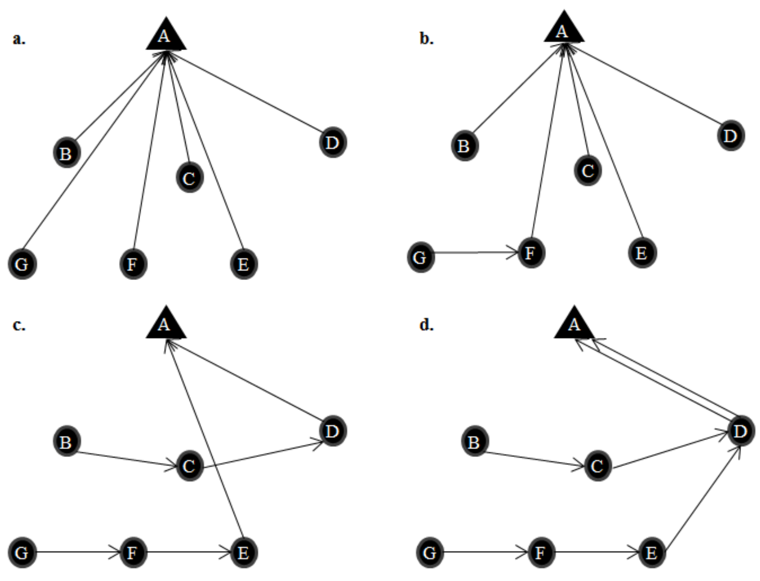

Figure 4.2.1: An illustration of steps in Esau-Williams' heuristic. a. Star layout b. First iteration c. Final iteration(infeasible) d. Feasible layout

4.3 A PROCEDURE TO PREVENT CROSSING

It is shown in example 4.2.1, that in each iteration two nodes are joined from two trees. In the cable layout problem, it is required to check for obstacles, and crossing when joining two turbine nodes. There are two parts to solving this challenge 1: finding node and arc crossing, and 2: finding shortest paths around obstacles.

Now, we discuss some important notations which are used while explaining the algorithm. A closed line segments $s = [P(i), P(j)]$ in the plane is defined with respect to two nodes, i and j. The problem we address is whether there exists an intersection among a collection of line segments and the line segment s, where arc (i, j) is being assessed to be added to the solution. We have made use of a standard *plane sweep* based algorithm to identify the line segment intersections. In Fig. 4.3.1, we have highlighted some cases considered as *intersection* as *True*. We do not allow case (v) in Fig. 4.3.1, as it may lead to node crossing later on, as depicted in the figure to the right of case (v).

Let us call B as the *Obstacle Set*, containing distinct convex obstacles in the sea bed. These convex obstacles are defined as the set of extreme points. We define a function $\mathbf{f}: O \mapsto B$, where O, is the set of all the obstacle nodes. The function maps each obstacle node $(o \in O)$ to a unique convex obstacle in *Obstacle Set* (B), such that $f(o)$ is the obstacle of which $P(o)$ is an extreme point. Each convex obstacle $b \in B$, represents a distinct restricted area in the sea bed. The set Q_b stores the location of extreme

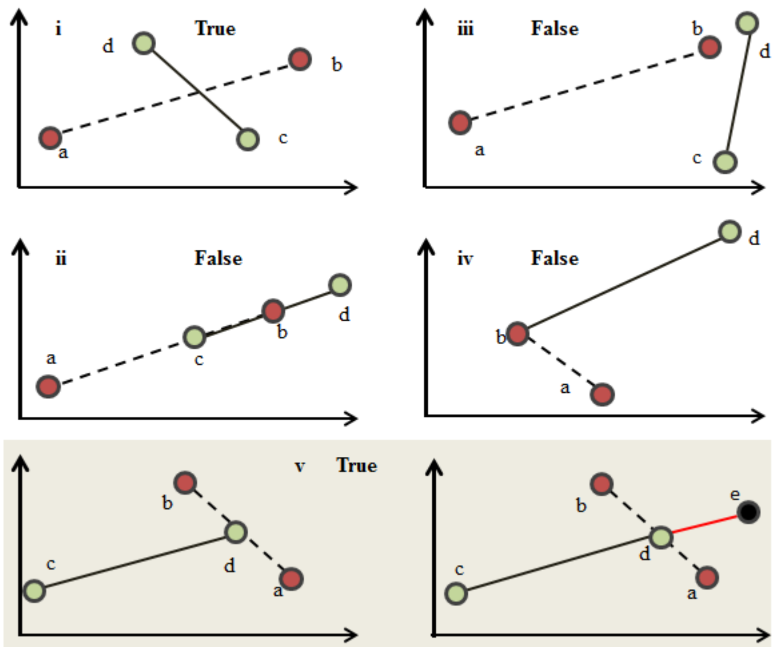

Figure 4.3.1: An illustration of intersections

points of the obstacle $b \in B$.

Now, we formally define a set C of line segments called *Obstacle Lines*. The Obstacle Lines (C) is a set of line segments between each pair of adjacent extreme points in Q_b, where $b \in B$. These sets are defined as follows:

$$Q_b = \{P(o) : o \in O, f(o) = b\} \subset \mathbb{R}^2 \tag{4.3.1}$$

$$C_b = \{[P(i), P(j)] : P(i), P(j) \in Q_b, i \neq j\} \tag{4.3.2}$$

$$C = \bigcup_{b \in B} C_b. \tag{4.3.3}$$

We also define a *Solution Set* (M), which is a set of arcs in the final solution, and results in creating unique paths from each turbine node to one of the substations. A line segment corresponding to each arc in the set M is stored in a set called *Turbine Lines* (H) defined as:

$$H = \{[P(i), P(j)] : (i, j) \in M\}. \tag{4.3.4}$$

We have given a summary of the important sets in Table. 4.3.1.

4.3.1 *Shortest feasible paths*

In graph theory, a path represents a finite sequence of edges. A directed simple path is defined in a similar way, but there are additional constraints; all the directed edges are in the same direction and there is no repetition

Set	Name	Brief Description
B	*Obstacle Set*	Set of obstacles
C	*Obstacle Lines*	Line segments between extreme points of obstacles
M	*Solution Set*	Arcs in the solution
H	*Turbine Lines*	Line segments corresponding to the arcs in the solution

Table 4.3.1: Explanation of the Sets

of nodes. We want to identify a *crossing free* path. This path can be represented as an ordered set of arcs between source node i and sink node j referred as $NC^{i,j}$. None of the arcs in the set $NC^{i,j}$ intersects with any of the line segments in *Turbine Lines* (H) (eq: 4.3.4) and *Obstacle Lines* (C) (eq: 4.3.3). Each node is reachable from itself, $NC^{i,i} = \{(i,i)\}$. We use the set of crossing edge pairs ω mentioned in Table 3.2.1 to define the set $NC^{i,j}$ as:

$$NC^{i,j} = \{(x,y) \in A : \{\{x,y\},\{k,l\}\} \notin \omega, [P(k),P(l)] \in H \cup C, NC^{i,x} \neq \emptyset, NC^{y,j} \neq \emptyset\}. \tag{4.3.5}$$

Pathfinding or pathing is the method of finding a sequence of edges between two points. There are standard algorithms that find a shortest path on a weighted graph. We have used Dijkstra's algorithm to find a shortest path between two nodes. However, the shortest path obtained from Dijkstra's algorithm does not guarantee a crossing free path. We deal with this challenge by checking each arc (m,n) of the shortest path P for intersections with line segments in sets Obstacle Lines (C) and Turbine Lines (H). The succeeding steps after identification of an intersection in the current shortest path P, depends on whether the intersection is with $[i,j] \in H$ or $[i,j] \in C$. In the latter case, we change the cost of an intersecting direct arc (m,n) to a large value referred to as *bigM* (which can be assigned any large value such as cost of a feasible star layout). This is because $\{\{m,n\},\{i,j\}\} \in \omega$, and thus, (m,n) is not allowed in Solution Set (M).

In the next run of Dijkstra's algorithm, we identify a new path for assessment. We use recursive calls to procedure *Crossing* on each edge of the current shortest path (see Alg. 2). In Fig. 4.3.2, we have highlighted a case where a crossing free path between nodes 1 and 2 is to be found. A corresponding recursion diagram of procedure *Crossing* is highlighted in Fig. 4.3.3. The direct arc (1,2) results in a crossing with line segments in Obstacle Lines (C), therefore it makes recursive calls to procedure Crossing.

In the other case, if the intersection is with line segments from set Turbine Lines (H), use of Dijkstra's algorithm on the modified graph may lead to node crossing. This is highlighted in Fig. (4.3.4). In this figure, the black solid nodes are turbine nodes, and the others are optional nodes. It is evident from Fig (4.3.4.a) that using Dijkstra's algorithm in such cases may lead to node crossing. A pseudocode of this procedure is presented in Alg. 2.

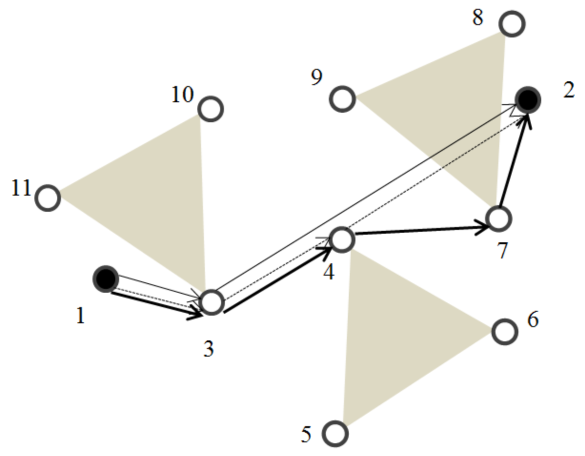

Figure 4.3.2: An example of a crossing free path between turbine nodes 1 and 2 (NC^{12}) (Bold line).

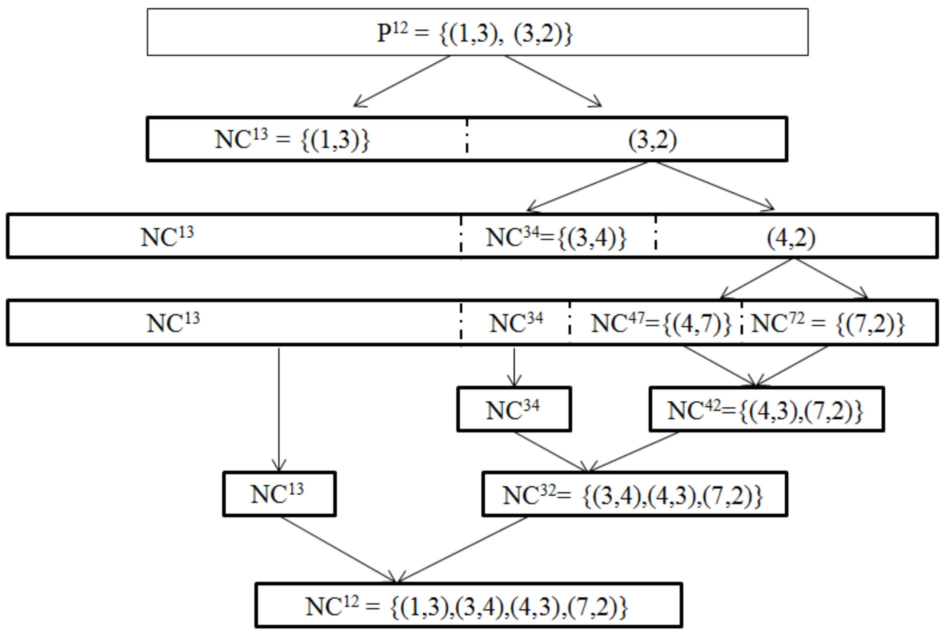

Figure 4.3.3: Recursion diagram for *Crossing* used to find a crossing free path between nodes 1 and 2 (see:Fig. 4.3.2)

