DTU Wind Energy

Optimization heuristics for offshore wind power plant collection systems design

Mauricio Souza de Alencar

DTU Wind Energy-M-0529

June 2022

Author: Mauricio Souza de Alencar

Title:
Optimization heuristics for offshore wind power plant collection systems design

DTU Wind Energy-M-0529
June 2022

Project period:
January - June 2022

ECTS: 30

Education: Master of Science

Supervisors:
Nicolaos Antonio Cutululis
Juan-Andrés Pérez-Rúa
DTU Wind Energy

Remarks:
This report is submitted as partial fulfillment of the requirements for graduation in the above education at the Technical University of Denmark.

DTU Wind Energy is a department of the Technical University of Denmark with a unique integration of research, education, innovation and public/private sector consulting in the field of wind energy. Our activities develop new opportunities and technology for the global and Danish exploitation of wind energy. Research focuses on key technical-scientific fields, which are central for the development, innovation and use of wind energy and provides the basis for advanced education at the education.

Technical University of Denmark
Department of Wind Energy
Frederiksborgvej 399
4000 Roskilde
Denmark
www.vindenergi.dtu.dk

Abstract

In the design of offshore wind power plants (OWPP), the electrical connection of the wind turbines generators (WTG) to power substations through submarine cables comprise the collector system layout. This layout is subject to a limit on the number of WTG connected to each substation circuit (due to maximum cable capacity) and to the interdiction of cable crossings. The optimization's objective function is investment cost, but the total cable length is used as a proxy for that target, while the design variable is the topology of the circuit (WTGs and substation positions are given).

The well-described problem within graph theory called the capacitaded minimum spanning tree (CMST) pursues the interconnection described above while abiding by the cable capacity. In order for a CMST to be a feasible solution for OWPP, it must be further constrained to prevent cable intersections. The CMST has been proven to be NP-hard, which implies that the computational cost to get the optimal solution grows exponentially with the number of OWPP. For this reason, heuristics that reach feasible, though sub-optimal, solutions are an important tool for integrating the collection system in ampler OWPP optimization efforts that combine many disciplines in a single objective function.

This work proposes some extensions to the Esau-Williams (EW) heuristic for the CMST problem with non-crossing constraint. The Crossing Preventing EW (CPEW) introduces Delaunay triangulation to efficiently avoid cable intersection with negligible impact on solution quality. The Obstacle Bypassing EW (OBEW) improves solution quality by expanding the graph with detour nodes that enable more subtrees to reach full capacity and lays the foundation for avoiding forbidden areas. Finally, a change of EW's savings calculation formula with a *rootlust* bonus promotes solutions with a structure more similar to that produced by exact solvers, further reducing the gap from the heuristic layout to the optimal one.

The algorithms are applied to a working set of 11 OWPP – 7 actually built ones, 2 proposed and 2 synthetic – with sizes ranging from 27 to 243 WTG. Cable capacities were varied from 2 to 15 WTG. The implemented heuristics reach feasible layouts for all the pairs OWPP-capacity. Compared with a naive introduction of non-crossing constraints to EW, the proposed algorithms achieved, on average, 1.9% reduction in total length. Global optima for these instances were procured by solving a mixed integer linear programming model (MILP) – by others — with guaranteed optimality gap of 1%. For comparison, these optima are, on average, 4.4% shorter in total length than the naive EW approach.

Algorithm complexity is shown to be polynomial, with $O(N^2)$ for CPEW and $O(N^3)$ for OBEW. This, along with the modest gap of around 3% to the global optimum make these heuristics suitable for use within integrated OWPP optimization efforts.

Contents

CHAPTER 1
Introduction

Wind power is a key resource for the transition to a low-carbon or carbon neutral primary energy matrix. Total generating capacity has been steadily growing and much more growth is expected in order to comply with emission reduction targets in the next decades.

Within wind power, the offshore wind power plants (OWPP) offer access to wind resources superior to most onshore sites. On the energy production side, the winds in the ocean are often stronger and more consistent than on land, while on the project development side, the number of stakeholders affected by an offshore development tend to be smaller than onshore, specially in densely populated countries. Moreover, the transportation of very long blades to the turbine sites are much less challenging on the ocean than for most land locations. In spite of these advantages, the cost of an offshore installation is still significantly higher than an onshore one and this is the main factor holding off that sector's growth.

In an OWPP, the collection system comprises the electrical connection of the wind turbines generators to power substations through submarine cables (also called inter-array cables). These circuits enable the energy produced by the wind turbine generators (WTG) to flow to offshore substations (OSS) that will send the aggregated energy of several WTGs further towards the point of connection (PoC) to the grid. This subsystem can represent around 10% of the total investment cost of a new plant.

Designing this interconnection layout would be a trivial task were not for some technical constraints that must be complied to. The ones that will be considered in the present work are the cable power limit (capacity constraint) and the impracticality of the intersection of two submarine cables (non-crossing constraint).

An OWPP design involves several different subsystems that fall in the scope of distinct disciplines. Most of them have many design variables to explore and this drives optimization efforts towards improving the economic case for the building of the OWPP. However, since the subsystems affect one another, the optimal design for one might restrict the design space of the other to a sub-optimal region. Therefore, only an integrated optimization approach may reach the best compromise between subsystems whose optimization pulls a design variable in opposite directions.

The goal of this work is to provide optimization heuristics for the integration of the design of the collector system layout in a multidisciplinary OWPP optimization framework. The heuristics should aim for solutions that approach the global optimum, but must execute fast enough to provide hundreds of optimal layouts per iteration of the entire OWPP optimization process. Their execution time should also grow at a manageable rate with the increase in problem size. The quality of the heuristics' results will be compared to best

Table 1.1: OWPP sites used for the heuristics' benchmarks.

Handle	Name	# of WTG	# of OSS	Total power	Operator
thanet	Thanet	100	1	300 MW	Vattenfall
dantysk	DanTysk	80	1	288 MW	Vattenfall and SWM
horns	Horns Rev 1	80	1	160 MW	Ørsted
anholt	Anholt	111	1	400 MW	Ørsted
sands	West of Duddon Sands	108	1	389 MW	Ørsted
ormonde	Ormonde	30	1	150 MW	Vattenfall
london	London Array	175	2	630 MW	Ørsted and RWE Renewables
rbn	Ronne Bank North	27	1	(proposed)	Baltic InteGrid
rbs	Ronne Bank South	53	1	(proposed)	Baltic InteGrid
tess	Triangular tesselation	114	1	(synthetic)	
tess3	Triangular tesselation – 3 OSS	243	3	(synthetic)	

estimates of the exact optimal solution.

The next chapter – Chapter 2 – builds up the context for the optimization of the collection system and reviews the literature on the subject. The optimization problem is formulated and the classic Esau-Williams heuristic is introduced.

Chapter 3 presents the algorithms developed to provide good but sub-optimal solutions to the optimization problem. The steps taken to improve upon Esau-Williams's heuristic are described and exemplified with layouts for real OWPPs. The quality of the proposed solutions is compared with globally optimal layouts obtained with a mixed-integer linear programming technique.

Chapter 4 reports the complexity analysis and empirical runtime measurements of the developed heuristics.

Chapter 5 wraps up the report with a summary of the outcomes obtained and suggests future research possibilities.

The OWPP sites used in this work are listed in Table 1.1 (KIS-ORCA, 2022). The locations of their wind turbine generators (WTG) and offshore substations (OSS) are depicted in Figure A.1.

CHAPTER 2
The problem

This chapter describes what the optimization effort is about, explains why heuristic methods are an useful tool, formally states the optimization problem and presents the classic algorithm of Esau-Williams, on which the heuristics presented in this work are based. In the end, previous work on the optimization of the collection system is reviewed.

2.1 Collection system layout optimization

Developing a new OWPP is an investment decision: a project's prospect of becoming an actual plant is linked to its economical attractiveness. Return on investment (ROI), internal rate of return (IIR) or levelized cost of energy (LCoE) are typical figures of merit for quantifying the likelihood of a good payoff for investors. Hence, the most common objectives of OWPP optimization are the improvement of those metrics (higher ROI and IIR, lower LCoE).

The collection system enters these calculations as a cost. It's relevance is greater in the investment costs (capital expenditures – CapEx), but it also influences operating costs (OpEx) through transmission losses and reliability issues. This work examines only the investment cost, which includes both acquisition and installation of the cables, which are together responsible for around 10% of the LCoE for an OWPP (BVG Associates, 2019).

The typical inputs to the collection system optimization are: the set of positions of WTGs and OSS to be interconnected; the power ratings of the generators; the available cable types with their respective power ratings; the maximum number of circuits the OSSs can handle; and the maximum number of connections a WTG can have. The output is a layout (the connections diagram and the cable types used) and its associated cost.

Regarding the interplay of collection system optimization and the entire OWPP optimization, there are two main approaches used: the sequential and the integrated. The first defines the quantity and positions of the WTGs in a previous step not considering cabling costs (or using some rough estimate) and then optimizes the cabling layout (possibly also the OSS positions). The second integrates the collection system optimization within the global OWPP iterations, allowing the former to influence the latter with respect to the WTG positions.

In a OWPP cash flow forecast, the revenue will be a function of energy delivered to the grid, typically summarized by the annual energy production (AEP) figure. This number is strongly affected by the relative positions among the WTGs, since they are subjected to wake losses from the generators up-wind from them. For this reason, AEP maximization tends to spread wind turbines apart, which, in turn, increases the length of collection cables required. It is possible that an AEP improvement, achieved by moving WTGs further from each other, might result in a decrease in IRR due to the additional cabling expenses.

As discussed in (Perez-Moreno et al., 2018) and in (Pérez-Rúa & Cutululis, 2022), the different disciplines that contribute to an OWPP design are frequently considered one at time in a sequence of narrow-domain optimizations. The first in the line tries to maximize AEP (the revenue source) with the quantity, placement and properties of the WTGs as its design variables. Next is the optimization of the support structures (e.g. monopiles), which is much more constrained in the choice of WTG placement. Lastly, the collection system optimization starts from the pre-defined positions and outputs the cable layout with minimal cost. Even if this sequential approach may be iterated a few times, the aforementioned works show that better economic figures for the OWPP can be obtained with an integrated approach, i.e. an optimization process that uses a single objective function encompassing the contribution of all the disciplines.

2.2 Graph representation

Every WTG in a OWPP must have an electrical path via cables to an OSS to deliver the energy it produces. In the ocean environment, the splicing of cables or introduction of junction boxes in sites where there are no WTG are ruled out because they add too much cost compared to the cable length they would save, therefore these electrical connections in the ocean are point-to-point between WTGs or WTG and OSS. The process of defining which connections to make, which cables to use for each of them and the path along which to lay those cables is called the collection system design. A set of values for these design choices will be called a layout or solution.

The electrical circuit formed by the interconnection of WTGs and an OSS can be represented by the *graph* mathematical structure. It is assumed that the reader is familiar with basic graph terminology and concepts such as vertex, node, edge, arc, leaf, path, loop, neighbor, component and reachability. The circuit will be represented by an undirected graph $G = (V, E)$, which has WTGs and the OSS as its vertices V and the cables between any two vertices as its edges $E = \{\{i, j\}\} : i, j \in V \wedge i \neq j$. An edge is simply defined as the pair of vertices at the ends of the cable segment. This model of the electrical circuit does not capture the entire scope of collection system design decisions, but it will be enough for the problem at hand, considering the simplifications that will be explained in the next few sections.

A graph that has no loops is is said to be a tree. If, in addition, all vertices are reachable from all other vertices, the graph is said to be a spanning tree. A spanning tree over the vertices of a OWPP accomplishes the goal of making all the WTG reachable from the OSS

without any redundant connections, therefore containing a spanning tree is a minimum requirement for a graph to be considered a collection system layout for the OWPP. In this context, it is also useful to define the OSS node as the graph's root node.

These are some additional terms that will be used throughout this report:

- *root* node: represents the OSS (also called central or sink nodes in the literature);
- *node*: represents the WTG (also called terminals in the literature);
- *gate* edge: any edge containing the root node;
- *subtree*: the set of nodes that need a particular gate edge to reach the root node;
- *detour* node: auxiliary nodes added at the same position as WTG nodes for the purpose of crossing avoidance;

2.3 Alternate topologies

A tree is not the only shape the collection system may take. It may use circuits that consist each of a single loop or it may be a network, with several embedded loops within. Loops introduce redundancy, which makes a layout more expensive than the non-redundant one, but also more resilient to failures. There can be an economic argument for including loops in a layout, but reliability analysis is out of the scope of this work.

Even not considering loops and networks, a tree may still have three different topologies: the star, the radial and the branched. The star is a very simple layout where all WTG nodes have a direct connection to the OSS node. In the radial topology, each WTG has, at most, two connections, such that every edge leaving the root node form a linear string of WTGs. The branched topology is the more general tree topology, where each WTG may connect to more than two other. Figure 2.1 shows a collection system layout with branching and non-branching circuits. The layouts generated in this work are of the branched type, as the savings in cable length enabled by branching frequently compensate the more expensive switchgears required on branched connections. Even though branching is allowed, the fraction of WTG nodes with branches in the solutions is very small and many of the subtrees where they occur could be turned into linear strings without increasing the cable length.

2.4 The Capacitated Minimum Spanning Tree – CMST

The collection system layout is subject to some constraints, among which there are: the maximum power that the cable can carry (limiting the number of WTGs on each circuit

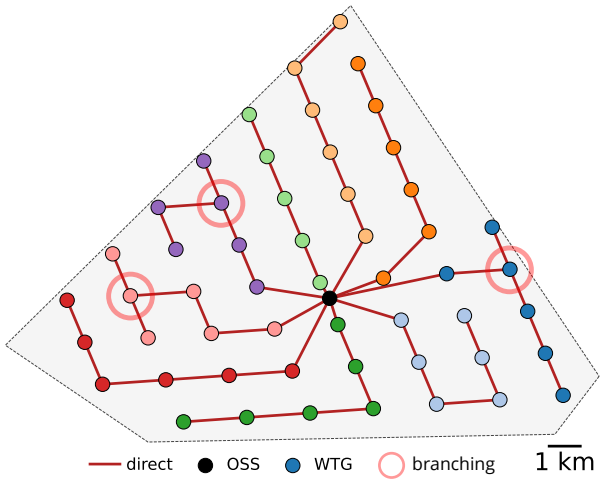


Figure 2.1: An OWPP layout with branching circuits (marked) and radial circuits (not marked).

leaving the substation); the impracticality of cable crossings; and the avoidance of some seabed areas inadequate for laying cables (forbidden areas).