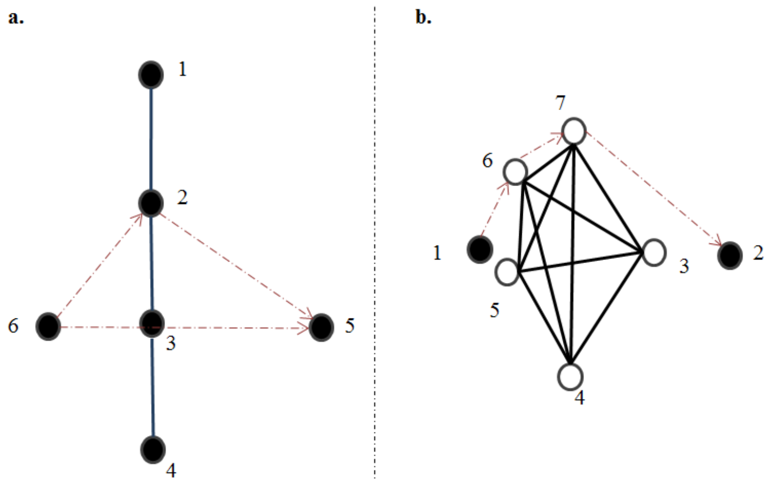

Figure 4.3.4: **a**. Intersection with line segments from set Turbine Lines (H), **b**. Intersections with line segments from set Obstacle Lines (C)

Algorithm 2: Crossing

```
Data: $G = (V, A)$, Turbine Lines($H$), Obstacle Lines($C$), $(i^*, j^*)$, function
      embedding nodes in the plane ($\mathbf{P} : i \mapsto \mathbb{R}^2, i \in V$)
Result: True: No crossing free path between nodes $i^*$ and $j^*$, False: A
        crossing free path has been found between the two nodes and
        $NC^{i^*,j^*}$ is updated with it
begin
    if $[P(i^*),P(j^*)]$ intersects $H$ then
        return True;
    else if ($[P(i^*),P(j^*)]$ intersects $C$) then
        change cost of the edge $(i^*, j^*)$ to large value;
        run Dijkstra's algorithm on $G$ from source node $i^*$;
        $P^{i^*,j^*}$ = shortest path from $i^*$ to $j^*$;
        initiate a boolean array called check;
        x=0;
        for each arc $(i, j)$ in $P^{i^*,j^*}$ do
            $check[x++] = Crossing(G, H, C, (i, j))$;
        if ($check[1] \vee check[2] \cdots \vee check[x]$) then
            return True;
        else
            for (each arc $(i, j) \in P^{i^*,j^*}$ ) do
                replace $(i, j)$ in $P^{i^*,j^*}$ with crossing free path $NC^{i,j}$;
            $NC^{i^*,j^*} = P^{i^*,j^*}$;
            return False;
    else
        $NC^{i^*,j^*} = \{(i^*, j^*)\}$;
        return False;
```

4.4 MAIN ALGORITHM

The heuristic being developed is referred to as *Obstacle-Aware Esau Williams*. We develop an intuition for the heuristic using the pseudocode in Alg. 3. The algorithm has two main parts: formation of rooted tree $\lambda \in Z$, where Z is a set of rooted trees spanning T. These rooted trees are crossing free as per our definition 4.1.3. The second task of the algorithm is identifying paths from each rooted tree λ to one of the substations $s \in S$. This results in a set of trees spanning nodes $T \cup S$.

We have used as inputs, previously defined graph G (section 4.1), a cost array c, and a set Obstacle Lines (C) (eq:4.3.3). The parameter K is the maximum cable capacity. The array c initially contains the cost of direct edges between each pairs of nodes in the graph. It is set as the Euclidean distance between the two nodes. During the run of the algorithm, we *join* two nodes using a crossing free path. The *joining* of two nodes, should be preceded by checking the number of active nodes post joining of two trees. It must be less than or equal to K. We have used a boolean variable *Edge-CostChanged* to keep track of whether any of the cost of edges between nodes have changed, and in case of change, the reduction vector is recalculated.

Procedure *PathsToSubstations* mentioned at the end of Obstacle-Aware Esau Williams' algorithm (Alg. 3) returns a set of crossing free directed paths from each rooted tree to one of the substations. In some cases, it is not able to find such a set of feasible paths, and thus, returns an empty set. In such a case, algorithm can still return a crossing free solution which is a *Star layout*.

Algorithm 3: Obstacle Aware Esau Williams

```
Data: $G = (V = T \cup O \cup S, A)$, cost array (c),cable capacity (K), *Obstacle
      Lines* (C), function embedding nodes in the plane ($\mathbf{P} : i \mapsto \mathbb{R}^2$,
      $i \in V$)
Result: *Solution Set*(M): set of *crossing free* arcs spanning $T \cup S$, such
        that there exists unique path from each turbine node to a
        substation using arcs in $M$
begin
    $H = \emptyset, M = \emptyset, NC^{ij} = \emptyset, \forall (i, j) \in A$;
    initiate R;
    while ($\exists i \in T : R_i > 0$) do
        $EdgeCostChanged = True$;
        while (EdgeCostChanged) do
            find $j(i) \forall i \in T$ ;                          // equation (2.3.3)
            compute $R_i \forall i \in T$ ;                        // equation (2.3.4)
            $i_0 = i^* \in \arg\max_{i \in T} R_i$ ;
            $i_n = j(i_0)$;
            if Crossing(G,H,C,$(i_0, i_n)$) then
                $c_{i_0,i_n}$ = bigM;
                $EdgeCostChanged = True$ ;
            else
                $NC^{i_0,i_n}$ = new path obtained from procedure *Crossing*;
                if $NC^{i_0,i_n}$ *is same as before* then
                    join $i_0$ and $i_n$;
                    $M = M \cup NC^{i_0,i_n}$;
                    $H = H \cup \{[P(i), P(j)] : (i, j) \in NC^{i_0,i_n}\}$;
                    $EdgeCostChanged = False$;
                else
                    if ($|NC^{i_0,i_n}| > 1$) then
                        $c_{i_0,i_n} = \sum_{(i,j) \in NC^{i_0,i_n}} c_{i,j}$;
                        $EdgeCostChanged = True$;
                    else
                        join $i_0$ and $i_n$;
                        $M = M \cup NC^{i_0,i_n}$;
                        $H = H \cup \{[P(i_0), P(i_n)]\}$;
                        $EdgeCostChanged = False$;
    $CP = PathsToSubstations(G, C, M)$ ;                       // Alg. 4
    if ($CP == \emptyset$) then
        return Star Layout;
    else
        Add each arc in the set of paths $CP$ to *Solution Set*($M$);
        return $M$;
```

4.5 PROCEDURE: PATHS TO SUBSTATIONS

The formation of a set of rooted trees spanning T in the first part of Alg. 3, is followed by connecting these rooted trees to one of the substations using crossing free paths. We store these paths in a set *Central Paths* (CP). We highlight the details of procedure *PathsToSubstations* in Alg. 4. The inputs to the procedure are graph G, *Obstacle Lines* (C) (4.3.3), and a set of arcs in *Solution Set* M which is obtained from the first part of Alg. 3. The main idea is to identify an active node ($i \in a_{\lambda}$), from each rooted tree $\lambda \in Z$, where Z is the set of rooted trees having arcs in M, and connecting $i \in a_{\lambda}$, to one of the substations.

We begin by collecting the locations of nodes in each rooted tree $\lambda \in Z$, and storing them in a set $X^{\lambda} \subset \mathbb{R}^2$. It is followed by finding the extreme points of the convex hull of points in X^{λ}. This is done using an algorithm *Graham Scan* (sec: 2.3.2). Thus, we identify a set E, consisting of sets of extreme points corresponding to each rooted tree $\lambda \in Z$. Then, we add line segments between each pair of extreme points to set *Obstacle Lines* (C). This results in making each rooted tree a convex obstacle. This is followed by identifying crossing free paths using the procedure *Crossing* (Alg. 2). In the end, we have a set *Central Paths* (CP), storing crossing free paths from rooted trees to substations. The algorithm is not capable of searching for a crossing free path in cases where all the active nodes of a rooted tree are inside the convex hulls of other rooted trees.

An illustration in Fig. (4.5.1.a), highlights such a case which cannot be handled by the procedure *PathsToSubstations*. However, our computational experiments highlight that due to the greedy nature of the algorithm, such cases rarely occur unless we drastically change the reduction function. This reflects that our heuristic only partially explores the solution space. We have used a standard convex hull inclusion test in the procedure *Inclusion*.

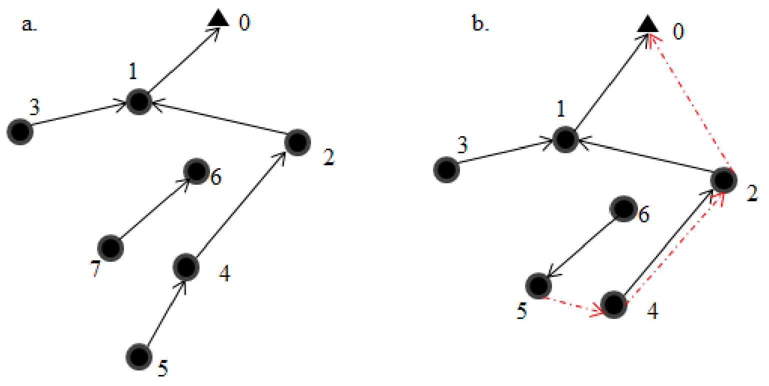

Figure 4.5.1: Possible layouts handled by the procedure *PathsToSubstations* **a**. Not possible to handle, **b**. Possible to handle

It checks whether a point $P(i) \in \mathbb{R}^2$, where $i \in a_{\lambda_x}$, is inside the convex hull of any other rooted tree λ_y, where $\lambda_y \neq \lambda_x$.

Algorithm 4: PathsToSubstations

```
Data: $G = (V = T \cup O \cup S, A)$, Obstacle Lines $(C)$, set of arcs from Alg. 3 $(M)$
Result: $CP$ : a set of directed paths connecting rooted tree $(\lambda)$ to
        substations $S$ by exactly one path in $CP$ or returns an empty
        set if at least one such feasible path cannot be found.
begin
    $Z$ = set of rooted trees formed using arcs in $M$, and spanning $T$;
    $E = \emptyset$;
    for (each rooted tree $\lambda \in Z$) do
        $X^\lambda = \{P(i) : i \in T \cup O, i \in N(\lambda)\} \subset \mathbb{R}^2$;
        $F^\lambda = \{x \in \mathbb{R}^2 : x \in extreme(conv(X^\lambda))\} \subseteq X^\lambda$ ; // Graham Scan
        $E = E \cup \{F^\lambda\}$;
        $C = C \cup \{[P(i), P(j)] : P(i), P(j) \in F^\lambda, i, j \in T \cup O\}$
    for (each rooted tree $\lambda \in Z$) do
        initiate $min = bigM$;
        for (each active node $i \in a_\lambda : P(i) \in F^\lambda$) do
            if ($\neg$ Inclusion($P(i)$,$E \setminus \{F^\lambda\}$)) then
                $J = \{[P(i), P(j)] : (i, j) \in A(\lambda)\}$;
                for each substation $s \in S$ do
                    if Crossing(G,J,C,$(i, s)$)) then
                        $c_{is} = bigM$;
                    else
                        $NC^{is}$ = path from procedure Crossing;
                        $c_{is} = \sum_{(a,b) \in NC^{is}} c_{ab}$;
                        if $c_{is} < min$ then
                            $min = c_{is}$;
                            remove the current best path from $CP$;
                            $CP = CP \cup \{NC^{i,s}\}$;
        if all active nodes in $a_\lambda$ are in the convex hull of other rooted
          trees then
            return $CP = \emptyset$;
    return $CP$;
```

4.6 ANALYSIS OF THE RUNNING TIME

The standard way to express the running time of an algorithm is by counting the number of arithmetic operations, as a function of the size of the input (n). Instead of reporting the lower order terms and coefficients, we usually represent the running time in big-oh notation represented with $O(.)$.