It has been already established in section 2.2 that a spanning tree accomplishes the necessary interconnection of WTGs and OSS and, in section 2.3, that the solutions will be of the branched tree type. If the cable maximum capacity limit is now introduced, the resulting optimization problem becomes the Capacitated Minimum Spanning Tree (CMST). This problem has been well described in combinatorial optimization literature and many exact and heuristic methods have been proposed for its solution.

The CMST, in the classic formulation, does not include the non-crossing or forbidden area constraints nor the use of multiple cable capacities. This last feature can be adapted to to the CMST problem as an additional step of cable assignment after a solution is obtained using only the maximum capacity cable. The algorithms developed in this work use that simplifying approach, which means that the minimization of the total length of the collection system is equivalent to minimizing its cost (length is used as a proxy for cost).

After a reaching a layout, over-sized connection can have their cable type changed to the cheapest one that can handle the actual demand, which is a trivial post-processing task – this procedure is presented as Algorithm 2 in (Peréz-Rúa et al., 2019). This two-step approach to cable type assignment may forgo some optimization opportunities (depending on the cables relative capacity and cost), but that greatly simplifies the calculation of the cost of each connection between vertices within the heuristic. The integration of cable types into the heuristic is suggested as a future improvement or the algorithm.

Both WTG and the electrical cables used to connect them to the OSS have power ratings that specify the maximum the former can produce and the latter can carry. Leaving charging currents aside (given the collection system is typically in medium voltage and the distances within a few tens of kilometers) and considering OWPP commonly use only one WTG model, the cable capacity can be treated as the maximum integer number of

WTG whose aggregated power a single segment of the cable can carry. In this work, the the capacity κ represents the integer part of the quotient of the ratio of cable nominal capacity to WTG nominal power, i.e. $\left\lfloor \frac{P^{\max}_{cable}}{P^{\max}_{WTG}} \right\rfloor$. This calculation uses the cable type with the highest capacity and assumes all WTGs are of the same power rating.

The exact solution to CMST optimization has been shown to be NP-hard (Papadimitriou, 1978), which means the necessary computations grow exponentially with the size of the problem. This makes finding the global minimum for OWPP only practical for a limited size range. Even in this limited range, an exact optimization for OWPPs with more than 50 WTG will require from minutes to hours to obtain a solution close (e.g. within 1%) to the optimum.

The alternative to the costly exact solutions is the use of heuristics and meta-heuristics. These are methods to obtain feasible layouts that are usually "good enough" in terms of total cost but whose distance to the global optimum is unknown.

In order to run one or more collection system optimization solution per iteration of an integrated optimization loop, it is fundamental to have a fast model for the collector system cost as the different turbine placements are assessed. Meta-heuristics, although faster than exact methods, are still a few orders of magnitude slower than heuristics. Besides, any improvements attained in heuristics, are easily transferrable to the meta-heuristics that use them; those gains can even benefit exact methods, by means of providing warm-start solutions for branch-and-cut solvers. Therefore, this work focuses only on the development of heuristic algorithms for the non-crossing constrained CMST problem, aiming at scaling well with problem size and providing good quality solutions.

2.5 Solving the CMST

Pérez-Rúa and Cutululis (2019) presented an extensive review of different approaches for the optimization of the collection system layout. They divided the papers according to method and objective function. The methods listed are: heuristics, meta-heuristics and global optimization; the objective functions were: total length, investment and investment plus electrical losses.

Heuristic is a strategy to reach a feasible good solution in a short time in a single run. These are very problem-specific and often get trapped in local optima, failing to find the exact solution. There is usually no information on how close to the global optimum the solution is and, in some cases, there is no guarantee that a feasible solution will be attained. Heuristics mentioned were: Prim, Dijkstra, Kruskal, Esau-Williams and Vogels Approximation Method.

Meta-heuristics are problem-independent algorithms to guide heuristics in exploring the solution space more broadly. They are typically non-greedy and accept some input variations that initially degrade the quality of the results. Meta-heuristics will apply the underlying heuristic multiple times to come up with one solution. Examples of meta-heuristics are Genetic Algorithm, Simulated Annealing, Tabu Search, Particle Swarm Op-

timization and Ant Colony System. These strategies make the problem tractable, but are still computationally expensive.

Exact solutions can be obtained with a binary integer programming (BIP) or mixed-integer linear programming (MILP) model. They require solver software to find the exact solution (often using the branch-and-cut method), which can be expensive. These solutions have quality guarantees, but their computational cost is exponential with problem size. For solving big problems, which would take too long to find the global optimum, it is possible to define a desired minimum distance from the optimum (the gap, as a percentage of the optimum value) as the stopping criterion for the iterative search.

2.6 Problem definition

In order to formally state the problem, a few more symbols need to be defined:

- let $B_k, k \in V$ be the set of vertices in the subtree k such that $V = \bigcup_{k \in V} B_k$ (note that when $\kappa > 1$ many of B_k will be empty);
- let C_i represent the point in the Cartesian plane corresponding to the position of vertex i.

Then, the CMST with non-crossing constraints problem is defined as:

$$\begin{aligned} \min_{E} \quad & \sum_{i,j \in E} d_{i,j} \\ \text{s.t.} \quad & |B_k| \leq \kappa \; \forall \; k \\ & \nexists \, \{h,i\},\{i,j\} \in B_t \cup \{\text{root}\} \text{ such that } \forall \; k \neq t, p, q \in B_k \cup \{\text{root}\} \\ & \qquad \begin{cases} \overline{C_h C_i} \cap \overline{C_p C_q} \neq \varnothing \vee \overline{C_i C_j} \cap \overline{C_p C_q} \neq \varnothing; \text{ and} \\ \text{the cone } \overleftarrow{C_h C_i} - \overrightarrow{C_i C_j} \text{ splits apart } \{C_p \, : \, p \in B_k\} \end{cases} \end{aligned} \tag{2.1}$$

The objective function in Equation 2.1 is just the total length of the edges. The first constraint is the capacity limit and the second one is the non-crossing constraint.

The simplest definition of a non-crossing constraint would be to forbid any intersections, except for edges with a common node, like in:

$$\overline{C_g C_h} \cap \overline{C_i C_j} \neq \varnothing \quad \text{iif} \quad h = i \; \forall \; \{g,h\},\{i,j\} \in E$$

But this definition would not allow for parallel cables or a cable running by a WTG without connecting to it. Hence, the less orthodox definition in Equation 2.1, where an intersection is necessary but not sufficient to confirm a crossing. There needs to be a split of the nodes of one of the subtrees by two consecutive edges of the other. Splitting in this context means that there are points of B_k both inside and outside the cone defined by the edge pair. Points that overlap with the lines that define the cone are neither inside nor outside.

2.7 The Esau-Williams heuristic

A heuristic that has been used for decades as a reference in the solution of the CMST problem is the Esau-Williams' (EW) algorithm (Esau & Williams, 1966). It was chosen as the basis for the development work, because it is easily extensible and has been the benchmark in most of the literature studying the CMST heuristics. As noted by Amberg et al. (1996), EW has, on average, the best performance among heuristics with similar computation times.

EW is an improvement heuristic, which means it starts with the trivial solution of a star graph and, at each step, replaces one of the node-root edges by a node-node edge in a way that reduces the total cost of the layout. It is also a greedy heuristic, because each step picks the replacement that gives the greatest reduction and this is the single criterion adopted.

Some additional symbol definitions for the EW algorithm:

- r: the root node (i.e. the OSS);
- b_i: a mapping providing the subtree index of node i (for indexing the mapping B of subtree sets);
- g_i: the cost of the gate edge of node i; and
- c: the cost matrix, such that $c_{i,j}$ is the cost of connecting node i to node j.

The workings of the original EW heuristic is expressed in pseudo-code in Algorithm 1 (unaware of crossings). This rendition of the algorithm uses the priority queue mechanism to make it more comparable to the extensions presented in chapter 3. For the same reason, the description of the algorithm is more detailed than usual. In addition, it assumes unitary demand on all non-root nodes (i.e. WTG have the same nominal power), which was not the case in the paper.

The priority queue is equivalent to a table where each entry contains a priority value and a data content. It keeps the table sorted by the priority and extracts the first entry when asked for the next record. The sorting order may be of increasing (min-priority) or decreasing (max-priority) priorities. Internally it is usually implemented using a *heap*, which keeps the entries always ordered and offers $O(\log N)$ complexity for each insertion or deletion of entries.

A fundamental part of EW is in line 19 and line 20, where the savings for the prospective edge of a subtree is calculated. A prospective edge is the best connection to make for a given subtree in replacement for its gate edge, i.e. the one that yields the great reduction in cable length. Prospective edges are ordered by decreasing savings in the priority queue, hence the greediness of the algorithm.

Algorithm 1 Original Esau-Williams heuristic adapted to use a priority queue.

Ensure: $c_{i,i} = \infty \, \forall i \in V$ *no edges from the node to itself*

1: **function** CLASSICEW(V, r, c, κ)

initialization starts here

2: $\mathbf{S} \leftarrow$ new max-priority queue *entries are tuples (priority, data)*

3: **for all** $i \in V$ **do**

4: $E \leftarrow E \cup \{(r, i)\}$ *edge set starts with a star graph*

5: $B_i \leftarrow \{i\}$ *one subtree for each node*

6: $b_i \leftarrow i$ *mapping: "the subtree of node i"*

7: $g_i \leftarrow c_{r,i}$ *gate edge cost of subtree i*

8: $s \leftarrow \max(g_i - c_{i,j} : j \in V)$ *best savings for B_i*

9: add $s, (i, j)$ to $\mathbf{S}$ *add entry for edge (i, j) with priority s*

finished initializing

10: get $s, (u, v)$ from $\mathbf{S}$ *highest savings edge*

11: **while** $s > 0$ **do**

12: **if** $|B[b_u]| + |B[b_v]| \leq \kappa$ **then** *if union within capacity*

13: $E \leftarrow E \cup \{\{u, v\}\}$

14: $B[b_v] \leftarrow B[b_v] \cup B[b_u]$ *merge subtrees*

15: **for all** $i \in B[b_u]$ **do**

16: $b_i \leftarrow b_v$

17: remove from $\mathbf{S}$ the entry $s, (i, j)$ such that $b_i = b_v$

18: **if** $|B[b_v]| < \kappa$ **then** *if there is capacity left*

19: $(i, j) \leftarrow \text{argmin}_{i,j}(c_{i,j} : i \in B[b_v] \wedge j \in (V - B[b_v]))$

20: $s \leftarrow g[b_v] - c_{i,j}$

21: add $s, (i, j)$ to $\mathbf{S}$ *add best edge for $B[b_v]$*

22: **else** *subtree full*

23: **for all** (i, j) such that $i \in V, j \in B[b_v]$ **do**

24: $c_{i,j} \leftarrow \infty$ *make $B[b_v]$ nodes unreachable*

25: **else** *demand of subtree union is higher than capacity*

26: **for all** (i, j) such that $i \in B[b_u], j \in B[b_v]$ **do**

27: $c_{i,j} \leftarrow \infty$

28: $c_{j,i} \leftarrow \infty$ *forbid their merging*

29: $(i, j) \leftarrow \text{argmin}_{i,j}(c_{i,j} : i \in B[b_u] \wedge j \in (V - B[b_u]))$

30: $s \leftarrow g[b_u] - c_{i,j}$

31: add $s, (i, j)$ to $\mathbf{S}$ *add best edge for $B[b_u]$*

32: get $s, (u, v)$ from $\mathbf{S}$

33: **return** V, E

2.8 Previous work on the collection system layout optimization

Many improvements to the EW heuristic have been proposed since it was first published. Öncan and Altınel (2009) present a modified savings calculation formula that includes three additional terms with corresponding factors α, β and γ. These multiply, respectively, the edges' cost, the cost difference between the subtrees' gates and the fraction of the capacity used by the subtree union being considered. Their approach becomes a meta-heuristic, as they suggest the application of the EW heuristic to all combinations of discrete sets of values for each factor and, subsequently, picking the best quality solution. An idea similar to the β factor, mixed with the fraction of the capacity already used, is explored in this work and is described in section 3.4.

Pillai et al. (2015) presented the enforcement of forbidden areas in an MILP model for obtaining the exact solution for the collection system optimization. They used Delaunay triangulation of the OWPP map with forbidden areas within a path-finding heuristic that determines the shortest paths between vertices while avoiding these areas. The length of those paths for each pair of vertices take the place of the euclidean distance between them in the subsequent steps of the method.

Katsouris (2015) examined heuristics for the collection system layout, building upon the work of Bauer and Lysgaard (2015). Their main approach is for the radial topology, which is formulated as the open vehicle route planning (OVRP) problem. They apply a modified Clarke and Wright savings heuristic (Clarke & Wright, 1964) that considers the non-crossing constraint. Katsouris also examines the branched layouts using a modified EW heuristic with non-crossing constraint. In addition, their works employ a local search heuristic as a second step to improve the quality of the solutions offered by the problem-specific heuristic.

Fischetti and Pisinger (2016) took a hybrid MILP + heuristic approach, which takes into account multiple cable types and forbidden areas in the OWPP site. They added extra nodes (called Steiner nodes) on the edges of obstacles to enable the MILP model to make paths around them. These nodes are akin to the detour nodes introduced in subsection 3.3.1. A few so called matheuristics are presented, whose purpose is to reduce the number of variables in the MILP model. The end result was a reduction in the computational cost of solving the MILP, enabling the solution of bigger problems.

Klein and Haugland (2017) also proposed a collection system design method that enables paths bypassing obstacles (such as forbidden areas or other subtrees), this time by adding non-WTG vertices that serve as optional connection points. The bypasses usually result in stretches of cables running in parallel, which is allowed. Their approach was not heuristic, but an exact one: they implemented the method in MILP model and solved for the global optimum with IBM ILOG CPLEX. This concept is similar to the detour nodes presented in subsection 3.3.1, though in the present thesis this was implemented as part of a heuristic.

Fotedar (2018) looked into the same problem and employed a modified EW heuristic.

His approach for crossing avoidance is more elaborate than Katsouris', using Dijkstra's shortest path algorithm (Dijkstra, 1959) to look for paths going around offending edges. However, his crossing detection method adds a lot of complexity to the heuristic. Another change introduced by Fotedar to the EW algorithm was a modified savings formula that includes a shape factor W. This resembles the *rootlust* factor presented in section 3.4 and is further discussed there. The shape factor, however, effectively turns his algorithm into a meta-heuristic, as it needs to sweep a range of values of W (and run EW on each one) to obtain an improvement in the solution quality. As an additional attempt at improving results, Fotedar investigated a post processing step consisting of local search strategy based on the work by Ahuja et al. (2003) which was adapted to check for crossings. These efforts resulted in gains in the quality of the solution, but also increased significantly the execution time. Fotedar hinted at using Delaunay triangulation for reducing potential crossings, but did not actually implement it.