We divide the Obstacle-Aware Algorithm (Alg. 3) into two independent parts. The first part is the one before procedure *PathsToSubstations*. This part returns a set of rooted trees spanning the set T of turbine nodes. The second part contains a procedure called *PathsToSubstations* (Alg.4). We analyze the running time separately, and use the higher order term to give an upper bound on the running time.

4.6.1 *Worst Case Analysis of the First Part*

We approach the task of finding the running time of the first part by identifying an upper bound on the work done to analyze each of the possible crossing free connections between turbines. This is done because a turbine node may be assessed $O(|T|)$ times before a crossing free path to one of the neighboring node is found. There are $O(|T|^2)$ possible connections to be analyzed. Each connection between two turbines is a combination of three cases.

In the first case, a direct arc joins the two turbine nodes, and we have to run *Crossing* only once. There is no intersection, so, we just check the intersection of line segment $[P(i), P(j)]$ with line segments in Obstacle Lines (C) and Turbine Lines (H). A standard plane sweep algorithm takes $O(n \cdot logn)$ time, where n is the number of line segments. In our case, this would amount to $O((|C| + |T|) \cdot log(|C| + |T|))$ time. We have used $|T|$ as the upper bound on the cardinality of set H.

In the second case, the joining of nodes i and j results in an intersection with one of the line segments in set Obstacle Lines (C). Thus, for every intersection with a line segment in C, we temporarily increase the cost of the intersecting edge to a large value $bigM$ and run Dijkstra's algorithm. The algorithm has a running time of $O(E \cdot logV)$, where E is the number of edges, and V is the number of nodes in the graph. In our case, this amounts to $((|T| + |O|)^2 \cdot log(|T| + |O|))$ time.

In the third case, the line segment corresponding to arc (i, j) intersects with a line segment from H. Therefore, we set the cost $c_{i,j} = bigM$, thus, the node connection between i and j is never processed again. This is because the cost of joining nodes i and j is $bigM$. Thus, node j is never the closest node to node i. Thus, the running time for the third case is $O((|C| + |T|) \cdot log(|C| + |T|))$

We combine all these three cases, and assign α, β, and γ to the number of times, we encounter case 1, case 2 and case 3 respectively for a connection between two nodes. By definition, α, β, γ are non-negative integers, where $\alpha \leq 1$, $\beta \leq |C|$, and $\gamma \leq 1$. The important point to note is that if $\alpha = 1$, which means a direct path is found, then both $\beta = \gamma = 0$. Whereas, if $\gamma = 1$, then $\alpha = 0$. We denote by $\mu = O((|T| + |C|) \cdot log(|T| + |C|))$, to represent the running time of the plane sweep algorithm. We define the running time for the first part as $T_1(|T|, \mu, |O|)$ or T_1. The running time for the first part of the algorithm is given as:

$$T_1 \in |T|^2 \cdot \left(\underbrace{\alpha \cdot \mu}_{\text{case 1}} + \underbrace{\beta \cdot \mu \cdot (|T| + |O|)^2 log(|T| + |O|)}_{\text{case 2}} + \underbrace{\gamma \cdot \mu}_{\text{case 3}} \right). \tag{4.6.1}$$

$$T_1 \in \begin{cases} O(|T|^2 \cdot \mu), & \text{if } \alpha = 1 \implies \beta = \gamma = 0 \\ O(|T|^2 \cdot \mu), & \text{if } \beta = 0 \text{ (no obstacles)} \\ O(|T|^5 \cdot |C| \cdot log(|T| + |C|) \cdot log(|T| + |O|)), & \text{otherwise.} \end{cases} \tag{4.6.2}$$

In equation (4.6.2), we have highlighted three key scenarios, and their running times. The first scenario is the best case, as there is neither an intersection with convex obstacles nor with the existing arcs in the solution. In the second scenario, there are no obstacles in the graph, therefore, $\beta = 0$. The last scenario is considered as the worst case. In this case, there are intersections with line segments from Obstacle Lines (C), and arcs in the solution (H). In the worst case, we can assume that $\beta = |C|$, which means that a connection between two nodes leads to intersection with every line segment in Obstacle Lines (C) once. Thus, further simplification, and expansion leads to a running time for the first part as $O(|T|^5 \cdot |C| \cdot log(|T| + |C|) \cdot log(|T| + |O|))$ with obstacles, and $O(|T|^3 \cdot log(|T|))$ in instances without obstacles, where $\beta = |C| = 0$.

4.6.2 *Worst Case Analysis of the Second Part*

The second part consists of procedure *PathsToSubstations* (Alg.4). There are three main components in this procedure. The first component is to identify convex hull of each rooted tree using Graham Scan algorithm, second is the use of convex hull inclusion test to identify whether a node is inside the convex hull of other rooted trees, and lastly, to identify a crossing free path from each turbine node to each substation. The last component is quite similar to the first part mentioned in the previous section, with the only different that we assess $O(|T| \cdot |S|)$ possible connections as opposed to $O(|T|^2)$. We provide the explanation of running time (T_2) in next paragraph. The formula for T_2 is given as:

$$T_2 \in \underbrace{O(|T| \cdot logK)}_{1} + \underbrace{O(|T|^2 \cdot log|K|)}_{2} + \underbrace{O(|T|^3 \cdot |S| \cdot log(|C| + |T|) \cdot |V|^2 log|V|)}_{3}. \tag{4.6.3}$$

As the third term dominates the first two, we get:

$$T_2 \in O(|T|^3 \cdot |S| \cdot log(|C| + |T|) \cdot |V|^2 log|V|). \qquad (4.6.4)$$

Some computations must be done for each tree. The running time is at best linear in the number of nodes in the tree. Then, in the worst case, the turbines are partitioned into fewest possible trees. No tree has more than K turbines. To do a worst-case analysis, assume there are $\lceil \frac{|T|}{K} \rceil$ trees with K turbines each. Therefore, we use these assumptions and our knowledge of running time of Graham Scan ($O(n \cdot logn)$), and inclusion test ($O(n \cdot logn)$) to come up with the expressions for first, and second component in equation (4.6.3).

However, we focus on third, and the dominating term. The amount of work done to identify crossing free path between each turbine node, and substation. This is done to find a least cost feasible path to one of the substations from each tree. Thus, there are $O(|T| \cdot |S|)$ possible connections to be analyzed. The analysis is same as in equation (4.6.1) ,however, the number of possible connections is $O(|T| \cdot |S|)$ and not $O(|T|^2)$.

While joining the rooted trees to the substations, apart from the obstacles given as input, we also deal with obstacles formed by the trees (referred to as *tree obstacles*). Therefore, instead of using an upper bound on $\beta \leq |C|$, we use $\beta \leq |C| + |T|$. The addition of $|T|$ is due to the fact that we consider other rooted trees as convex obstacles, and add a line segment between each adjacent extreme points of the convex hull of a rooted tree. However, this does not make any difference as the number of line segments becomes $O(|T| + K + |C|)$, and $|T| >> K$, so μ (as defined for the first part) still remain the same ($\mu = O((|T| + |C|) \cdot log(|T| + |C|))$). While running the Dijkstra's algorithm, we use all the nodes V, therefore, the running time for Dijkstra's algorithm becomes $O(|V|^2 log|V|)$. Thus, the worst case running time with obstacles is given by $O(|T|^3 \cdot |S| \cdot log(|C| + |T|) \cdot |V|^2 log|V|)$. Even without obstacles, we have $\beta \neq 0$, as $\beta = |T| + |C|$. We just put $|C| = 0$ in such an instance.

This is clearly a loose upper bound, as we are assuming that all the line segments referring to the obstacles (both given as input, and tree-obstacles) are intersected. Such a scenario in a large instance would be quite difficult to even conceptualize, especially with our greedy heuristic. More research is required to better understand the geometry to come up with tighter bounds.

4.6.3 *Discussion*

Instances with many obstacles have a high running time, as the number of line segments in the set *Obstacle Lines* (C) , and nodes in the graph (|V|) increases. Each intersection with a line segment from C, results in a call to

Dijkstra's algorithm. One of the easiest way to reduce the number of calls to Dijkstra's algorithm is by removing some of the unnecessary arcs from the arc set A. This includes removing edges between the non-adjacent extreme points of a convex hull of obstacle from the graph. Thus, line segments between non-adjacent extreme points is not required to be added to set C. Hence, line segments only between adjacent extreme points is sufficient to prevent cables from entering restricted areas.

One of the other approaches quite oftently used in VLSI chip design is using Delaunay triangulation to remove some of the unnecessary edges. A delaunay triangulation of the set of points $D \subset \mathbb{R}^2$, denoted by $DT(D)$, is a set of triangles where no point in D is inside the circumcircle of any triangle in $DT(D)$. This can be helpful in reducing the number of edges in the graph, and thus, reducing unnecessary calls to Dijkstra's algorithm. However, this may not have any impact on the running time in big-oh notations or may result in sub-optimal solutions.

5

LOCAL SEARCH HEURISTICS

Local search algorithms are usually applied to hard combinatorial optimization problems. They generate new candidate solutions, and evaluate their objective values. Local search algorithms move from one solution to the other using local moves based on a particular neighborhood structure. We described a large neighborhood structure involving *path* and *cycle*-based exchanges in Section 2.3.3. In this section, we describe some of the details about the neighborhood structure, and present a local search framework to explore new candidate solutions, and to improve solution quality.

Both cycle and path-based exchanges can either have nodes exchanged between trees or entire sub-tree rooted at a node being exchanged. A cycle-based node exchange can be defined by a tree disjoint cycle $a_1 - a_2 \cdots - a_r - a_1$. This exchange represents node a_1 leaving its tree and entering the tree of a_2, subsequently node a_2 leaving its tree and entering the tree containing node a_3 and so on. On the other hand, a cycle-based tree exchange $a_1 - a_2 \cdots - a_r - a_1$, represents that all the nodes in the sub-tree rooted at node a_1 move to the tree containing node a_2, sub-tree rooted at a_2 move to tree containing node a_3 and so on.

The other type of exchange is a path-based exchange. It can be defined by a tree disjoint path $a_1 - a_2 \cdots - a_r$. The last node in the path is not the one at the beginning. The path-based exchange can involve moving of nodes between trees or entire sub-tree at a node to other tree. In the former, only nodes are exchanged between rooted trees, whereas, in the latter entire tree rooted at a node is exchanged. The computational investigations done in [Ahuja et al., 2001] show that for the CMST instances with uniform demand, both path and cycle-based exchanges only transferring nodes have better performance than tree exchanges. Therefore, for cable layout problem we decide to only exchange nodes between rooted trees. We use both path and cycle-based exchanges involving exchange of nodes between rooted trees.

5.1 IMPROVEMENT GRAPH

We denote a feasible set of rooted trees spanning $T \cup S$, and rooted at substations as N. The set N is obtained from the heuristic developed in the previous section. The set X_i denotes the set of nodes in the rooted tree containing node $i \in T$.