CHAPTER 3
Heuristics

A faithful pseudo-code version of the EW algorithm as initially proposed was introduced in Algorithm 1. In this chapter, the implemented extensions to this algorithm will be presented and explained; they are the following:

- MREW: Multi-Root Esau-Williams (support for multiple root nodes);
- CPEW: Crossing Preventing Esau-Williams (compliance to the non-crossing constraint);
- OBEW: Obstable Bypassing Esau-Williams (detour nodes for going around obstacles);
- Rootlust factor for savings calculation.

Each extension was built upon the previous one, thus only the presentation of the multi-root extension will start from the original EW heuristic. The source code (in *Python*) for the heuristics developed in this work is available at https://github.com/mdealencar/interarray.

3.1 Multi-Root EW

OWPP are becoming larger in both number of WTG and in area. It is unusual for new projects to have a single OSS, therefore an important extension to the algorithm is to support multiple root nodes. Treating a multi-root problem in a single run of the heuristic can allow for less costly solutions when compared to clustering the nodes around each root and solving a separate problem for each cluster.

Multi-root support was the first implemented extension, since it only involves modifying the initialization procedure. The algorithm will now take a set of root nodes R as one of its inputs instead of getting the single root r.

Within that extension, a new mapping is introduced: ρ_i. It stores the root node to which the node i is connected. This is initially the closest root to the node, but may change as the subtrees merge. For the multi-root extension, ρ has no use after the initialization, but it will be more relevant for the remaining extensions. Algorithm 2 shows the new initialization procedure that enables the same main loop of Algorithm 1 to work with multiple roots. It will be referred to from other algorithms further in this report.

Algorithm 2 Initialization of data structures for Multi-Root EW.

Ensure: $c_{i,i} = \infty \; \forall i \in V$

1: **procedure** INITIALIZEMREW(V, R)
2: $\mathbf{S} \leftarrow$ new max-priority queue
3: **for all** $i \in V$ **do**
4: $\rho_i \leftarrow \arg\min(d_{r,i} : r \in R)$ *find closest root*
5: $E \leftarrow E \cup \{(\rho_i, i)\}$ *add gate edge*
6: $B_i \leftarrow \{i\}$
7: $b_i \leftarrow i$
8: $g_i \leftarrow c_{\rho_i,i}$
9: $s \leftarrow \max(g_i - c_{i,j} : j \in V)$
10: add $s, (i, j)$ to $\mathbf{S}$

3.2 Crossing Preventing EW

The laying of submarine cables involves digging trenches in the seabed floor where the cables are laid and covered afterwards. Although the laying of cables across each other is not technically unfeasible, it is highly undesired both for its high cost and for its detrimental effect on the cable's lifetime due the introduction of a hot spot. The EW heuristic is oblivious to cable crossings, which are highly likely to appear on the algorithm's solutions for all but the smallest OWPP, hence the need for the crossing preventing EW (CPEW) extension.

The original heuristic was aimed mainly at graphs in the euclidean plane, but the authors stressed that the cost of the edges could be a function of other characteristics other than the distance between the nodes without any change in the algorithm. In the case of OWPP's collection system layout, the geographical location of the nodes matter for more than the definition of the cost of the edges – it is required to avoid the crossing over of cables. This constraint makes it necessary to deal with the euclidean coordinates of the nodes within the algorithm.

In CPEW, two new data structures are introduced w.r.t. the one presented in Algorithm 1. The first is that the coordinates of all vertices in the OWPP are passed to the algorithm via the list C, which can be indexed by either WTG or OSS (root) nodes, returning a pair of values representing its (x, y) coordinates. The second is the use, within the EW loop, of a set of allowed edges A over the vertices in V. Each edge has as a property it's associated cost. This takes the place of the cost matrix c as the means to forbid edges from being used in savings calculations. Instead of setting $c_{i,j}$ to ∞, the edge (i, j) is removed from A (e.g. when a subtree reaches κ, all edges that contain one of the subtree's nodes are removed).

In addition, since the distance between nodes is being used as a proxy for the cost of the links throughout this work, it is convenient to use a matrix-like notation for retrieving the length of an edge within the pseudo-code. Therefore, a a matrix-like element d will be used to easily refer to the distance between nodes, where $d_{i,j}$ denotes how far apart nodes

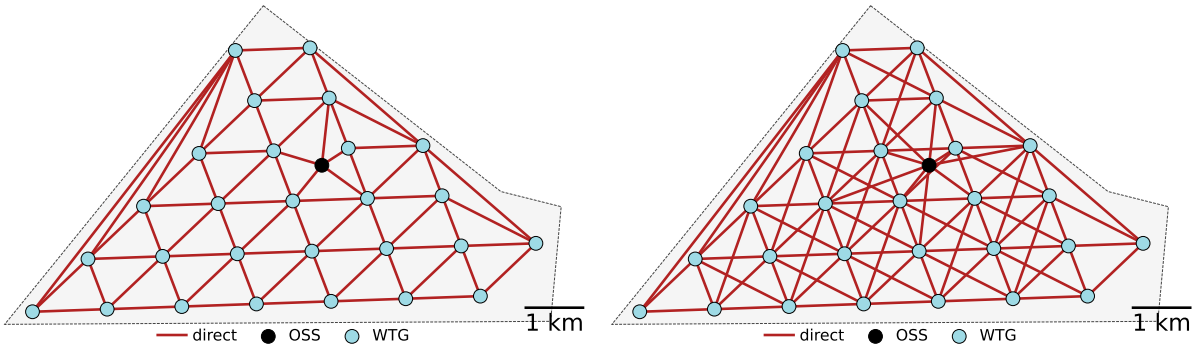


Figure 3.1: Delaunay triangulation (left) and the expanded graph (right) for **rbn**.)

i and j are and is considered as the cost of connecting them.

The approach used to make an efficient check for cable crossings involved restricting the amount of connections that each node can make. The edges relevant for the EW algorithm are the ones leading to nodes close to the one being examined. Only in the later steps of the algorithm, when nearby nodes may already belong to another subtree that is too big to join, looking for nodes further away may be the only option to achieve more savings. But it is also at this later stage that connecting to a distant node will most likely result in a crossing.

It is a known result that the minimum spanning tree (the non-capacitated one) is a subset of the edges of the Delaunay triangulation when dealing with euclidean distances (Shamos, 1978). This triangulation can be computed efficiently in $O(N \log N)$ and contains no crossings.

The Delaunay triangulation is very useful for reducing the available edges from a full graph to much smaller set, but it proved to be too restrictive for finding good solutions for the CMST. Therefore, the edges given by the triangulation were expanded with edges that connect the non-common vertices of each pair of neighboring triangles across the shared side between them. Neighboring triangles form a quadrilateral plus one of its diagonals; this procedure adds the other diagonal if it is not longer than 2.15 times the existing diagonal and if the quadrilateral is convex. This length threshold was chosen arbitrarily after some experimentation. Figure 3.1 depicts the Delaunay triangulation and the expanded edge set for a small OWPP.

The total number of Delaunay edges is given by $2N - 2 - H$, where H is the number of nodes in the convex hull and N is the total number of vertices. The expansion procedure will increase the number of edges by a factor not greater than two, so that the edge count is still $O(N)$. This makes it practical to check for crossings with the help of a mapping from edge to crossing edges that is built along the expansion procedure. Table 3.1 shows the number of edges for the Delaunay, expanded Delaunay and full graph networks as well as the ratio of expanded edges to number of nodes for each of the OWPP in the working set.

The gate edges, however, are mostly not in the Delaunay expanded triangulation. Those edges have the property of being all connected to a root node, which allows for a cheap triage before making more expensive intersection calculations. This triage uses the an-

Table 3.1: Comparison of number of nodes and edges for the OWPP set for different rules.

OWPP handle	Vertices	Delaunay edges	Expanded edges	Full graph edges	Ratio Expanded edges:Vertices
thanet	101	274	426	5050	4.2
dantysk	81	212	320	3240	4.0
horns	81	215	385	3240	4.8
anholt	112	316	485	6216	4.3
sands	109	288	457	5886	4.2
ormonde	31	72	106	465	3.4
london	177	500	768	15576	4.3
rbn	28	70	98	378	3.5
rbs	54	139	209	1431	3.9
tess	115	321	588	6555	5.1
tess3	244	699	1308	29646	5.4

gular position of nodes w.r.t. root nodes (which are calculated at the beginning of the algorithm). It involves comparing the angle between the two nodes under consideration for connection (obstruction angle) with the angle of the existing gate edges, an $O(N)$ operation. A second cheap triage is then made by comparing the length of the target subtree's gate with that of the gates that fell within the edge's obstruction angle. Only the gates longer than the former then go through a line segments crossing algorithm adapted from "Faster Line Segment Intersection" in (Kirk, 1994). When a prospective edge is found to intersect an existing one in the solution graph, it is discarded and another option is searched for the subtree in question.

The CPEW algorithm is presented in two parts: first, the initialization procedure is introduced in Algorithm 3 and only differs from Algorithm 2 by the setting up of the new data structures; second, Algorithm 4 shows the extended EW heuristic as explained in this section.

The implementation encompasses a lot of details that are omitted here for clarity. It is particularly complicated when edges overlap or when a node from one edge is contained in the line segment defined by the other one. These cases are commonly considered as intersections from a geometric point of view, but are not so in the implemented algorithms: they can tell them apart from mid-segment crossings and verify if the overlap causes the sectioning of subtrees (see Equation 2.1). The reason for this relaxation from the geometric criterion is that a concrete realization of a layout with overlaps will require minor amounts of extra cable length to implement parallel lines where a superposition is suggested by the solution (since the clearance of cable trenches are small compared to the inter-WTG distances). A similar reasoning applies to edges going over a WTG without connecting to it.

The CPEW algorithm is able to produce feasible layouts for the entire set of OWPP for each capacity value κ. As an example, Figure 3.2 shows a layout problem for which the classic EW solution is unfeasible because of cable crossings next to the feasible solution

Algorithm 3 Initialization of data structures for CPEW and OBEW.

procedure INITIALIZEEXTENDEDEW(V, R, C)
  $\mathbf{S} \leftarrow$ new max-priority queue
  $A \leftarrow$ EXPANDEDDELAUNAY(V, R, C) *explained in the text*
  for all $i \in V$ **do**
    $\rho_i \leftarrow \arg\min(d_{r,i} : r \in R)$ *store closest root*
    $E \leftarrow E \cup \{(\rho_i, i)\}$ *add gate edge*
    $B_i \leftarrow \{i\}$
    $b_i \leftarrow i$
    $g_i \leftarrow d_{\rho_i,i}$
    for all $r \in R$ **do**
      $\theta_{i,r} \leftarrow$ ANGLEOF(C_i, C_r) *basic geometry, not detailed further*
      $X_{i,r} \leftarrow (i, i)$ *extremities of subtree – only relevant for OBEW*
    $j \leftarrow \text{argmin}_j(d_{i,j} : \{i, j\} \in A)$
    $s \leftarrow g_i - d_{i,j}$
    add $s, (i, j)$ to $\mathbf{S}$

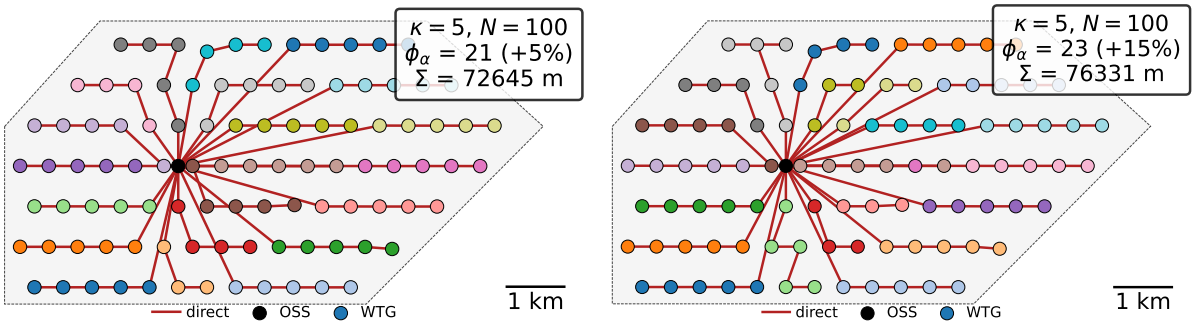


Figure 3.2: Thanet EW results without crossing detection (left) and with it (right)

provided my CPEW for the same problem.

3.2.1 Quality implications of CPEW

The introduction of constraints to EW in CPEW inevitably reduces the overall quality of the solution, i.e. the CMST graph produced has, in general, a greater total length than the unconstrained EW. In order to access that effect, the algorithms are applied to the set of OWPP displayed in Figure A.1, with κ values ranging from 2 to 15. To make it easier to compare results across OWPP of diverse sizes, the relative change of total layout length λ w.r.t. a reference Λ is calculated (for each farm F, for each capacity κ). Results are summarized per OWPP with the mean values Q_F across capacities for each farm, as shown in Equation 3.1.

Algorithm 4 Crossing Preventing EW

```
1: function CrossingPreventingEW(V, R, C, κ)
2:     InitializeExtendedEW(V, R, C)                                  Algorithm 3
3:     get s, (u, v) from S
4:     while s > 0 do
5:         A ← A − {{u, v}}                                  edge won't be searched anymore
6:         if (u, v) does not cross any edges in E then      check for gate/non-gate edges
7:             if |B[b_u]| + |B[b_v]| ≤ κ then
8:                 E ← E ∪ {{u, v}}
9:                 B[b_v] ← B[b_v] ∪ B[b_u]
10:                for all i ∈ B[b_u] do
11:                    b_i ← b_v
12:                remove from S the entry s, (i, j) such that b_i = b_v
13:                if |B[b_v]| < κ then
14:                    B^upd ← B^upd ∪ {b_v}          B^upd holds subtrees that need a new edge
15:                else
16:                    for all (i, j) such that {i, j} ∈ A ∧ j ∈ B[b_v] do
17:                        A ← A − {{i, j}}
18:                        if (i, j) is in S then
19:                            remove entry s, (i, j) from S
20:                        B^upd ← B^upd ∪ {b_i}
21:            else                                  demand of subtree union is higher than capacity
22:                for all (i, j) such that i ∈ B[b_u] ∧ j ∈ B[b_v] do
23:                    A ← A − {{i, j}}
24:                B^upd ← B^upd ∪ {b_u}
25:        else                                                          crossing found
26:            B^upd ← B^upd ∪ {b_u}
27:        for all k ∈ B^upd do          find a new edge for subtrees whose situation changed
28:            (i, j) ← argmin_{i,j}(d_{i,j} : i ∈ B_k ∧ {i, j} ∈ A)
29:            s ← g_k − d_{i,j}
30:            add s, (i, j) to S
31:        B^upd ← ∅
32:        get s, (u, v) from S
33:    return V, E
```

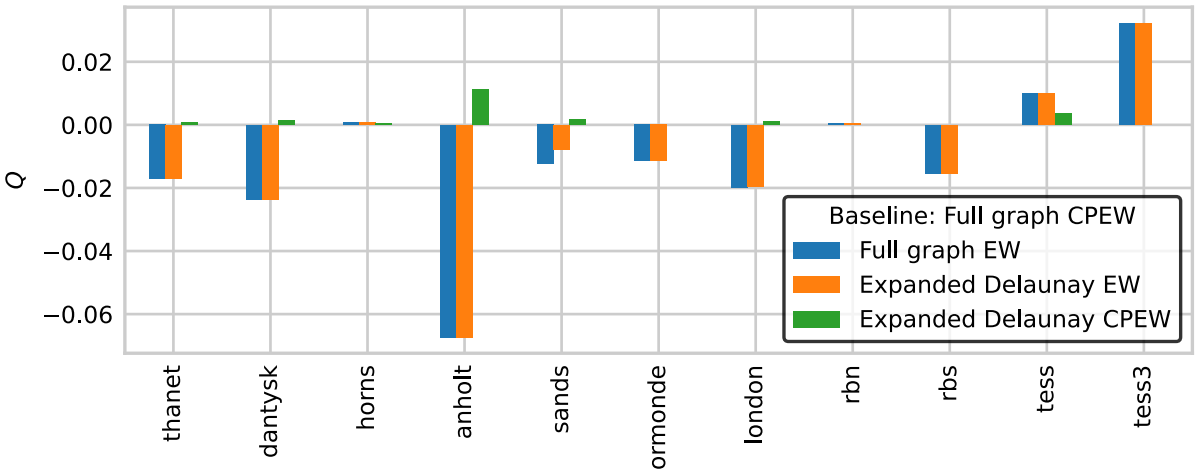


Figure 3.3: Quality comparison of EW and CPEW with and without Delaunay extended edges.

$$Q_F = \frac{1}{14} \sum_{\kappa=2}^{15} \frac{\lambda_{F,\kappa} - \Lambda_{F,\kappa}}{\Lambda_{F,\kappa}} \tag{3.1}$$

The reference algorithm that provides $\Lambda_{F,\kappa}$ is the CPEW using a full graph. This algorithm was chosen because it produces feasible layouts for all OWPP and κ values and is still quite similar to the original EW heuristic. It will be used as the reference for quality comparisons for the remaining of this report.

Figure 3.3 shows the Q values of the algorithms described so far for the full OWPP set. As expected, the unconstrained EW reaches cheaper layouts than the others in most cases, except for the two synthetic OWPP (**tess** and **tess3**), that contradict the overall trend. It should be noted that the restriction on the available edge choices has a very small impact in the qualities of both EW and CPEW heuristics. Considering the benefit of reduced computing complexity when checking for crossings that the expanded Delaunay provides, this seems like an attractive trade-off.

It is somewhat unintuitive that the added restrictions of using only the expanded Delaunay edges and of forbidding cable crossings resulted in a reduction in the total length of the solutions for those two OWPP. Figure 3.4 shows the solutions produced by the original EW and by CPEW for one case of **tess3**, where its clear that the more radially directed subtrees of CPEW decrease the cable required. This behavior is a consequence of the implementation in CPEW of a tie-breaking criterion introduced in the search for the maximum savings edge on Algorithm 4, line 28: if there are multiple edges with approximately the same length, pick the one connecting to the subtree with the shortest gate edge. This tie-breaking has the greatest effect exactly on OWPPs that have very uniformly spaced WTG, such as the synthetic farms **tess** and **tess3**. The lack of a tie-breaking criterion in the original EW results in it choosing essentially in a random way from the several equal-savings options. Hence, even though CPEW is more constrained than the original EW, this slight incentive in moving radially towards the root creates better layouts in some instances.

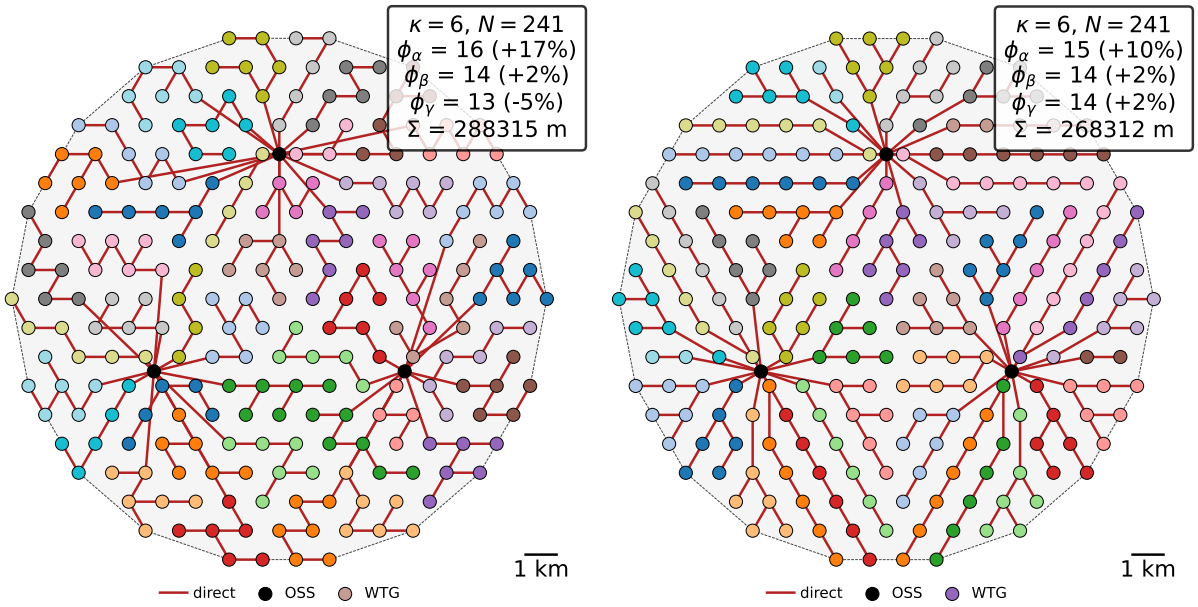


Figure 3.4: Original EW (left) and CPEW (right) layouts for **tess3**.

As for the shortcomings of the CPEW, the resulting layouts often leave small subtrees that still have spare capacity isolated. Some of these are close enough to each other and have compatible sizes for achieving length savings if they were merged, but the edge to effect that is obstructed by another subtree. Besides increasing the total length of the layout, this behavior also incurs in an elevated number of feeders (i.e. gates, represented on the figures by the symbol ϕ) in the OSS, which increase costs and may not even be viable in some cases.

The layouts shown in Figure 3.5 illustrate this point clearly. The information box on the upper right of each layout shows the number of feeders of their OSS; next to this number, in parenthesis, there is the percentage change in number of feeders relative to the minimum quantity necessary at that cable capacity (i.e. $\left\lceil \frac{N}{\kappa} \right\rceil$). The amount of feeders is around 40% higher than the minimum because of the many sub-capacity subtrees.

3.3 Obstacle Bypassing EW

In view of the shortcoming of the CPEW regarding length and number of feeders, another extension was developed, aimed at getting the subtrees to more often grow to their full capacity. This extension was called obstacle bypassing EW (OBEW) and it builds upon the CPEW.

The principle is to examine alternatives to a crossing other than to discard the offending edge. Due to the characteristic behavior of EW, the addition of edges progresses typically from the nodes further away from the root node to those near it. Most of the crossings are with the gate edges of subtrees that have already reached capacity. These edges

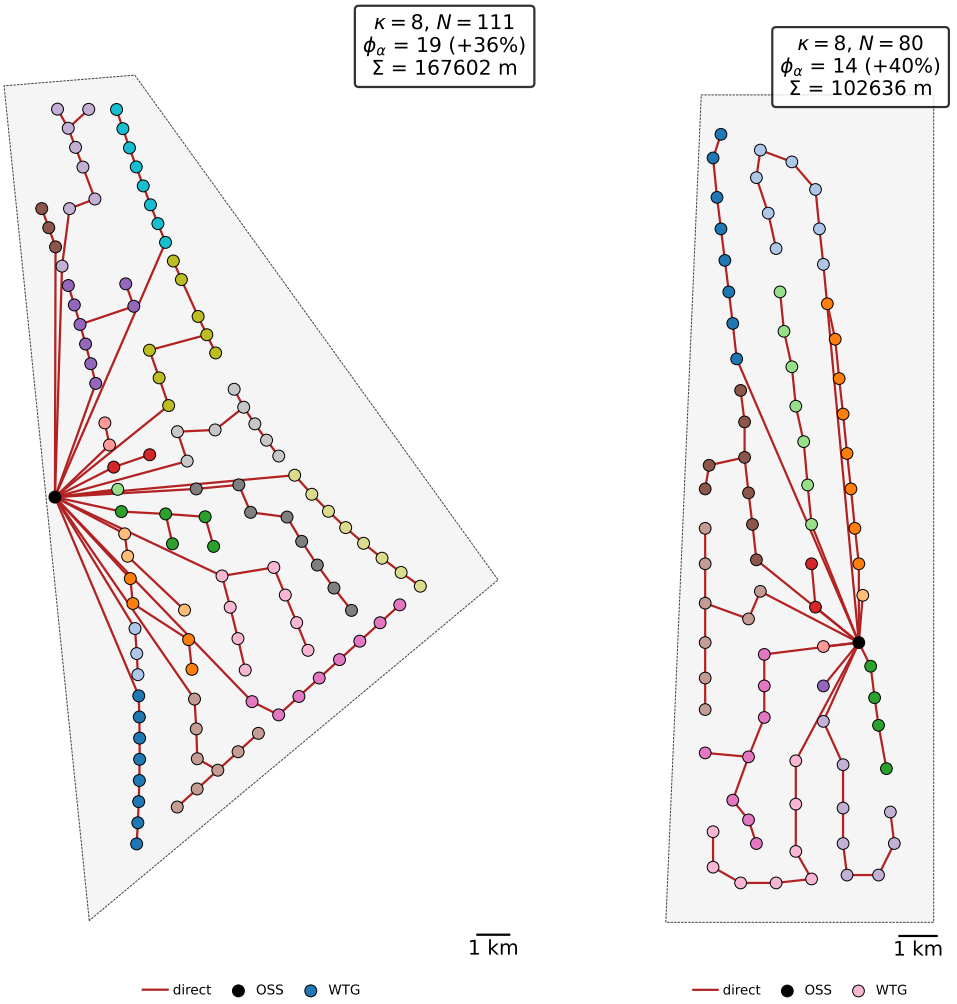


Figure 3.5: Anholt (left) and **dantysk** (right) results from CPEW for $\kappa = 8$.

are commonly long compared to node-to-node edges and can be made to deviate, or take a "detour", from a straight line without much increase in their total length.

Implementing this extension imposed several challenges. The most important of them is that the Delaunay expanded edges are not enough to provide good detour alternatives. Therefore, while the older part of the algorithm still uses them for edge selection and crossing detection, the detour edges must be checked against all edges in E and prospective edges checked against all detour edges. Since the number of detour edges is small compared to the total edge count in a layout, this does not add a lot of computation to the heuristic.

The extension implemented considers alternative routes for all the gate edges that are found to be on the way of a potential edge. It does so by examining the extremities of the obstacle that would be formed if the subtrees were joined and devising paths around them. These detour routes will incur in an added cost in terms of cable length, which must be, in total, less than the savings provided by the edge under scrutiny. In this fashion, every edge addition will have a positive saving on the overall cable length and the subtrees will

have more opportunities to grow. Some candidate edges will still get discarded if the cost of the detours is higher than the savings they offer.

3.3.1 Detour planning

A subtree's extreme points with regards to detour planning are the nodes that the detour may go around in order not to cross any of the subtree's edges. This concept makes the extremities dependent on both nodes of the edge that needs to be replaced. To avoid going through elaborate geometric calculations or graph searches for every detour planning, the algorithm uses instead the nodes most likely to be the extremities: the one furthest in the clockwise direction (low limit) and the one furthest in the counterclockwise direction (high limit), taking the root as the origin of the polar coordinate system. The implementation of extreme nodes assumes that no subtree will span more than 180°. Since gate edges are radial (i.e. in a line that contains the root node) and implemented detours are likely to be roughly in a radial direction as well, this choice of extremes works well most of the time.

The detour extension has two phases: the planning and the execution. Between them takes place the length accounting to decide if the edge should be taken or discarded. Each crossing is examined individually and it is possible for detour gates from previous iterations to enter the process again and get a new detour. A detour will typically convert an edge into two, which will have a new node, called detour node, as their common "corner". Detour nodes are placed at the same position in the Cartesian plane as the WTG node that must be cornered, but they are not connected to that node. It should be noted that the path planning algorithm looks for shorter paths skipping previous detour nodes as well; it may also choose to connect to a different WTG node of the subtree being modified if it is closer to the detour node.

The detour planning procedure has the following inputs:

- r: the root to which the subtree is connected;
- (y, z): the edge in E that needs a detour;
- (u, v): the prospective edge to be added to E;
- (x_L, x_H): the extremities of the subtree that would block (y, z);
- s: detour cost limit (note: this is just to rule out too expensive detours early).

And the return values are:

- c: the detour cost in excess of $d_{y,z}$;
- P: the sequence nodes that defines the detour;
- W: the corresponding side of the obstacle each of the nodes in P is going around.

Algorithm 5 presents the function PLANDETOUR, which finds a crossing-free alternative route for an edge whose removal is being considered. Since there are cases where the edge (y, x_L) or (y, x_H) crosses another edge already in E, the function is recursive, calling itself with the detour edges that need themselves a crossing-free alternative. All the extra distances are accounted for, so that the detours are guaranteed not to cost more than the initial savings. The detour planning is not supposed to provide elaborate routes, it makes a best effort attempt and will give up if it gets too complicated. Still, multiple successive uses of detours will result in non-trivial paths. In addition, the procedure examines both the high and low extremities of an obstacle, which increases the chance of finding a viable alternative.