We create a directed complete graph called the *Improvement graph*. Each rooted tree disjoint cycle of the improvement graph refers to a potential multiple node-exchange in N. The improvement graph corresponding to N, denoted by $G^1(N)$, is a complete directed graph with node set T and capacity limitation K. Hence, $G^1(N) = (T, A^1)$. We define the cost of the arcs $(i, j) \in A^1$ in the complete improvement graph $G^1(N)$ as :

$$\beta_{ij} = \begin{cases} MST(\{i\} \cup X_j) - MST(X_j), & \text{if } (i,j) \in A^1 : i \notin X_j, |X_i| + |X_j| \le K \\ \text{bigMM}, & \text{otherwise.} \end{cases} \tag{5.1.1}$$

Here $MST(F)$ is the cost of the minimum spanning tree on a subgraph containing nodes in the node set $F \cup \{j^*\}$, where $j^* \in S$, where S is defined as the substation node set and $F \subseteq T$. The node j^* is the substation node nearest to the new rooted tree T_{i^*}, where $(i^*, j^*) \in \arg\min_{(i,j)\in A: i\in F, j\in S} c_{i,j}$, where $c_{i,j}$ is the Euclidean distance between the nodes.

It is easy to interpret a cycle-based node exchange from a tree-disjoint cycle in an improvement graph, but transforming a path exchange with respect to N into a tree-disjoint cycle requires some additional nodes and arcs in the improvement graph. In [Ahuja et al., 2001], the authors suggest adding two types of additional nodes: One pseudonode for each rooted tree in N, and an additional node v called the origin node. So, there are three types of nodes in the improvement graph; regular nodes (turbine nodes) , pseudonodes (one for each rooted tree) and a single origin node v. We denote the set of pseudo-nodes as H, whose cardinality must be equal to the number of rooted trees in the set of trees N. Now, we define a modified improvement graph by adding the above mentioned nodes and a few more arcs: $G^2(N) = (T \cup H \cup \{v\}, A^2)$, where $G^2(N)$ is also a complete graph. The cost of arc between any two nodes i and j can be defined as:

$$\beta_{ij} = \begin{cases} MST(\{i\} \cup X_j \setminus \{j\}) - MST(X_j), & \text{if } i, j \in T, i \notin X_j \\ MST(\{i\} \cup X_j) - MST(X_j), & \text{if } i \in T, j \in H, i \notin X_j, |X_j| < K \\ 0, & \text{if } i \in H, j = v \\ MST(X_j \setminus \{j\}) - MST(X_j), & \text{if } i = v, j \in T \\ \text{bigMM}, & \text{otherwise.} \end{cases} \tag{5.1.2}$$

In equation (5.1.2), bigMM refers to a large number. Thus, $G^2(N)$ is a complete graph with arc weights as described in the equation (5.1.2). Therefore, if both nodes are regular nodes (turbine nodes), then the arc (i, j) denotes adding node i to and removing node j from X_j. An arc from a turbine node to a pseudo-node denotes addition of the node i in X_j. The corresponding cost is positive as none of the nodes are removed from X_j.

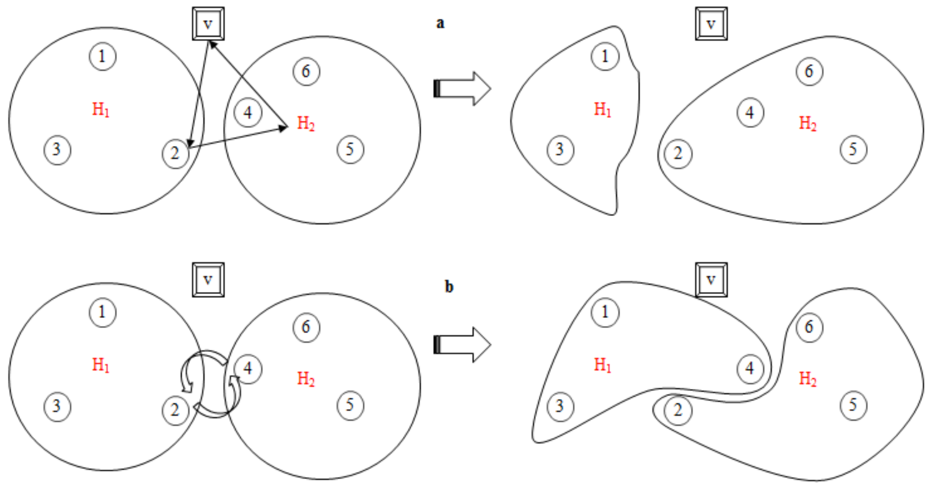

Figure 5.1.1: a: Path-based node exchange using pseudonodes and origin node b: cycle-based exchange

Direct arcs from a pseudonode to origin node v result in zero cost. Any directed arc from the origin node v to a regular node j results in removal of node j from X_j without addition of any node in X_j. Thus, these arcs have negative weights. All the other arcs are given large values so that they are not considered. In Figure 5.1.1, we depict how pseudo-nodes enable transforming path-based exchanges to valid cycles. The final objective is to identify tree-disjoint negative cycles in the improvement graph $G^2(N)$ called *valid cycles*.

Definition 5.1.1. *Valid cycles* in an *improvement graph* are negative weight tree disjoint directed cycles. The sum of the edges of the cycle is negative, and the cardinality of each rooted trees post node exchange is less than or equal to the capacity limitation K.

Lemma 5.1.1. *There is one-to-one correspondence between cost reducing cyclic node exchanges with respect to set of trees N, and valid cycles in the* improvement graph $G^2(N)$ *[Thompson and Orlin, 1989].*

5.2 IDENTIFYING TREE-DISJOINT PATHS

The formation of the improvement graph is followed by identification of *valid cycles*. The problem of identifying all the negative cycles in a graph has been proved *NP Complete* [Thompson and Orlin, 1989]. We recognize the fact that any directed cycle can be obtained by closing a directed path, that is, joining the first and the last node. For example, a directed path $1-2-3-4$, can be used to form a directed cycle by adding arc $(4,1)$. So, this simple idea can be used to enumerate all the valid negative cycles. This method is destined to fail due to enormous redundancies. For example, to obtain a valid cycle $1-2-3-4-1$, we can add arc $(1,2)$ to path $2-3-4-1$, or add arc $(2,3)$ to path $(3-4-1-2)$. Since we are interested in identifying only valid cycles, we reduce these redundant operations by making use of a famous lemma by [Lin and Kernighan, 1973].

Lemma 5.2.1. *If $S = i_1 - i_2 - i_3 - \cdots i_r - i_1$ is a negative cost directed cycle, then there exists a node i_k such that all the partial paths $i_k - i_{k+1}, i_k - i_{k+1} - i_{k+2}, i_k - i_{k+1} - i_{k+2} \cdots$ are negative cost directed paths.*

This idea is used in algorithm *Valid Cycle Detection* (Alg. 5). In the beginning a negative arc is selected, and checked whether it can be extended by one more arc or a negative cycle can be created using a back edge.

For any path P, we define *from(P)* to be the first node and *to(P)* to be the last node, $c(P)$ denotes the cost of the path, which is the sum of the weights of the arcs in the path P. The rooted tree containing node i is given by *cluster(i)*, which is an identifier of the tree. We have divided all the nodes into different rooted trees. *CLUSTERS(P)* is the set of identifiers of all the unique trees visited by arcs in the path P.

Definition 5.2.1. Path A *dominates* a path B if and only if:

- $from(A) = from(B)$,
- $to(A) = to(B)$,
- $CLUSTERS(A) = CLUSTERS(B)$, and
- $c(A) < c(B)$.

We define a set P_k, which stores all the valid *non-dominated* paths of length k. We consider *from*(P), *to*(P) and *CLUSTERS*(P) as the *key value* of a path P.

Observation 5.2.2. If the *key values* of two paths having the same length (k) are equal, then it is possible to replace the dominated path by the dominating path in the set P_k to get the corresponding least cost *valid cycle*.

Algorithm 5 (Valid Cycle Detection) works on the principle of implicit enumeration. After obtaining negative arcs, we use these arcs to find the tree disjoint path of length 2, and so on. While forming these valid paths, we only store those paths which are not dominated by any other path with the same length and key values. For each path in the set P_k, the algorithm checks if the path P can be *extended*.

Definition 5.2.2. A path P is *extendable*, if there exists a neighboring node j to node $to(P)$, such that $cluster(j) \notin CLUSTERS(P)$, and total cost of the resulting path is less than 0.

Thus, Algorithm 5 (Valid Cycle Detection) inductively uses the set P_k to create P_{k+1}.

Algorithm 5: Valid cycle detection [Ahuja et al., 2002]

```
Data: $G^2(N), R$(maximum length of the valid path)
Result: $W^*$: valid cycle with maximum length $R$
$P_1 = \{(i,j) \in A^2 : c_{ij} < 0\}$;
$k = 1$;
$W^* = \emptyset$;
while ($k < R$ && $c(W^*) \geq 0$) do
    while $P_k \neq \emptyset$ do
        remove a path $P$ from $P_k$;
        $i$ =to($P$); $h$ = from($P$) ;
        if $(i,h) \in A^2$ && $c(P) + \beta_{ih} < c(W^*)$ then
            $W^* = P \cup \{(i,h)\}$
        for $(i,j) \in A^2$ do
            if cluster($j$) $\notin$ CLUSTERS($P$) $\wedge$ $c(P) + \beta_{ij} < 0$ then
                add the path $P \cup \{(i,j)\}$ to $P_{k+1}$;
                if $P_{k+1}$ contains another path with identical key as
                  $P \cup \{(i,j)\}$ then
                    remove the dominated path from $P_{k+1}$;
return $W^*$;
```

5.3 DATA STRUCTURE

There is a challenge associated with removal of dominated paths in every iteration. The total number of paths of length k is of the order $O((\frac{n}{k})^k)$, where k and n are length of path and total number of nodes, respectively. Due to a potentially large number of paths with length k in P_k, placing these paths in an array requires a special data structure to reduce the time required to search for paths with identical key values. The data structure should prevent going through the entire array to check whether the new path is dominated by or dominating other paths in P_k. Therefore, we rely on a well-know abstract data type called *hash table*. It utilizes a hash function which maps key values of a path to an array index. This reduces the time to search for the current dominating path. We show this by an example of a Path P with key: from$(P) = 21$, to$(P) = 40$, CLUSTERS$(P) = \{0,2,6\}$, where a unique integer is assigned to every node and tree. We use a standard function used for such calculations:

$$\text{index} = [((31 \times \text{from}(P) + \text{to}(P)) \,\%\, 97) \times 31 + \sum_{x \in CLUSTERS(P)} x] \,\%\, 97 \quad (5.3.1)$$

The notation % is modulo operator in 5.3.1. The use of this formula gives a value of 89 for the above mentioned example. Thus, we check at row 89 and column 2 of the hash table for a path of length 2 dominating the current path or being dominated. The directed path is stored in the appropriate position if no such path exists. We make a note that the constants in the function should be changed depending on the size of the problem to keep the array as compact as possible.

5.4 LOCAL SEARCH ALGORITHM

We make use of the Valid Cycle Detection algorithm (Alg. 5) in a local search framework. In Local Search (Alg. 6), we use *First Accept* strategy. We accept the first feasible solution which is better than our current solution in the neighborhood. This is followed by updating the current best solution, and looking for a better solution in its neighborhood. The algorithm exits once a locally optimal solution is found.

The inputs to the Local Search (Alg. 6) are the initial graph G, solution set of trees N obtained from Obstacle-Aware Heuristic, the set *Obstacle Lines* (C) (eq: 4.3.3), cost of edges between pair of nodes c and cable capacity (K).