Algorithm 5 Detour Planning

```
1:  function PLANDETOUR(r, (y, z), (u, v), (x_L, x_H), s)
2:      for all σ ∈ {L, H} do                    will try to corner on the low and high extremes
3:          h ← argmin_j(d[x_σ, j] : j ∈ B[b_y])    find node in subtree that is closest to x_σ
4:          while h is a detour node do
5:              i ← neighbor of h moving away from root
6:              if angle (i, h, x_σ) points to obstacle at h then
7:                  h ← i                        this eliminates unnecessary bends in the path
8:          c_σ ← d[h, x_σ] + d[x_σ, z]−LENGTHOF(h, ..., y, z)    calculate added cost
9:          if s − c_σ < 0 then
10:             c_σ ← ∞                          not worth making a detour on this side
11:         else
12:             if (h, x_σ) intersects one (i, j) ∈ E then
13:                 (χ_L, χ_H) ← X[b_i, r]
14:                 (c, P, W) ← PLANDETOUR(r, (h, x_σ), (i, j), (χ_L, χ_H), s − c_σ)
15:                 c_σ ← c_σ + c
16:                 P_σ ← (P, x_σ)
17:                 W_σ ← (W, σ)
18:             else if (h, x_σ) intersects many (i, j) ∈ E then
19:                 c_σ ← ∞                      too complex, giving up
20:             else                             no crossings
21:                 P_σ ← (h, x_σ)
22:                 W_σ ← (σ, )
23:     σ ← arg min(c_σ : σ ∈ {L, H})
24:     if c_σ < ∞ then
25:         return c_σ, P_σ, W_σ
26:     else
27:         return ∅
```

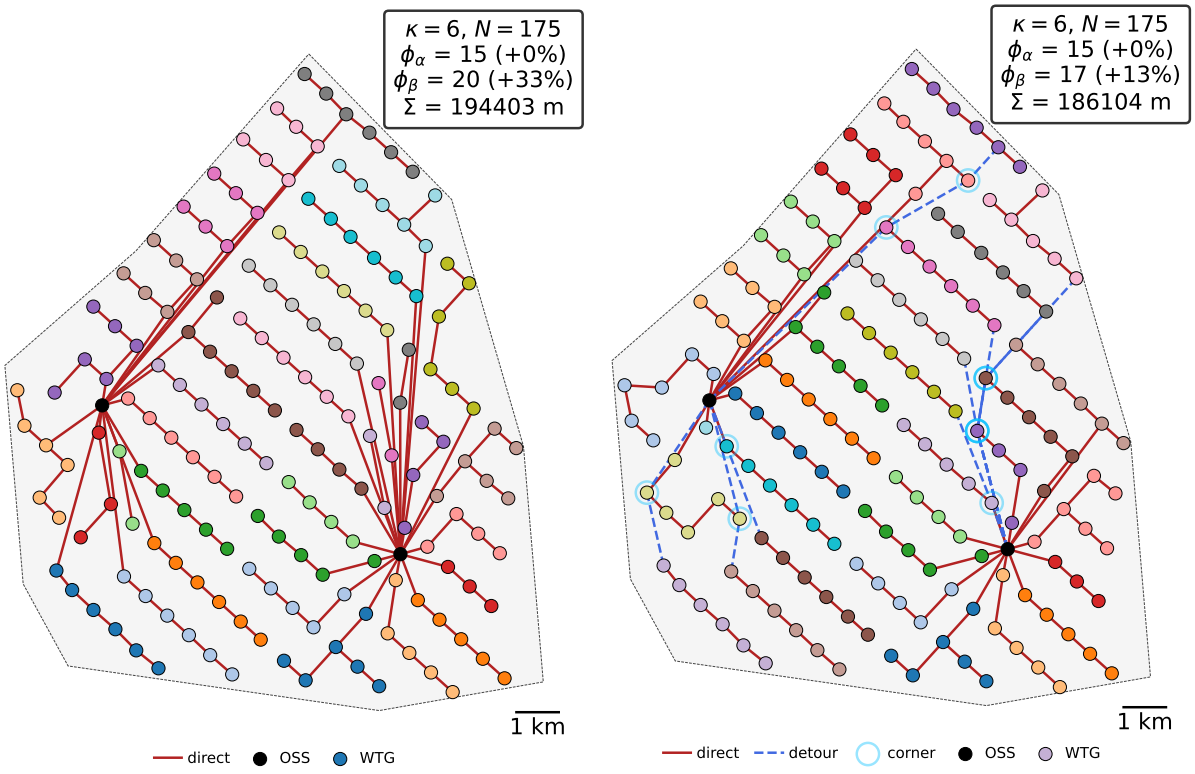


Figure 3.6: CPEW (left) and OBEW (right) layouts for **london** at $\kappa = 6$.

3.3.2 OBEW heuristic

The OBEW is shown in Algorithm 6. The check for intersections with regular edges is separate from the one with gate edges. The former is eliminatory – (u, v) is discarded if a crossing is detected – while the latter triggers detour planning. The crossings might be multiple and an attempt will be made to find a detour for each. Next, the costs are summed up a a decision made to add or discard the prospective edge. Adding an edge that requires detours will implicate the modification of E to remove some edges and add others. Another additional step compared to CPEW is the updating of the subtree's extremities w.r.t each of the roots after a merger is performed.

Figure 3.6 illustrates the effect of introducing detour nodes in the EW heuristic. There is a significant reduction in both the total length and in the total number of feeders.

The aggregated effect across capacities of OBEW can be seen in Figure 3.7. This heuristic provides better quality solutions for all OWPP except **rbn**. This is the smallest OWPP of the set (only 27 WTG) and the quality is less than half a percentage point worse, so it is fair to say that OBEW is a significant improvement over CPEW.

Algorithm 6 Obstacle Bypassing EW

function ObstacleBypassingEW(V, R, C, κ)
  InitializeExtendedEW(V, R) *Algorithm 3*
  get $s, (u, v)$ from **S**
  while $s > 0$ **do**
    $A \leftarrow A - \{\{u, v\}\}$
    if (u, v) does not cross any non-gate edges in E **then**
      if $|B[b_u]| + |B[b_v]| \leq \kappa$ **then**
        $p \leftarrow 0, \quad P \leftarrow \emptyset$
        $\chi \leftarrow$ gate edges in E that intersect (u, v) *this uses the nodes' angles*
        for all $(y, z) \in \chi, i \in [1, |\chi|]$ **do**
          $(\chi_L, \chi_H) \leftarrow$ extreme nodes of $X[b_v, \rho[b_y]] \cup X[b_u, \rho[b_y]]$
          $(c_i, P_i, W_i) \leftarrow$ PlanDetour$(\rho_v, (y, z), (u, v), (\chi_L, \chi_H), s)$ *plan detours*
          $p \leftarrow p + c_i$
        if $p < s$ **then** *either detour is within budget or there are no crossings*
          implement paths $P_i \, \forall \, i \in [1, |\chi|]$ in E in substitution of crossed gates
          $E \leftarrow E \cup \{\{u, v\}\}$
          $B[b_v] \leftarrow B[b_v] \cup B[b_u]$
          update extremities $X[b_v, r] \, \forall \, r \in R$
          for all $i \in B[b_u]$ **do**
            $b_i \leftarrow b_v, \quad \rho_i \leftarrow \rho_v$
          remove from **S** the entry $s, (i, j)$ such that $b_i = b_v$
          if $|B[b_v]| < \kappa$ **then**
            $B^{upd} \leftarrow B^{upd} \cup \{b_v\}$
          else
            for all (i, j) such that $\{i, j\} \in A \wedge j \in B[b_v]$ **do**
              $A \leftarrow A - \{\{i, j\}\}$
              if (i, j) is in **S** **then**
                remove entry $s, (i, j)$ from **S**
                $B^{upd} \leftarrow B^{upd} \cup \{b_i\}$
        else *detour not found for gate crossing*
          $B^{upd} \leftarrow B^{upd} \cup \{b_u\}$
      else *demand of subtree union is higher than capacity*
        for all (i, j) such that $i \in B[b_u] \wedge j \in B[b_v]$ **do**
          $A \leftarrow A - \{\{i, j\}\}$
        $B^{upd} \leftarrow B^{upd} \cup \{b_u\}$
    else *edge to edge crossing*
      $B^{upd} \leftarrow B^{upd} \cup \{b_u\}$
    for all $k \in B^{upd}$ **do** *find a new edge for subtrees whose situation changed*
      $(i, j) \leftarrow \text{argmin}_{i,j}(d_{i,j} : i \in B_k \wedge \{i, j\} \in A)$
      $s \leftarrow g_k - d_{i,j}$
      add $s, (i, j)$ to **S**
    $B^{upd} \leftarrow \emptyset$
    get $s, (u, v)$ from **S**
  return V, E

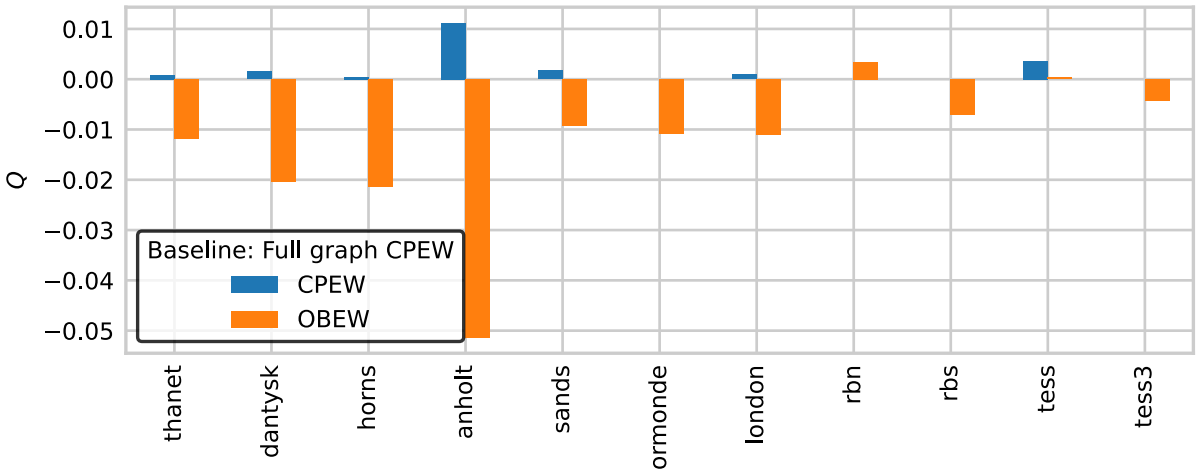


Figure 3.7: Quality comparison between CPEW and OBEW.

3.4 Rootlust bonus factor

A good source of ideas for improving the heuristics is to look at the solutions known to be very close to the global minimum of the optimization problem. Pérez-Rúa and Cutululis (2022) described a mixed integer linear program (MILP) global optimization formulation of the CMST problem. This approach looked into minimizing the total investment cost, which, in the general case, is not a linear function of total cable length. However, by restricting the cable types in the model to a single one, the solution becomes equivalent to the minimal length layout. Their implementation of the MILP model (Pérez-Rúa, 2022) was used along with IBM ILOG's CPLEX version 22.1 to obtain layouts within 1% of the global minimum. The MILP formulation does not consider anything akin to detour nodes, hence those solutions are of the type produced by CPEW, with only direct connections between nodes.

Several comparisons of the layouts produced by OBEW with those given by the global optimizer (GO) revealed that the subtrees by GO have a different general structure from those by OBEW (and CPEW, for that matter). Figure 3.8 and Figure 3.9 show the solutions by the two different methods side by side and illustrate that point. The subtrees in GO tend to have the WTGs closer to the gate aligned almost radially (i.e., pointing to the root).

These observation supported the introduction of a change in the savings calculation formula to favor the union to subtrees that have a shorter gate edge than the one being examined for connection options. This concept is not new, it has been proposed by Öncan and Altınel (Öncan & Altınel, 2009) as the parameter β and by Fotedar (Fotedar, 2018) as the shape factor W. Both authors intended on being able to direct the progression of EW towards radial edge lines. Their parameters, however are applied to static properties of the nodes (i.e. related to their relative positions). They managed to get better quality layouts by sweeping a range of values for their parameters, in what could be called a metaheuristic.

A different approach is proposed here. The incentive (or savings bonus) is to be proportional to both the reduction in gate length (just like the β factor) and the fraction of the

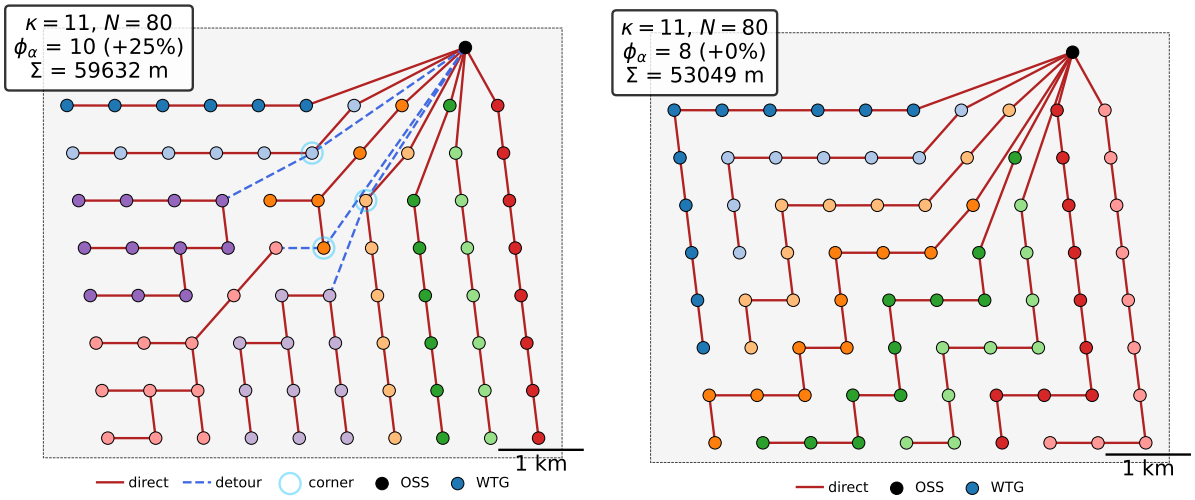


Figure 3.8: OBEW (left) and Global Optimizer (right) layouts for **sands** at $\kappa = 11$.

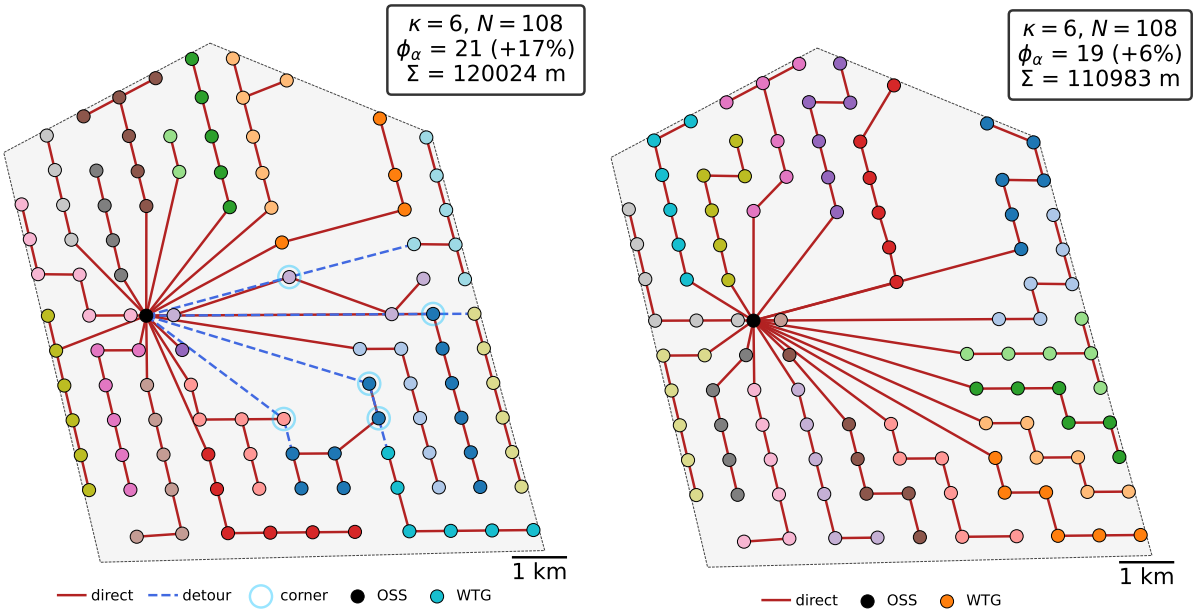


Figure 3.9: OBEW (left) and Global Optimizer (right) layouts for **sands** at $\kappa = 6$.

total capacity κ already filled by the subtree. Equation 3.2 shows the proposed formula for the savings calculation using the *rootlust* bonus factor $\lambda|B[b_i]|/\kappa$.

$$s_{i,j} = g[b_i] - d_{i,j} + \lambda \frac{|B[b_i]|}{\kappa}(g[b_i] - g[b_j]) \tag{3.2}$$

This formulation incentivizes the selection of edges that will join the subtree to others with shorter gate edges, which means moving in a radial direction. Differently from previous work, the size of the incentive grows as the subtree approaches its capacity limit. This way, there is still a phase of less directed subtree expansion while it is small, preventing an early run to the center and allowing for accumulating the savings of linking nodes that are far from the root.

The change in Algorithm 6 to implement the *rootlust* bonus involves only line 39 and line 40, as shown in Algorithm 7.

Algorithm 7 Changing OBEW (Algorithm 6) to use *rootlust*

39: $(i, j) \leftarrow \text{argmin}_{i,j}(d_{i,j} : i \in B_k \wedge \{i, j\} \in A)$
40: $s \leftarrow g_k - d_{i,j}$ *original EW savings formula*
changes to:
39: $(i, j) \leftarrow \text{argmin}_{i,j}(d_{i,j} - \lambda \frac{|B_k|}{\kappa}(g_k - g[b_j]) : i \in B_k \wedge \{i, j\} \in A)$
40: $s \leftarrow g_k - d_{i,j} + \lambda \frac{|B_k|}{\kappa}(g_k - g[b_j])$ *rootlust savings formula*

Figure 3.10 puts together four layouts obtained by different methods for the same problem. The advancements in total length reduction are made both with the decrease in the number of subtrees as well as with more of the subtrees including nodes closer to the root.

The aggregated results per OWPP for OBEX, OBEX with *rootlust* ($\lambda = 0.6$ and $\lambda = 0.7$) and Global Optimizer are presented in Figure 3.11, aggregated by OWPP, and in Figure 3.12, aggregated by capacity (refer to Figure A.2 for the non-aggregated results). The most relevant gains are introduced with the OBEX heuristic, but the *rootlust* savings formula provides an additional benefit for most of the OWPP. Interestingly, when looking at results at Figure 3.12, the proposed heuristics actually achieve, in average, a lower total length than the global optimization for $\kappa = 2$. It is at the lower end of capacities than the ability to route around other subtrees provide the greatest benefit. As capacity increases, the *rootlust* bonus factor becomes more relevant for the improvement in total length reduction. The capacity range $\kappa = [7, 10]$ is where the gap between the algorithms solutions and the global optima is the largest.

To take a closer look at the lower end of the capacity, where OBEW sometimes reach better layouts than the exact optimization method, some comparisons are shown in Figure 3.13 and Figure 3.14. It is clear that at low capacities the EW property of grouping the furthest nodes together is a good strategy, as long as their gate edges do not prevent other good groupings to form along the way – which is enabled by the detour nodes. It is likely that a MILP implementation that allows the bypassing of obstacles, like the one suggested by Klein and Haugland (2017), will achieve exact solutions even shorter that the ones produced by OBEW.

Figure 3.10: A view of the progression towards the global optimum for **anholt** at $\kappa = 13$.

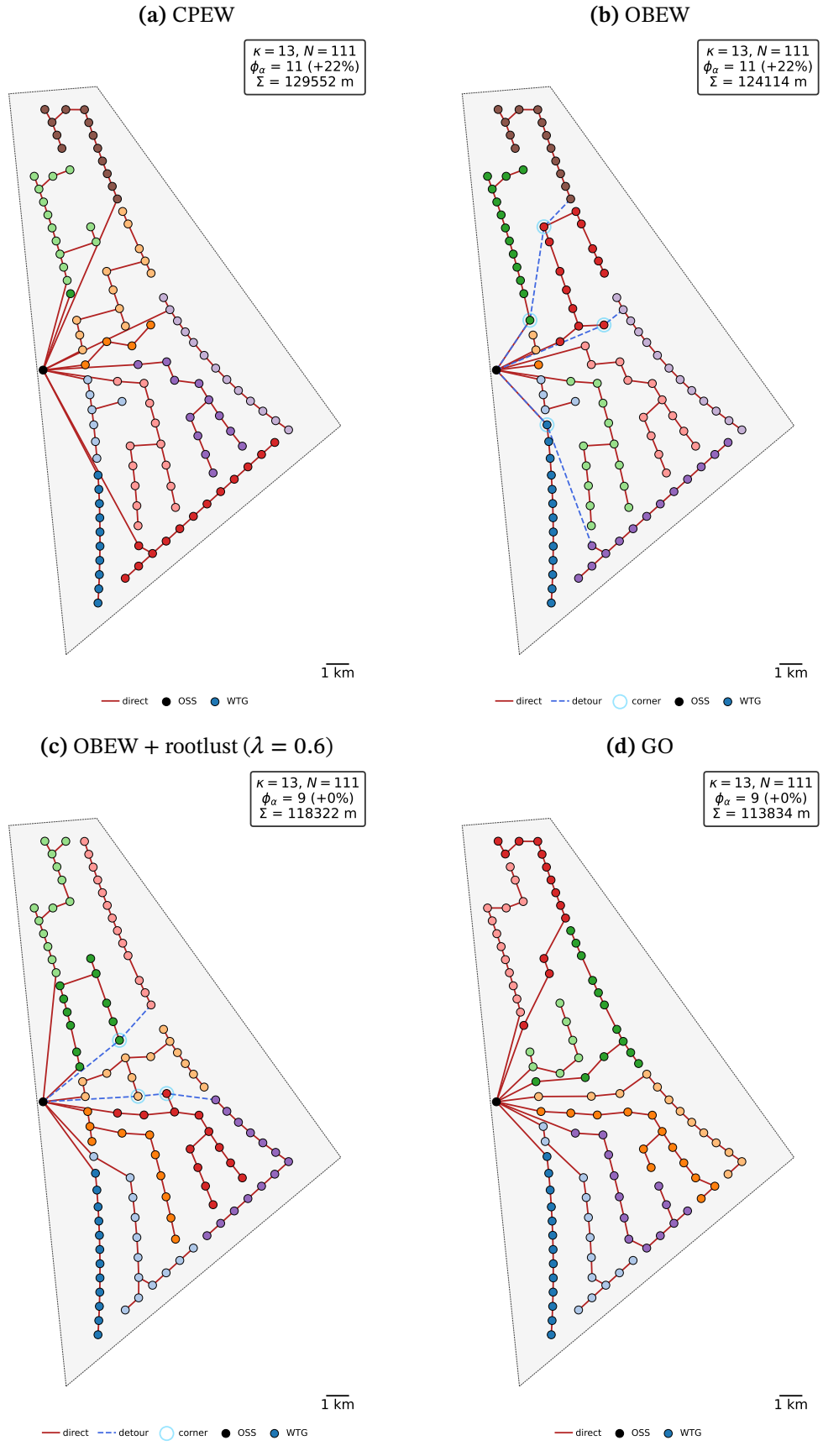


Figure 3.11: Quality comparison of OBEW, OBEW with *rootlust* and the Global Optimizer.

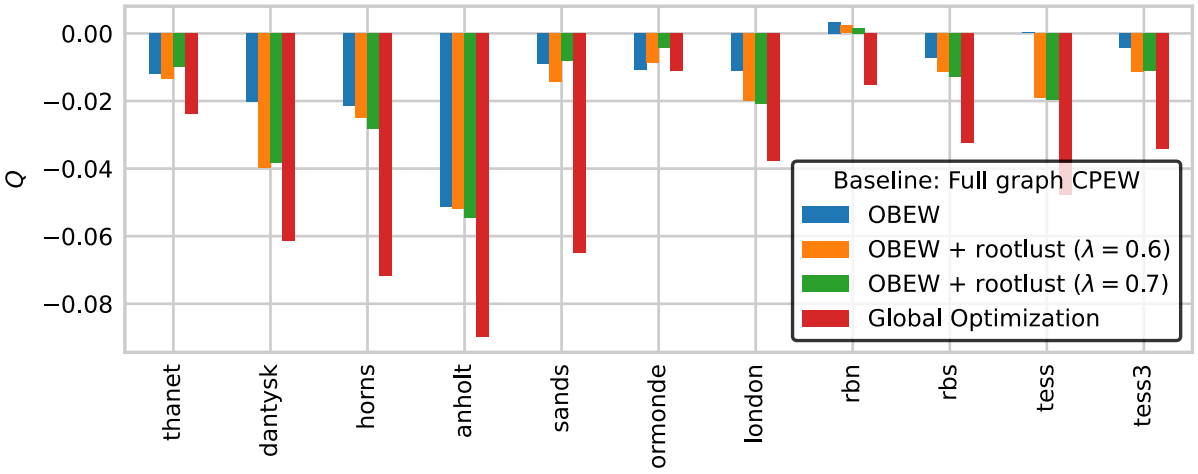
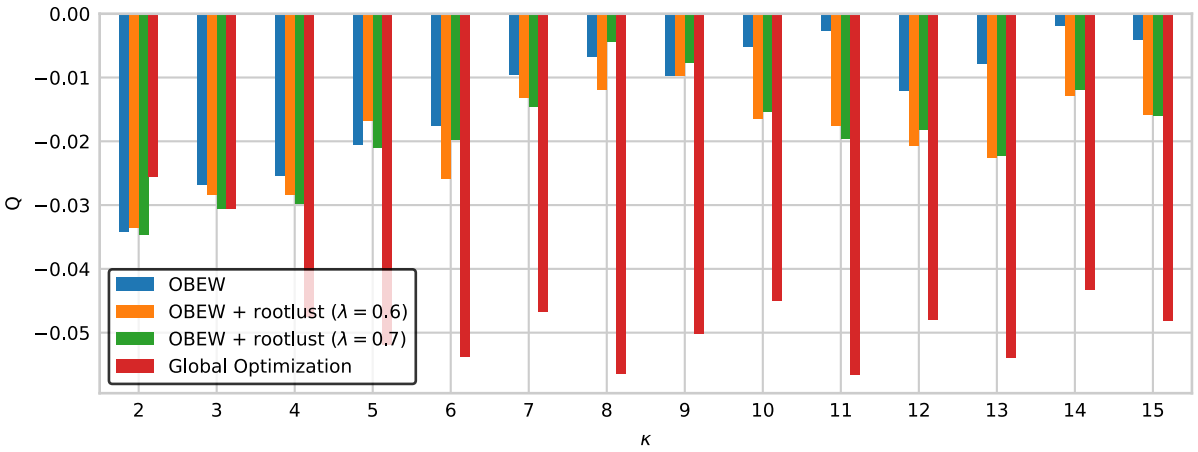


Figure 3.12: Quality comparison aggregated by capacity.



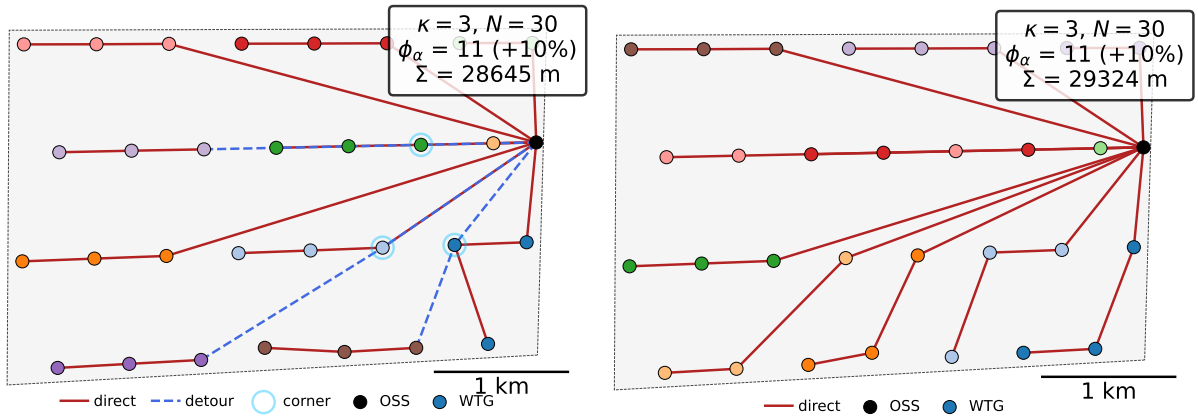


Figure 3.13: OBEW (left) and Global Optimizer (right) layouts for **ormonde** at $\kappa = 3$.