We form the improvement graph, and use the Valid Cycle Detection algorithm to form a *minimum priority queue* of valid cycles. The minimum priority queue stores all the valid cycles obtained from the improvement graph, and arranges them in a queue according to their increasing cost for later evaluation. As opposed to the original algorithm presented in Alg.5 where only the best valid cycle is stored, we need to all the valid cycles. This is mainly due to the need to check whether any of these valid cycles results in a crossing free layout with lower cost than the current solution (N).

This is done using a procedure called *CrossingFree*, whose output is a new feasible set of rooted trees (N') achieved by making multiple node exchanges corresponding to the current valid cycle. We do not mention details about this procedure, as it is similar to the Obstacle Aware heuristic, with the only difference being that the partitioning of turbine nodes in different rooted trees is already done. We accept the new crossing free layout if cost of the new solution is lower than the current (N), otherwise we retrieve another valid cycle from the priority queue. The outer loop (external while) is exited when a locally optimal solution is achieved, and the priority queue is empty.

Algorithm 6: Local Search

```
Data: G, Current Solution (N), Obstacle Lines (C), cable capacity (K),
      cost array (c)
Result: $N$: Updated new set of rooted trees spanning $T \cup S$, which is
        locally optimal
begin
    break = False;
    while (break== False) do
        $G^2(N) = ImprovementGraph(N, K)$;
        initiate a minimum priority queue (minPQ) of cycles;
        minPQ = ValidCycleDetection($G^2(N)$);
        $exit = False$;
        while ($\neg$(minPQ.isEmpty()) $\wedge$ $exit == False$)) do
            $W^*$ = remove a cycle with least cost in minPQ;
            $N' \leftarrow CrossingFree(G, N, W^*, C, K, c)$;
            if (Cost of $N'$ is less than the cost of $N$) then
                $exit = True$;
                $N = N'$;
            else
                if minPQ.isEmpty() then
                    $break = True$;
    return $N$;
```

6 EXPERIMENTAL RESULTS

We compare results from the solution methods developed in this work with the results presented in [Klein and Haugland, 2017] . The exact model in [Klein and Haugland, 2017] is similar to the model discussed in Sec. 3.2.1. The model implemented in [Klein and Haugland, 2017] uses CPLEX 12 Python 3.4 API. All the experiments are carried out on a fast computer - Intel Xenon with 72 logical cores and 256GB RAM. The results are presented for wind farms Walney1, Walney 2, Barrow and Sheringham Shoal. We compare our results obtained from developed solution methods with the optimal solutions from [Klein and Haugland, 2017], and report the optimality gaps.

All the experiments are carried out on hardware with the following details: Intel Corei5 2.5GHz, number of processors= 1, number of cores = 2 and 4GB RAM. Numerical experiments are made for multiple cable capacities, and took less than a minute to provide the final solutions.

6.1 INTRODUCTION TO WIND FARM INSTANCES

We have considered 9 real wind farm instances to test solution methods developed in this work. We performed a total of 21 experiments using different cable capacity limitations. We obtained the geodetic coordinates of the turbines and substations from the website [KIS, 2018]. We have converted the geodetic coordinates to geocentric Earth-Centered Earth-Fixed (ECEF) cartesian coordinates using World Geodetic System of 1984 (WGS 84) reference ellipsoid. The data set consists of the following offshore wind farms:

- **Thanet** : The Thanet Offshore Wind Farm is operated by Vattenfall. The wind farm is located approximately 12km from the shore in Southern North Sea, off the East Kent coast. The wind farm has 100 turbines, one substation, capacity of 300 MW, and covers an area of 35km^2.
- **Walney 1 and Walney 2**: Walney 1 and Walney 2 are operated by Ørsted. Walney 1 is located approximately 14km from shore in the East Irish Sea, off the North Lancashire coast. The wind farm has 51 turbines, one substation, capacity of 183MW, and covers an area of

approximately $28km^2$. The neighboring Walney 2 has 51 turbines, one substation, capacity of $183MW$, and encompasses an area of approximately $45km^2$.

- **Sheringham Shoal**: Sheringham Shoal Offshore Wind Farm is operated by Statoil. It is located $23km$ from shore in the Southern North Sea, off the East Norfolk coast. The wind farm has 88 turbines, two substations, a capacity of $316MW$ and encompasses an area of approximately $35km^2$.

- **Barrow**: Barrow is operated by Ørsted. It is situated approximately $7km$ from shore in the East Irish Sea, off the North Lancashire coast. The wind farm has 30 turbines, one substation, a capacity of $90MW$ and encompasses an area of approximately $10km^2$.

- **Race Bank**: Race Bank is operated by Ørsted. It is located approximately $27km$ from shore in the Central North Sea, off the Lincolnshire coast. The wind farm has 91 turbines, two substations, and a capacity of $573MW$. It covers an area of approximately $62km^2$.

- **Gwynt Y Mor** (G.Y.Mor): The Gwynt Y Mor Offshore Wind Farm is operated by RWE Innogy. It is located $16km$ from shore in the Southern Irish Sea, off the North Wales coast. The wind farm has 160 turbines, two substations, and a capacity of 576 MW, and encompasses an area of approximately $68km^2$.

- **Dudgeon**: The Dudgeon Offshore Wind Farm is operated by Statoil. It is located $38km$ from shore in the Southern North Sea, in the Outer Wash. The wind farm has 67 turbines, one substation, and a capacity of 402 MW, and encompasses an area of approximately $55km^2$.

- **The West of Duddon** (Duddon): The West of Duddon Sands Offshore Wind Farm is operated by Ørsted . It is located $15km$ from shore in the Irish Sea, off the North Lancashire coast. The wind farm has 108 turbines, a capacity of $389MW$ and encompasses an area of approximately $67km^2$.

In some wind farms such as London Array, and Greater Gabbard, actual cable layout is not a forest, because of the presence of loops to increase reliability. Hence, in such cases, comparing our solution with actual layout is unfair. In such cases, a capacitated vehicle routing approach is more realistic than a capacitated minimum spanning tree based approach. Apart from the four wind farms Walney 1, Walney 2, Sheringham Shoal, and Barrow, we do not have access to the optimial solutions for other wind farms. Therefore, we compare our solutions with the actual installed cable layout of the wind farms. This reflects the savings which can be achieved by practical use of our solution methods. However, there can be additional constraints imposed on these layouts, and these savings might not be realized.

6.2 INITIAL RESULTS AND MODIFICATIONS

We present a comparison of optimal solutions from [Klein and Haugland, 2017] and solutions from Obstacle-Aware Esau Williams (Alg.3) in Table 6.2.1. We have presented results for three wind farm instances; Barrow having 30 turbines, Walney 1 having 51 turbines, and Walney 2, also having 51 turbines. We have performed computational experiments for cable capacities 2, 4, 5 and 6. We have achieved optimality in only 1 out of 12 experiments. The optimality gap is given by the relation $\frac{cost(heuristic)-cost(exact)}{cost(exact)}$. Optimality gap is on the higher side with the worst being 20% for Walney 2. The Obstacle-Aware heuristic, performs much better in Walney 1 as compared to Walney 2, mainly due to complex geometry of Walney 2. In Fig. 6.2.1, we show the solution from the Obstacle-Aware heuristic for Walney 1 with a capacity limitation (K) of 6. In this instance, we have an optimality gap of 5.89%.

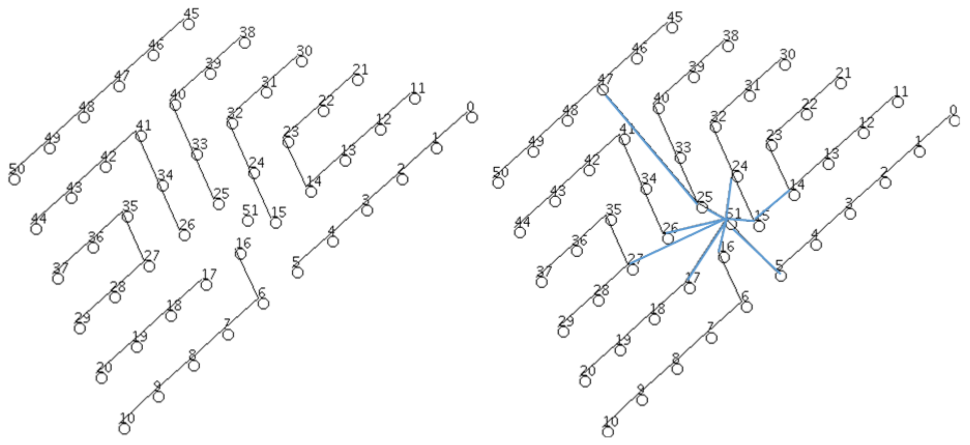

Figure 6.2.1: Final solution obtained from Obstacle-Aware heuristic (Alg.3). Left: Set of rooted trees, Right: Final Layout of Walney 1 with cable capacity 6

K	Barrow(T=30)			Walney-1(T=51)			Walney-2(T=51)		
	Exact	Alg 3	gap(%)	Exact	Alg 3	gap(%)	Exact	Alg. 3	gap(%)
2	36990	38115	3.04	70286	75105	6.85	97885	117849	20.39
4	23208	23243	0.00	47411	49534	4.47	63496	73374	15.55
5	20691	21815	5.43	43420	44444	2.35	56904	62739	10.25
6	18374	20980	14.18	41418	43858	5.89	52981	63568	19.98

Table 6.2.1: Comparison of the solutions from exact method and Obstacle-Aware heuristic method (Alg. 3).

Since, the results are not satisfactory, we propose a hypothesis explaining the bad performance of the heuristic. We analyzed the layouts to understand main reasons. In Fig.6.2.2, it is clear that due to the greedy nature

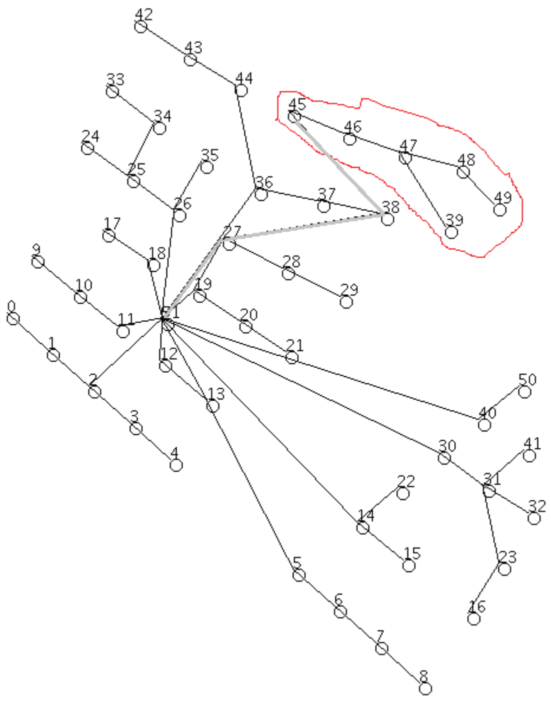

Figure 6.2.2: Walney 2 layout using Obstacle-Aware heuristic (Alg. 3) and K=6

of the heuristic, partitioning of the turbine nodes leads to extremely long paths to the substation. The rooted tree at node 45 is connected to the substation node 51 using a long *crossing free* path. So, our hypothesis to improve the solution quality is partitioning the turbine nodes in such a way that the final connection between the rooted trees and substation should not be extremely long. In the next section, we present a method to improve the solution quality, and compare the results with solutions from our first heuristic.