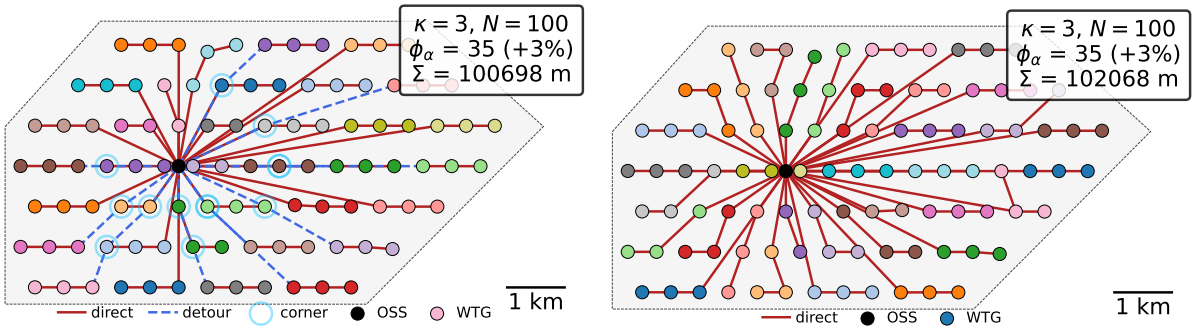


Figure 3.14: OBEW (left) and Global Optimizer (right) layouts for **thanet** at $\kappa = 3$.

The actual total lengths obtained in each of the solutions presented in the figures are available in Table A.1 and Table A.2.

CHAPTER 4
Computational cost

This chapter examines the behavior of the theoretical and empirical computational demand of the proposed heuristics as the problem size changes. The former takes into account the worst case scenarios for each of the algorithms' steps, while the later looks at the execution of the procedures on the OWPP working set.

4.1 Algorithmic complexity

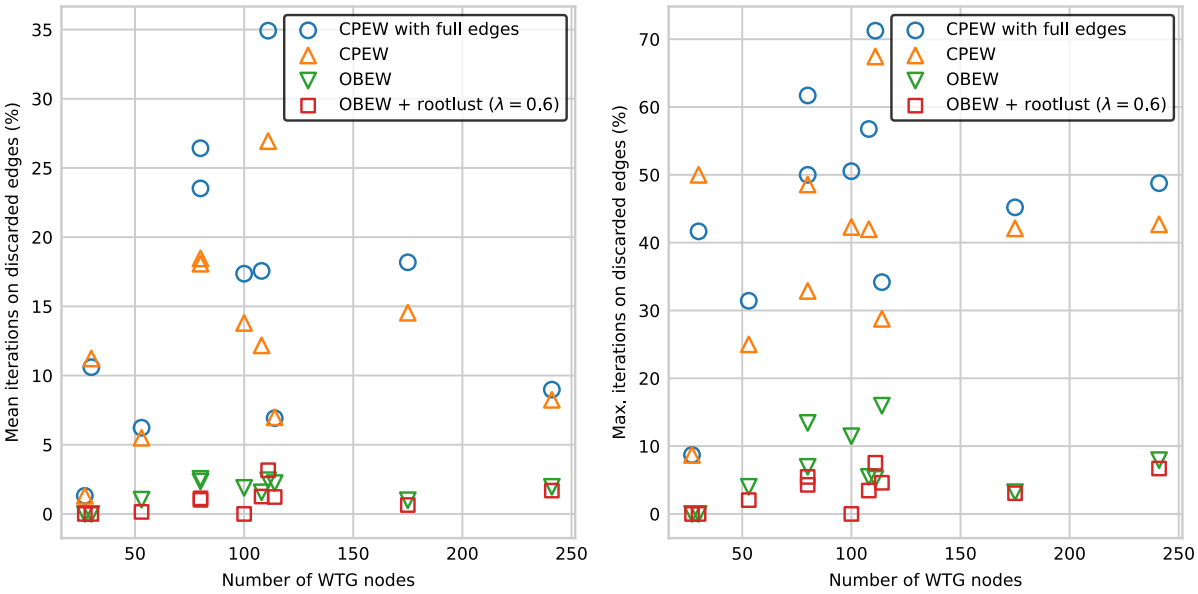
An efficient implementation of the original Esau-Williams heuristic is described in (Kershenbaum, 1974), where the complexity is shown to be $O(N \log N)$. This section examines the complexity of the extended versions of EW presented in **??** – CPEW and OBEW. The Multi-Root extension and the *rootlust* factor do not change the complexity of the algorithms.

The number of iterations of the main loop of both algorithms is considered to be proportional to N. At the best case (i.e. no crossings are detected), one loop iteration is needed for each replacement of a gate edge by a non-gate edge – these happen at most $N - \lceil N/\kappa \rceil$ times. Each detected crossing by a potential edge will add one additional loop iteration, since when an edge is discarded no change is made to the graph.

These algorithms are running with the expanded Delaunay edge set, which is proportional to N. Hence, the worst case of discarded edges has to be bound by the number of edges available to choose from. This is basis for using $O(N)$ as the bound for the main loop iterations. Nevertheless, it is informative to look at the actual iteration counts from the optimization of the working OWPP set.

Figure 4.1 was obtained by running four heuristic variants – CPEW with full edges, CPEW, OBEW and OBEW with rootlust – for all farms with $\kappa \in [2, 15]$. Each plot summarizes the percent fraction of the algorithms' iterations that were spent with discarded edges; one graph shows the mean, while the other shows the maximum across the values of κ for each of the OWPP. The data points are positioned in the x axis according to the number of WTGs in the OWPP that originated them.

The figure conveys two important insights: that the OBEW extension discards significantly less edges than the CPEW extension and that the fraction of discarded edges does not increase with increasing number of WTG. There is a variability in the value of that fraction across the OWPP set, but in the context of complexity analysis, the theoretical $O(N)$ bound seems to hold.



(a) mean **(b)** maximum

Figure 4.1: Percentage of iterations spent on discarded edges, aggregated for each OWPP across $\kappa \in [2, 15]$.

Another information that can be gathered from Figure 4.1 is that the use of expanded Delaunay edges reduced the number of discarded edges, but the more significant reduction was enabled by the OBEW extension. The *rootlust* factor also reduces this metric when compared to pure OBEW, but its effect is less visible due to the scale of the y axis. This is expected, since the more radially oriented the edges are, the less likely the occurrence of crossings.

In the next sections, the complexity $O(\log N)$ is used for one addition or extraction from the priority queue, as discussed in section 2.7. As for tasks that depend on the number of neighbors of a node within the expanded Delaunay edge set, the bound used is 12, which means $O(1)$. This is based on the average number of neighbors in a Delaunay triangulation (6) multiplied by 2 because of the expansion.

4.1.1 Initialization

The initialization procedure (algorithm 3) is very straightforward:

- create expanded Delaunay edge set: $O(N \log N)$
- loop for populating S (N times):
  - add entry to priority queue $O(\log N)$
  - other tasks: $O(1)$

Overall complexity of initialization: $O(N \log N)$

4.1.2 CPEW

Crossing Preventing EW's main loop (as in Algorithm 4). For each iteration ($O(N)$ times):

- check crossings with edges (a list of crossings is available): $O(1)$
- check crossings with gates: $O(N)$
- remove S entry once in case of sub-κ merger (map subtree:entry is known): $O(\log N)$
- remove S entry κ times in case of κ merger (map subtree:entry is known): $O(\log N)$
- loop to update S (just once in most iterations, but never more than 12κ times):
  - find best edge (κ comparisons): $O(1)$
  - add entry to priority queue $O(\log N)$
- extract item from priority queue: $O(\log N)$

Overall complexity of CPEW: $O(N^2)$

4.1.3 OBEW

Structure of Obstacle Bypassing EW's main loop (Algorithm 6) is similar to CPEW's, with the additional check for crossings with detour edges and the planning (Algorithm 5) and implementation of detours (line 15 of Algorithm 6). Only the additional steps will be analyzed here.

- check crossings with detours: $O(N^2)$;
- PLANDETOUR can be called by two mechanisms: loop and recursion
  - loop calls are limited by the maximum number of edges in the solution: (N)
  - recursion depth is limited to 4, each run can make 2 calls, worst case is 31 calls;
  - loop $\times$ recursion limited by $31N$: $O(N)$
- execution of PLANDETOUR:
  - check crossings with edges: $O(N)$
  - other tasks: $O(1)$.
- implement detours (at worst $N/2$ times with, at most, $4 + 1$ edges to add): $O(N)$;

The additional steps of crossings checks in the main loop and within PLANDETOUR are both of complexity $O(N^2)$ and supersede the complexity of the parts in common with CPEW. The overall worst-case complexity of OBEW becomes $O(N^3)$.

4.2 Running time

This section complements the complexity analysis with some empirical data on running times and allows for a qualitative comparison between the heuristics.

The algorithms were implemented in *Python* and executed by the interpreter CPython version 3.9 in a desktop PC with an AMD Ryzen 3 3200G processor. The heuristics uses the Python modules *networkx* for graph data structures and *numpy* for array operations. Just for reference, the global optimization performed by CPLEX was executed in a high performance computing cluster from DTU (this step will not be detailed further as its performance is not within the scope of this work).

The purpose of presenting running time data is only for observing the influence of algorithm choice and OWPP size on the computational cost of the heuristics. The data points' absolute value have little meaning by themselves, for they are hardware-specific and *Python* is an interpreted language (i.e. its performance is lacking when compared with compiled languages).

Figure 4.2 shows the time each algorithm needs to output a solution for a given OWPP and κ. The heuristics rank as expected by the complexity analysis, with the reference CPEW with full edges taking the longest time, CPEW being the fastest and the two OBEW instances ranking close together not far from CPEW. The behavior of running time vs. WTG number, however, is much closer to a linear relationship than the complexity analysis suggests.

Two conjectures about this discrepancy are proposed: a) this analysis is based on worst case scenarios, which may be only triggered by WTG positions that are not useful from a practical perspective; b) the higher exponents tasks within each heuristic represent a very small fraction of the total computations performed by the algorithm, thus they will become noticeable only on much bigger problems. Points *a* and *b* are not mutually exclusive, so a combination of them is also possible. Overall, these data do not challenge the assertion that the proposed heuristics have the important property of polynomial complexity, which makes big problems tractable within reasonable time-frames.

Just as a reference, the exact solutions of the global optimization took from a few minutes for the smaller OWPP to a few hours for the bigger ones. This duration is manageable if only a few optimization runs are needed, but for an integrated optimization approach, which requires the generation of thousands of optimal collection system layouts, a heuristic solution will enable a much faster convergence of the multi-disciplinary objective function. Afterwards, with the design space reduced to a few best projects, an exact global optimum with guaranteed quality may be pursued.

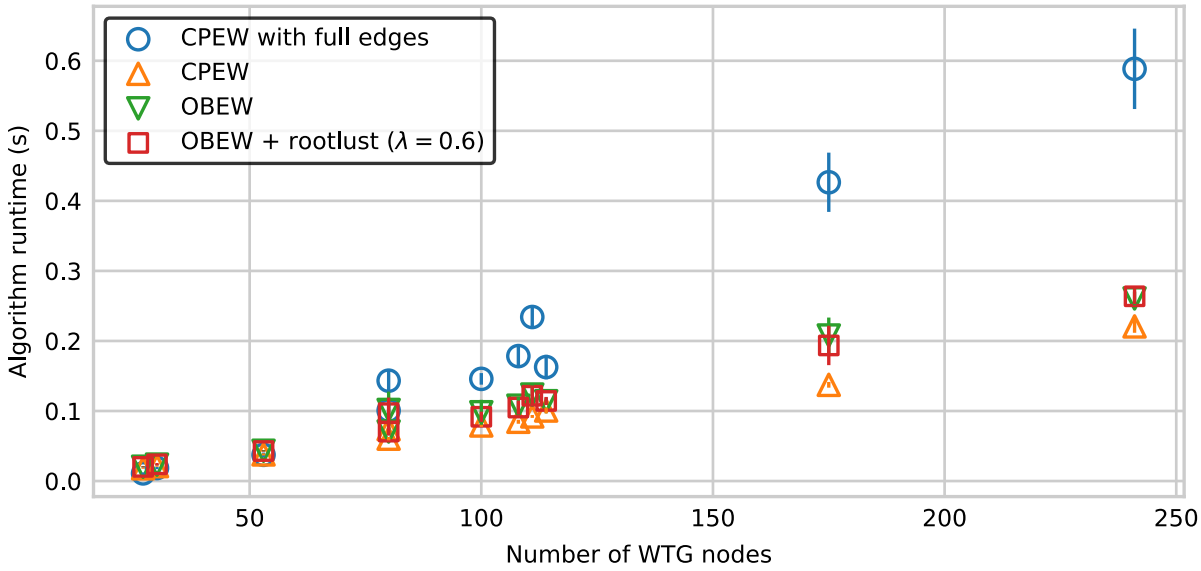


Figure 4.2: Running times for the EW extensions for the OWPP set (error bars are the std. dev. across $\kappa = [2, 15]$).

CHAPTER 5
Conclusion

This work engaged with the optimization of the investment cost of the collection system layout, using the simplifying assumption of a single cable type. This reduces the task to a total cable length minimization problem, which maps to the classic capacitated minimum spanning tree (CMST) problem with an additional constraint of cable crossing interdiction.

Algorithms that build upon the classic Esau-Williams (EW) heuristic were presented and its solutions analyzed and compared to reference ones. The Crossing Preventing EW (CPEW) and the Obstacle Bypassing EW (OBEW) provide feasible solutions by complying to both the cable capacity limits and to the non-crossing constraint while keeping the computational cost polynomial on the number of WTG of the OWPP. The algorithms were applied to a working set of 11 OWPP – 7 actually built ones, 2 proposed and 2 synthetic – with sizes ranging from 27 to 243 WTG. Cable capacities were varied from 2 to 15 WTG. The implemented heuristics reach feasible layouts for all the pairs OWPP-capacity.

The use of Delaunay triangulation for the restriction of edges within EW in order to reduce algorithm complexity was not mentioned in previous literature. The use of detour nodes to improve solution quality was mentioned before, but implemented differently. The *rootlust* bonus factor extends a preexisting idea, but innovates in involving the relative demand of the subtree (i.e. subtree size/κ) in its calculation, making it applicable in different problem instances without the need to calibrate.

CPEW has a worst case complexity of $O(N^2)$ and its layouts are, on average, 0.2% longer than a trivial (and slow) implementation of edge intersection avoidance to the original EW (naive heuristic). OBEW has a worst case complexity of $O(N^3)$ and its solutions are, on average, 1.3% shorter than the naive heuristic. With the introduction of the *rootlust* bonus factor to EW's savings calculation formula, the results become, on average, 1.9% shorter than the naive approach. For comparison, the best optimal estimates (within 1%, obtained with a mixed-integer linear programming solver) are, on average, 4.4% shorter in total length than the naive EW approach.