6.2.1 *Introducing a shape factor*

In this section, we test our hypothesis of improving solution quality by merely recognizing the fact that we need to also encourage radial orientation of the rooted trees towards the substation. The Obstacle Aware heuristic (Alg.3) discussed in the previous section is a greedy heuristic. It tries to keep the rooted trees as compact as possible. This approach is not able to provide good solutions in most of the instances. We tackle this challenge by introducing a shape factor denoted by W ($W \in [0,1]$), where $W \in \mathbb{R}$. We modify the equation of R_i, originally given by the equation (2.3.4) to (6.2.2) $R_i^{'}$. We have also included a more general case for the instances, where the cardinality of substation node set S is not one. This is done by finding

Wind Farm	Exact	Obstacle-Aware		Modified Algorithm	
K=6		value	gap(%)	value	gap(%)
Walney 1	41418	43858	5.89	42580	2.80
Barrow	18374	20980	14.18	18900	2.86
Walney 2	52981	63568	19.98	53214	0.44
K=5		value	gap(%)	value	gap(%)
Walney 1	43420	44444	2.35	43498	0.18
Barrow	20691	21815	5.43	20948	1.24
Walney 2	56904	62739	10.25	57816	1.60
K=4		value	gap(%)	value	gap(%)
Walney 1	47411	49534	4.47	48396	2.07
Barrow	23208	23243	0.15	23243	0.15
Walney 2	63496	73374	15.55	63579	0.13

Table 6.2.2: Comparison of Best Solutions obtained using Modified Algorithm

the substation node s_i closest to the node $i \in T$ (6.2.1). The following two equations define the new reduction value for a turbine i:

$$s_i \in \arg\min\{c_{is} : s \in S\} \tag{6.2.1}$$

$$R_i^{'} = \begin{cases} c_{i,s_i} - \min\{c_{j,i} + W \times c_{j,s_j} : j \in S(i)\}, & S(i) \neq \emptyset \\ 0, & S(i) = \emptyset. \end{cases} \tag{6.2.2}$$

In equation (6.2.2), as mentioned in (2.3.2), $S(i)$ is the set of neighboring nodes which can be joined with node i.

The new reduction function uses a constant called shape factor W to encourage a radial orientation of rooted trees. We have varied the shape factor in the range from 0 to 1 with 1000 equidistant values and identified the best resulting solution. The solution comparison is presented in Table 6.2.2.

6.2.2 *Results after modification - Modified Algorithm*

The introduction of a shape factor W has resulted in improving the performance of the heuristic. The results are compared in Table 6.2.2. The modified algorithm with shape factor identifies better feasible solutions and thus, significantly lowers the optimality gap. We can see from Fig. 6.2.3, that the new layout has a radial orientation and a much lower cable length. We have also plotted the sensitivity of cable length to changing shape factor in Fig. 6.2.4. It can be seen that iterating over shape factors from 0 to 0.4 can give us best solution.

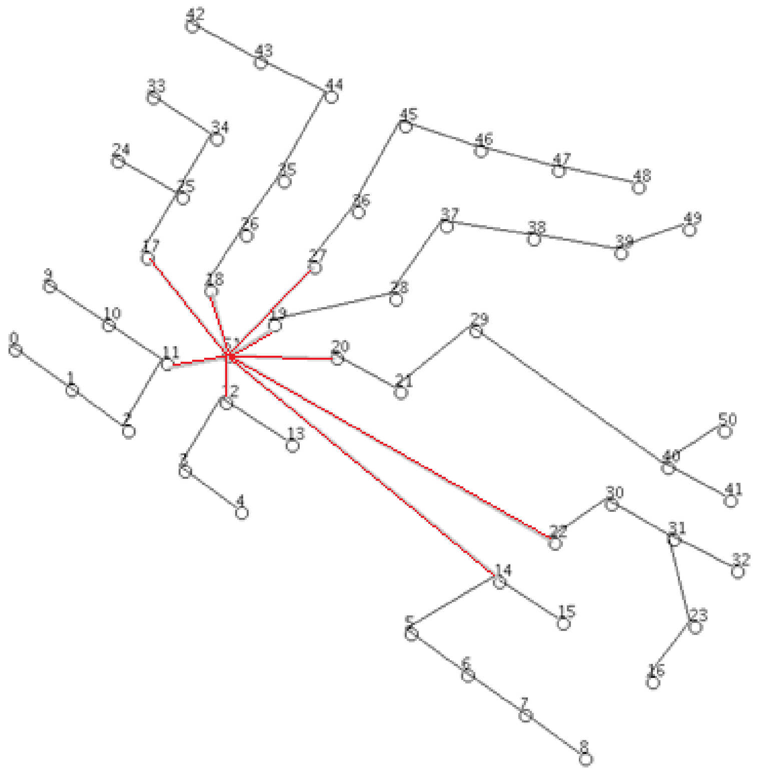

Figure 6.2.3: Final Layout of Walney 2 (K=6) using modified version of Obstacle Aware with a Shape Factor

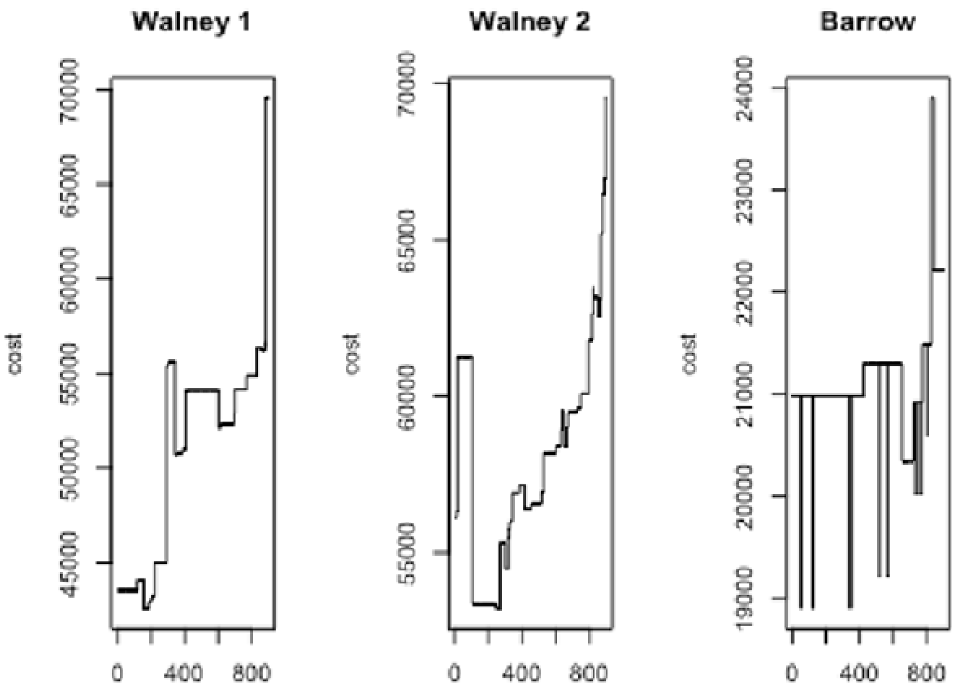

Figure 6.2.4: Change in cost with change in shape parameter, K=6

6.3 FINAL RESULTS

In the previous section, we have presented results using Obstacle-Aware heuristic with shape factor. Now, we also present result obtained after using the local search method developed in Alg. 6. First, we present the results achieved for the four instances for which we have access to optimal solutions. The optimality gap, and actual values are mentioned in Table 6.3.1. The developed algorithm performs well in all the instances. We achieve near optimality (defined as less than 1%) in 8 of the 12 experiments performed on 4 offshore wind farm instances. The final heuristic based algorithm referred to as OWCL algorithm (Alg. 7), which is a combination of both Obstacle Aware heuristic, and local search has outperformed Obstacle-Aware heuristic. We have made use of both the construction heuristic as well as the local search to obtain final solutions. For example, in Barrow (K=6) instance, the optimality gap reduced from 2.86% to optimal after including the local search. Similar impact is seen in Walney 1 (K=4). All the computations were finished in less than a minute.

We have also tried our algorithm on some other wind farms, and compared our solutions with the installed layout. We do not have access to their optimal solutions, so we are not aware of the lower bound. The saving is defined as $\frac{cost(actual)-cost(heuristic)}{cost(actual)}$, where $cost(actual)$ is the cost of installed layout. The savings achieved using our solution methods are presented in Table 6.3.2. Since per meter cost of subsea cables are quite high, even small reduction in cable length may result in huge cost savings. The results are very encouraging, and shows the benefits of using the developed heuristic for larger instances. The heuristic provides better solutions for all the 9 wind farms than the actual layout which is installed. We have also compared our solutions with the actual cable layouts for the wind farms Walney 1, Walney 2, Sheringham Shoal, and Barrow. We have used their actual cable capacity (K).

Some of the large instances, such as Thanet (100 turbines), Race Bank (91 turbines), Gywnt Y Mor (160 turbines), The West of Duddon (108 turbines) also gave better results than the installed layout. This is one of the key benefits of using such a fast heuristic, as these large instances are almost impossible to solve using exact methods.

Wind Farm	Exact	Obstacle-Aware		Modified Algorithm		OWCL
K=6		value	gap(%)	value	gap(%)	best
Walney 1	41418	43858	5.89	42580	2.80	2.80
Barrow	18374	20980	14.18	18900	2.86	0.00
Walney 2	52981	63568	19.98	53214	0.44	0.44
Sheringham Shoal	61463	64817	5.45	63838	3.86	2.89
K=5		value	gap(%)	value	gap(%)	best
Walney 1	43420	44444	2.35	43498	0.18	0.18
Barrow	20691	21815	5.43	20948	1.24	0.72
Walney 2	56904	62739	10.25	57816	1.60	0.10
Sheringham Shoal	64362	72816	13.13	68423	6.30	5.48
K=4		value	gap(%)	value	gap(%)	best
Walney 1	47411	49534	4.47	48396	2.07	0.68
Barrow	23208	23243	0.15	23243	0.15	0.15
Walney 2	63496	73374	15.55	63579	0.13	0.13
Sheringham Shoal	68862	71389	3.67	71389	3.67	2.94

Table 6.3.1: Optimality Gap

Wind Farm	T	K	Actual	Obstacle-Aware	Modified Algorithm	OWCL
				savings(%)	savings(%)	savings(%)
Barrow	30	8	16829	1.56	1.56	1.56
Walney 1	51	10	41048	1.75	2.84	3.05
Walney 2	51	10	50279	0.15	6.24	7.13
Dudgeon	67	6	67047	-7.88	0.62	2.78
Sheringham	88	8	62248	0.0	2.94	3.04
Race Bank	91	6	84073	0.84	6.45	8.49
Thanet	100	10	49828	1.15	1.15	1.75
Duddon	108	10	96328	5.18	5.18	6.77
G.Y.Mor	160	10	106600	-3.88	-0.84	0.32

Table 6.3.2: Savings achieved by heuristic

Algorithm 7: OWCL Algorithm

```
Data: G, Obstacle Lines (C), cable capacity (K), cost array (c)
Result: BEST: Set of trees spanning all turbines and substations with
        each tree having at most $K$ active nodes
current = Star Layout;
/* for each shape factor                                    */
for $w=0:0.001:1$ do
    new = ObstacleAware(G,C,w,K,c) ;                 // Alg. 3
    if $cost(new) < cost(current)$ then
        current = new;
N = current;
BEST = LocalSearch(G,N,C,K,c) ;                      // Alg. 6
return BEST;
```

6.4 CASE STUDIES

In this section, we present layouts of four wind farm instances. In Fig. 6.4.3, we compare the installed layout of The West of Duddon Offshore Wind Farm, and the new layout attained from the solution method (Alg. 7) developed in this work. The final layout is significantly different than the installed layout. The rooted trees are more radially oriented towards the substation as compared to the installed layout. In Fig. 6.4.4, we compare the final layout achieved for Sheringham Shoal after using OWCL algorithm (Alg. 7), with the output from the Modified Obstacle Aware heuristic discussed in previous section. The node exchange which has resulted in the cost reduction can be represented as $60-31-60$. This means that the node 60 leaves its tree to join tree of node 31, whereas, the node 31 leaves its tree to join the tree containing node 60. So, the two nodes 60 and 31 have exchanged their tree memberships. The edges in some of the rooted trees have also changed, because we use a minimum spanning tree algorithm while forming the rooted trees. In Fig. 6.4.4(Top and Bottom), the rooted tree at node 59, has the same nodes in both the layouts, but the edges are different. We also show the comparison for a smaller wind farm instance such as Barrow offshore wind farm. In Fig.6.4.2, it is clear that even for smaller instances such as Barrow, there are ways to improve the solutions.

In Fig. 6.4.1, we have compared the layout of Dudgeon Offshore Wind Farm obtained from the Modified Obstacle-Aware heuristic (Modified Algorithm), and the layout from OWCL algorithm (Alg. 7). Following are the sequence of node exchanges which has taken place in Dudgeon instance:

1. The first node exchange is a cycle-based exchange of nodes represented by $29-28-29$. Node 29 moves to the tree containing node 28, whereas, node 28 joined tree containing node 29. Node 29 forms an edge with node 31, and node 28 forms an edge with node 21

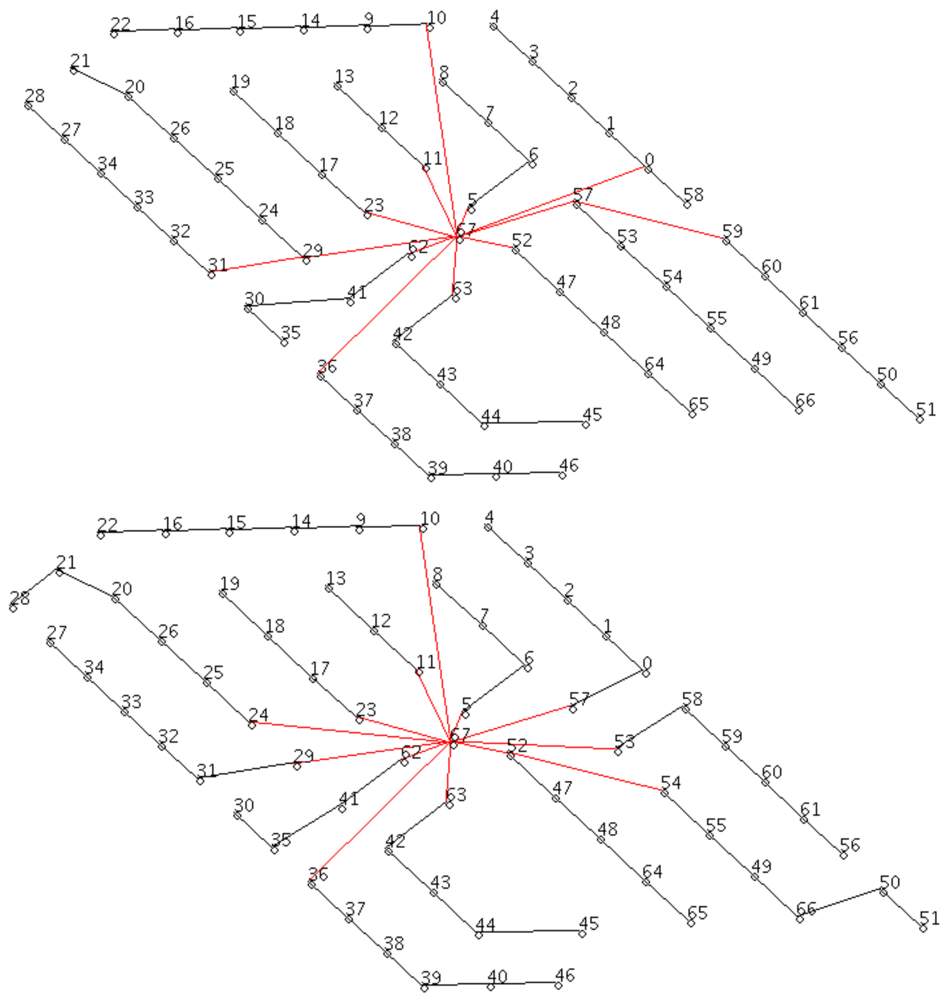

Figure 6.4.1: (Top): Layout obtained by using Modified Obstacle-Aware Algorithm (Modified Algorithm) (Alg. 3) on Dudgeon Offshore Wind Farm having 66 turbines. (Bottom): Final Layout from the solution method developed in this work (Alg. 7). Both the layouts have maximum cable capacity 6

2. The resulting layout was then again subjected to another node exchange 57 – 58 – 57

3. The last node exchange which resulted in a locally optimal solution is 53 – 50 – 53. Final layout is the one at the bottom in Fig. 6.4.1.

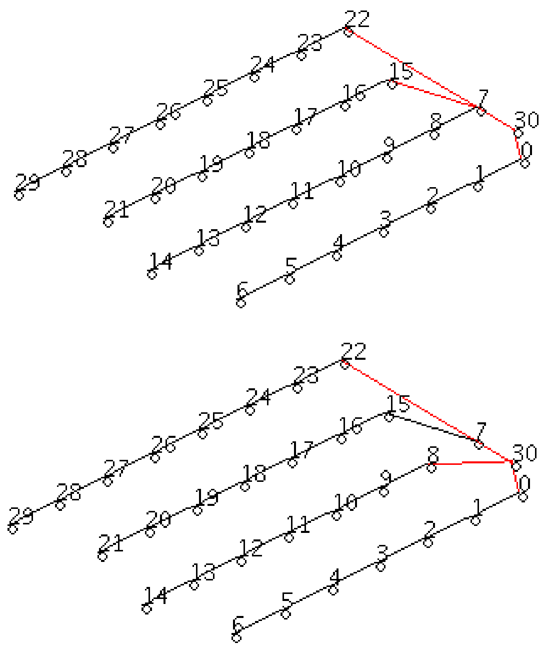

Figure 6.4.2: (Top): Actual Layout of Barrow offshore wind farm having 30 turbines. (Bottom): Final Layout from the solution method developed in this work. Both the layouts have maximum cable capacity 8

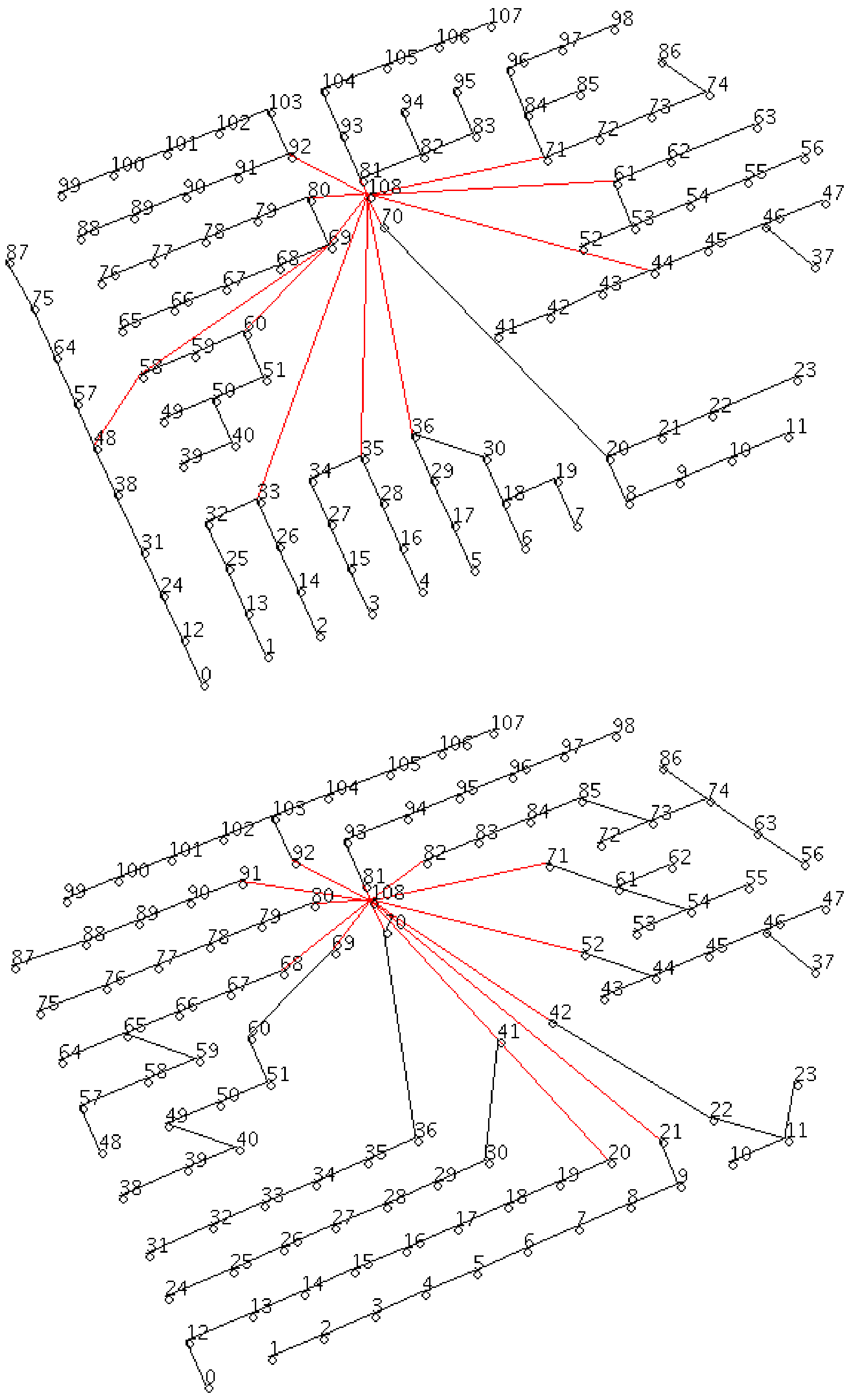

Figure 6.4.3: (Top): Actual Layout of The West of Duddon Offshore Wind Farm having 108 turbines. (Bottom): Final Layout from the solution method developed in this work called OWCL (Alg. 7). Node 108 is the substation node. Both the layouts have maximum cable capacity of 10

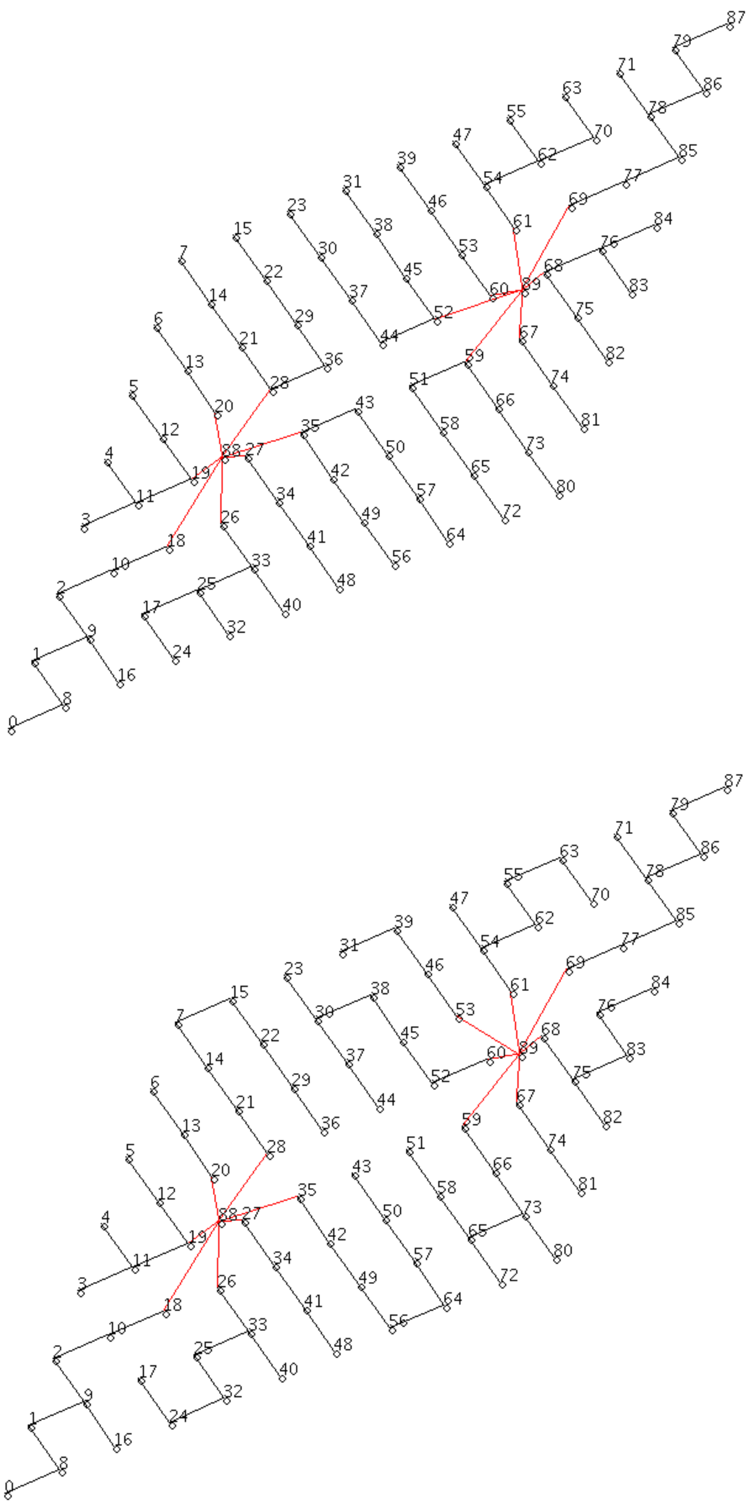

Figure 6.4.4: (Top): Layout obtained by using Modified Obstacle-Aware Algorithm (Modified Algorithm) (Alg. 3) on Sheringham Shoal wind farm having 88 turbines and 2 substations. (Bottom): Final Layout obtained after using OWCL (Alg. 7) developed in this work. The layouts have maximum cable capacity 8.

7 CONCLUSION

In the current work, we have developed a construction heuristic, and a local search heuristic, which attains a good, and feasible solution of the offshore wind cable layout problem. The construction heuristic is a modification of the well know heuristic for the capacitated minimum spanning tree problem known as the Esau-Williams' heuristic. There are two important modifications which have been done to this heuristic to prevent cable crossing, and improve the solution quality. These two modifications are procedures to prevent cable crossing, and using a shape factor to encourage radial orientation of each rooted tree in the solution. The former modification finds feasible layouts, whereas, the latter modification improves the solution quality, seen in Table. 6.3.1 (Obstacle-aware and Modified Algorithm (with shape factor)).

This is followed by development of a local search heuristic based on the multiple node exchange neighborhood presented in [Ahuja et al., 2001], [Ahuja et al., 2002]. We have modified the local search method in such a way that we get feasible layouts for our problem. The solutions from the heuristic have been compared with optimal solutions provided in [Klein and Haugland, 2017]. The solutions are promising, and we are able to get near-optimal solutions in many of the instances and the worst performance is an optimality gap of 5.48%. There are 8 out of 12 experiments that achieved an optimality gap of less than 1%. The solution method developed in the current work is a valuable tool to find good solutions for larger instances. In some of the large instances, it would be simply impractical to use exact methods. This can also be a good tool to do multiple *What-if* analysis using different cable capacities or may be incorporating new restrictions.

7.1 FUTURE RESEARCH DIRECTIONS

In this section, we discuss both the future directions for the heuristic developed in this work and also, for the cable layout problem in general.

The Obstacle Aware heuristic (Alg.3) uses shape factors to generates multiple feasible solutions. Instead of only choosing the best solution, and running local search on it, we could also use some of the other feasible solutions. This will prevent the heuristic from pre-mature convergence and increases the likelihood of identifying global optimum. There are different meta- heuristic frameworks which suggest different acceptance criterion for such a method.

Lastly, new formulations of the layout problem have surfaced in some of the recent work. This is mainly due to the demand in the industry of a customized solution. One such recent work is mentioned in [Fischetti and Pisinger, 2018]. In this paper, the authors have highlighted the need for inclusion of branching penalties, and closed loop structure to increase the reliability of the cable layout. Similar trends are also seen in some other relevant work. Therefore, there is need to add some limitation on the number of branching, or adding some penalties to it. Another important observation regarding this problem is the inclusion of power losses. Although it will complicate the problem, but the final model will be closer to reality.

BIBLIOGRAPHY

[Ahuja et al., 2001] Ahuja, R., Orlin, J., and Sharma, D. (2001). Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Math. Program*, 91:71–97.

[Ahuja et al., 2002] Ahuja, R., Orlin, J., and Sharma, D. (2002). A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Operations Research Letters*, 31:185–194.

[Amberg et al., 1996] Amberg, A., Domschke, and VoB, S. (1996). Capacitated minimum spanning tree: Algorithms using intelligent search. *Combinatorial Optimization: Theory and Practice*, 1:9–39.

[Bauer and Lysgaard, 2015] Bauer, J. and Lysgaard, J. (2015). The offshore wind farm array cable layout problem-a planar open vehicle routing problem. *Journal of Operation Research Society*, 33:360–368.

[Chandy and Russel, 1972] Chandy, K. and Russel, R. (1972). The design of multipoint linkages in a teleprocessing tree network. *IEEE transactions on computers*, c-21:1062–1066.

[Chen et al., 2013] Chen, Y., Dong, Z., Meng, K., Luo, F., Yao, W., and Qiu, J. (2013). A novel technique for the optimal design of offshore wind farm electrical layout. *J. Mod. Power Syst. Clean Energy*, 3:258–263.

[Dahmani et al., 2015] Dahmani, O., Bourguet, S., Macmoum, M., Guérin, P., Rhein, P., and Jossé, L. (2015). Optimization of the connection topology of an offshore wind farm network. *IEEE systems journal*, 9:1519–1528.

[Esau and Williams, 1966] Esau, L. and Williams, K. (1966). On teleprocessing system design part ii. *IBM Systems*, 5:142–147.

[EWEA, 2017a] EWEA (2017a). Accessed: 31-06-2017: `https://windeurope.org/about-wind/wind-energy-today/`.

[EWEA, 2017b] EWEA (2017b). Wind Europe mid year offshore statistics 2016 . Accessed: 31-06-2017: `https://windeurope.org/wp-content/uploads/files/about-wind/statistics/WindEurope-mid-year-offshore-statistics-2016.pdf`.

[Fischetti et al., 2015] Fischetti, M., Leth, J., and Borchersen (2015). A mixed-integer linear programming approach to wind farm layout and inter-array cable routing. In *Conference paper, 2015 American Control Conference, Palmer House Hilton, USA.*

[Fischetti and Pisinger, 2016] Fischetti, M. and Pisinger, D. (2016). Optimizing wind farm cable routing considering power losses. *European Journal of Operational Research*, 261:1–10.

[Fischetti and Pisinger, 2018] Fischetti, M. and Pisinger, D. (2018). Mixed integer linear programming for new trends in wind farm cable routing. *Electronic Notes in Discrete Mathematics*, 64:115–124.

[Gonzaélez-Longatt et al., 2012] Gonzaélez-Longatt, F., Wall, P., Regulski, P., and Terzija, V. (2012). Optimal electric network design for a large offshore wind farm based on a modified genetic algorithm approach. *IEEE systems journal*, 6:164–171.

[Gouveia and Martin, 1999] Gouveia, L. and Martin, P. (1999). The capacitated minimal spanning tree problem. *Anals of Operations Research*, 86:271–294.

[Hou et al., 2016] Hou, P., Hu, W., Chen, C., and Chen, Z. (2016). Optimisation of offshore wind farm cable connection layout considering levelised production cost using dynamic minimum spanning tree algorithm. *IET Renewable Power Generation*, 10:175–183.

[Karnaugh, 1976] Karnaugh, M. (1976). A new class of algorithms for multipoint network optimization. *IEEE transactions on communications*, 24:500–506.

[KIS, 2018] KIS (2018). Accessed: 01-02-2018: `http://www.kis-orca.eu`.

[Klein and Haugland, 2017] Klein, A. and Haugland, D. (2017). Obstacle-aware optimization of offshore wind farm cable layouts. *Annals of Operation Research*, 255:1–33.

[Lin and Kernighan, 1973] Lin, S. and Kernighan, B. (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21:498–516.

[Macedo de Lacerda and de Medeiros Junior, 2006] Macedo de Lacerda, E. and de Medeiros Junior, M. (2006). A genetic algorithm for capacitated minimum spanning tree problem. *IEEE Congress on Evolutionary Computation,Vancouver,BC,Canada*, 1:725–729.

[Malik and Yu, 1993] Malik, K. and Yu, G. (1993). A branch and bound algorithm for the capacitated minimum spanning tree problem. *Networks*, 23:525–532.

[Malik and Yu, 1996] Malik, K. and Yu, G. (1996). Experience with cutting plane approach for the capacitated spanning tree problem. *INFORMS Journal of Computing*, 8:219–234.

[Martin, 1967] Martin, J. (1967). *Design of Real-Time Computer Systems*. Prentice-Hall.

[Pillai, 2017] Pillai, A. (2017). Application of an offshore wind farm layout optimization methodology at middelgrunden wind farm. *Ocean Engineering*, 139(10):287–297.

[Pillai et al., 2015] Pillai, A., Chick, J., Johanning, L., and et all (2015). Offshore wind farm electrical cable layout optimization. *Engineering Optimization*, 47:1689–1708.

[Sharaiha et al., 1997] Sharaiha, Y., Gendreau, M., Laporte, I., and Osman, A. (1997). A tabu search algorithm for the capacitated shortest spanning tree problem. *Networks*, 29:161–171.

[Sharma and Bardai, 1970] Sharma, R. and Bardai, E. (1970). Suboptimal communications network synthesis. *DATACOMM '73 Proceedings of the third ACM symposium on Data communications and Data networks: Analysis and design*, pages 148–156.

[Thompson and Orlin, 1989] Thompson, P. and Orlin, J. (1989). The theory of cyclic transfers. *Working Paper OR*, 200:89.

[Toth and Vigo, 1995] Toth, P. and Vigo, D. (1995). An exact algorithm for the capacitated shortest spanning arborescence. *Anals of Operations Research*, 61:121–141.

[Voss, 2008] Voss, S. (2008). Capacitated minimum spanning trees. *Encyclopedia of optimization*, 1:347–357.

[Wędzik et al., 2016] Wędzik, A., Siewierski, T., and Szypowski, M. (2016). A new method for simultaneous optimizing of wind farm's network layout and cable cross-sections by milp optimization. *Applied Energy*, 182:525–538.