The heuristics run within a fraction of a second on a regular PC for plants with a few hundred WTG. This speed and the quality of the layouts obtained make them suitable for application within the larger framework of integrated OWPP optimization.

5.1 Future work

Some simplifying assumptions were made in the course of this work that compromise the optimization in more complex OWPP instances. Some of them could be incorporated in the implemented algorithm with additional effort. The most relevant ones would be the consideration of multiple cable types with their respective capacity limit and linear cost, as well as the avoidance of forbidden areas in the sea floor. The latter one seems like a relatively simple extension of the OBEW algorithm.

Another line for further development is the improvement of the internal mechanisms of the OBEW heuristic. One change that could enhance solution quality is to make detour calculation a part of the edge search procedure. This way, prospective edges would chosen and prioritized according to their actual savings (i.e. discounted the detour cost). The savings calculation formula may also be explored further, as its effect on the morphology of EW-based layouts can be further tuned to make them more similar to the globally optimal ones.

Yet another possible research goal is to integrate the proposed heuristics with more sophisticated meta-heuristics or with exact optimization methods as a warm-start for branch-and-cut solvers.

APPENDIX A
Additional data

Figure A.1: OWPPs used in this work in the format **handle**: number of WTG.

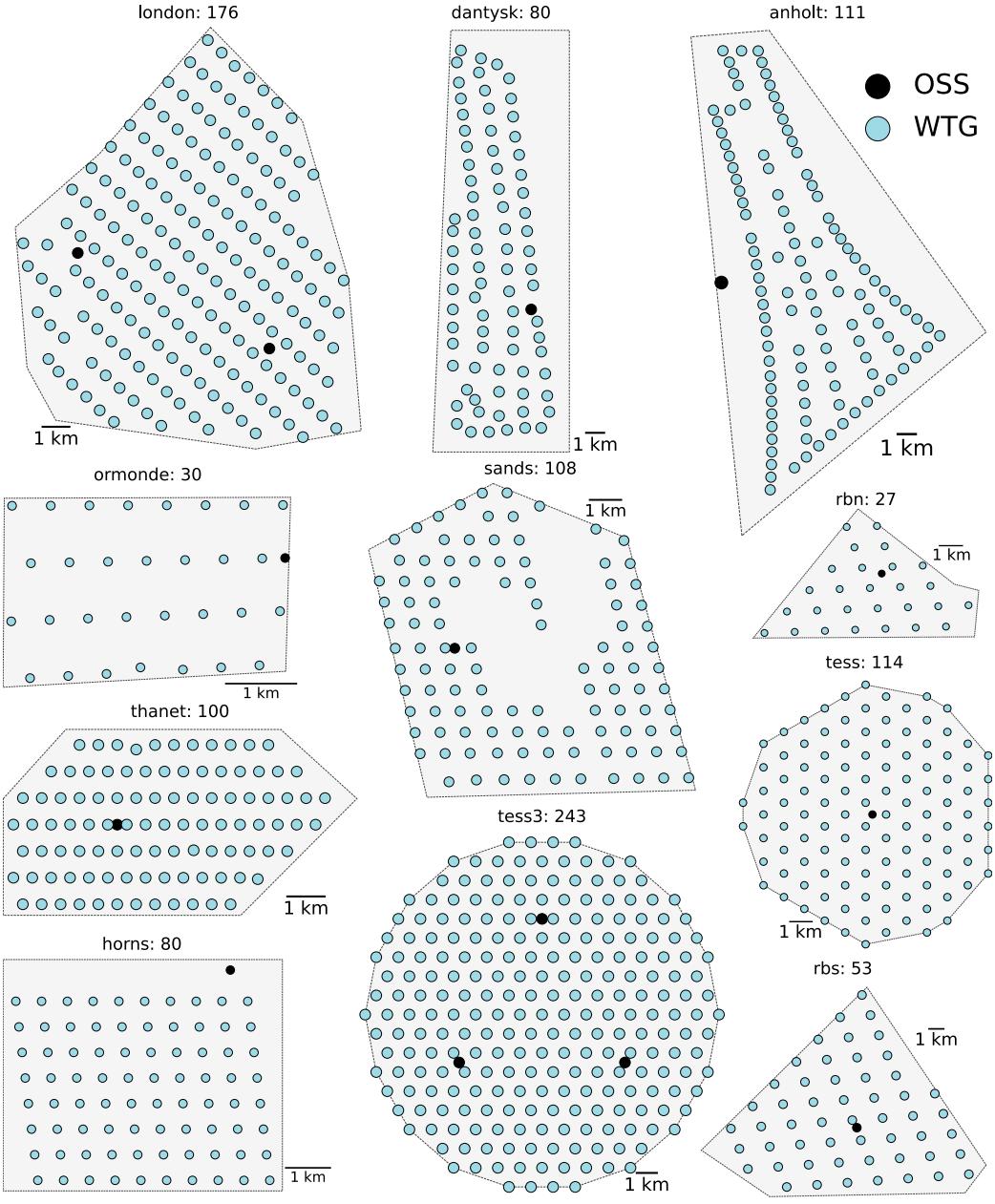


Figure A.2: Quality of the solutions for all OWPP sites and values of κ (baseline is CPEW with full edges).

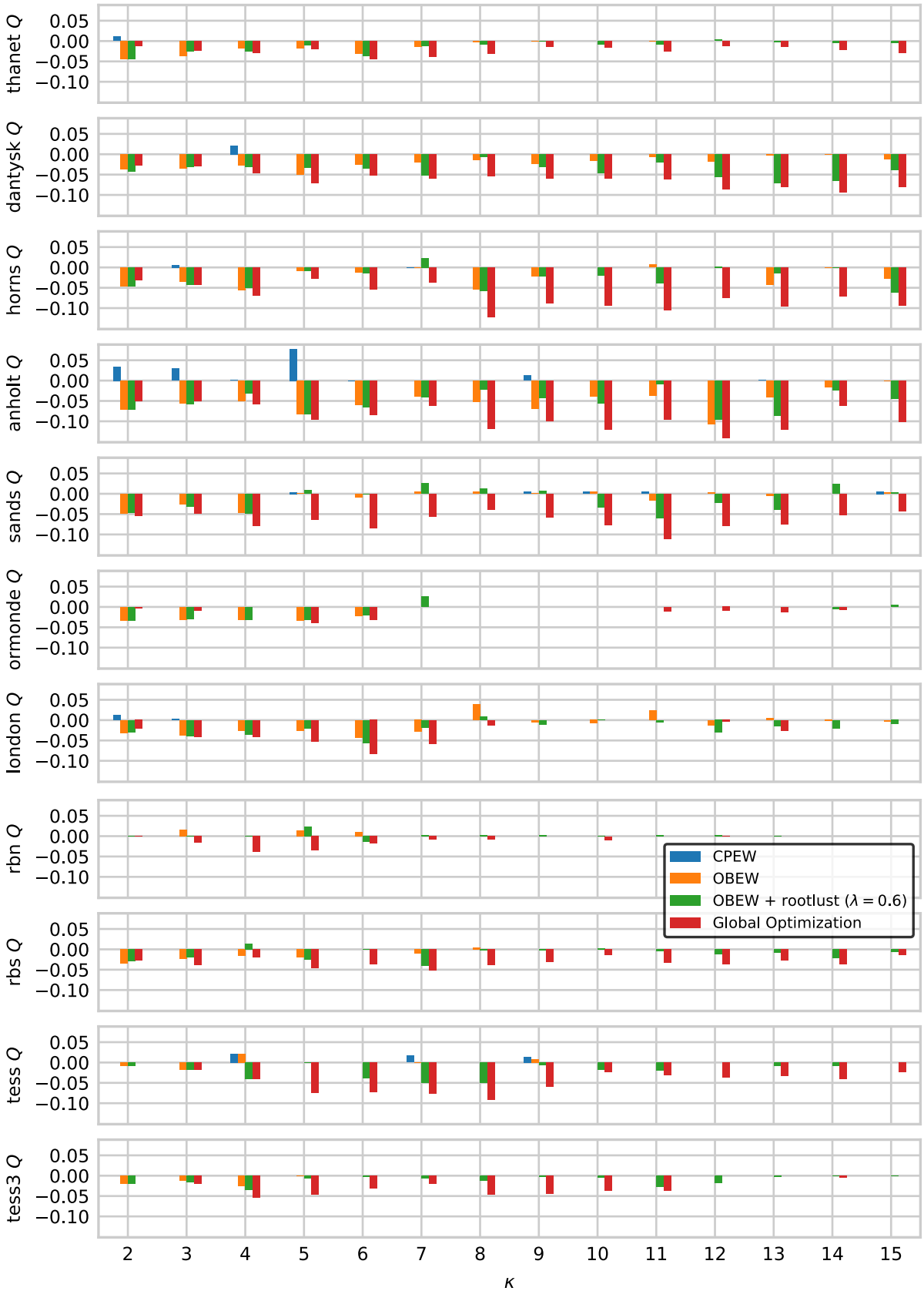


Table A.1: Total length (m) obtained with the heuristics (**thanet**-**ormonde**).

(a) thanet

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	143152	135251	135256	139706
3	104643	100698	101915	102067
4	85891	84301	83745	83379
5	74412	73132	73678	72952
6	69577	67374	67002	66434
7	63908	62992	63073	61454
8	59987	59821	59472	58072
9	56428	56452	56429	55632
10	54494	54494	54076	53604
11	54338	54309	53842	52935
12	53124	53124	53305	52519
13	53052	53052	52925	52282
14	52745	52745	52508	51626
15	52745	52745	52508	51172

(b) dantysk

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	265768	256131	254537	258472
3	192505	185927	186309	186965
4	160739	152936	152424	150107
5	138628	131574	134114	128883
6	120659	117664	116354	114424
7	111723	109534	105885	105055
8	102636	101241	102051	97072
9	98844	96589	95825	93039
10	94667	93062	90226	88952
11	90355	89709	88621	84861
12	89480	87909	84559	81738
13	86847	86679	80611	79813
14	84843	84906	79285	76909
15	83197	82169	80027	76516

(c) horns

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	165152	157561	157561	159938
3	120651	115721	114887	114842
4	98971	93425	93952	92186
5	82046	81304	81304	79828
6	74568	73601	73525	70608
7	66559	66559	68103	64103
8	67222	63567	63374	59003
9	61434	60130	60145	55976
10	58884	58884	57713	53410
11	59245	59632	56912	53048
12	54979	54979	55098	50830
13	55640	53264	54809	50281
14	52929	52890	52890	49151
15	54112	52641	50746	49028

(d) anholt

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	473959	425419	425806	434938
3	333232	305412	305215	307601
4	259285	245771	250868	243877
5	245937	209453	209361	206265
6	196292	184376	183590	179735
7	174414	167522	167424	163795
8	167602	158857	163988	147645
9	156288	143911	147762	139203
10	148072	142438	139858	130374
11	135136	130183	133918	122180
12	138539	123801	125193	119065
13	129552	124114	118322	113833
14	118906	116974	115985	111616
15	120303	120304	114992	108125

(e) sands

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	249452	237228	237701	235908
3	181559	177049	175939	172952
4	153403	146206	145957	141460
5	131251	130896	131932	122476
6	121088	120024	120912	110983
7	109263	109798	112014	103170
8	102364	102870	103598	98437
9	100668	100273	100867	94451
10	99492	99510	95762	91457
11	99944	97798	93601	88469
12	94059	94354	92072	86681
13	92041	91578	88417	85176
14	87813	87813	89914	83223
15	86413	86155	86172	82324

(f) ormonde

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	39285	37964	37989	39145
3	29563	28645	28677	29324
4	24620	23857	23857	24780
5	22292	21546	21577	21429
6	20096	19650	19682	19474
7	18039	18039	18501	18039
8	16921	16921	16921	16921
9	16921	16921	16921	16921
10	16921	16921	16921	16921
11	16921	16921	16921	16739
12	16921	16921	16921	16760
13	16921	16921	16921	16703
14	16844	16844	16757	16733
15	16408	16408	16485	16408

Table A.2: Total length (m) obtained with the heuristics (**london**-**tess3**).

(a) london

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	384252	367801	368612	371811
3	286171	274935	274309	273692
4	235923	229692	227441	226125
5	206252	200924	202100	195552
6	194402	186104	183641	178137
7	179178	174069	175941	168680
8	163423	169718	164962	161489
9	155038	154273	153372	–
10	150991	149839	150965	–
11	146802	150409	146033	–
12	147665	145898	143140	147080
13	144086	144644	141922	140399
14	142276	141997	139421	–
15	138325	137837	136969	–

(b) rbn

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	40600	40600	40614	40544
3	33843	34389	33836	33298
4	31079	31079	31126	29860
5	29444	29871	30131	28423
6	28245	28530	27863	27716
7	27385	27385	27480	27168
8	26801	26801	26896	26552
9	26552	26552	26615	26552
10	26449	26449	26481	26201
11	26201	26201	26264	26201
12	26201	26201	26264	26156
13	25805	25805	25837	25805

(c) rbs

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	148943	143761	144658	144927
3	116778	114001	114419	112191
4	98750	97129	100146	96763
5	92909	90983	90541	88555
6	86845	86845	86910	83608
7	85807	84983	82238	81373
8	83069	83522	82876	79779
9	80840	80840	80641	78342
10	78949	78949	79216	77816
11	79418	79418	78984	76847
12	79366	79366	78328	76461
13	78120	78120	77418	75934
14	78589	78589	76881	75684
15	76817	76817	76354	75684

(d) tess

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	248179	246068	246068	–
3	190328	186860	186860	186885
4	167039	167039	157102	156969
5	151084	151084	151289	139734
6	139532	139532	134106	129458
7	137995	135860	128780	125185
8	134106	134106	127462	121802
9	128446	127846	125912	119283
10	120903	120903	118706	118023
11	120753	120753	118251	116942
12	120219	120219	120219	115860
13	119665	119665	118557	115778
14	119665	119665	118557	114778
15	116942	116942	116942	114251

(e) tess3

κ	CPEW	OBEW	$\lambda = 0.6$	GO
2	479594	470181	470181	–
3	369822	365313	363672	362392
4	324262	316009	313133	306837
5	289191	288693	287341	275707
6	268312	268312	267340	260145
7	259743	259743	258187	254498
8	259072	259072	256015	247119
9	255430	255430	254765	244211
10	251685	251685	250555	242302
11	250506	250506	243478	241070
12	248147	248147	243478	–
13	241070	241070	240424	241102
14	241070	241070	240633	239865
15	241070	241070	240633	–

Bibliography

Ahuja, R. K., Orlin, J. B., & Sharma, D. (2003). A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Operations Research Letters*, *31*(3), 185–194. https://doi.org/10.1016/S0167-6377(02)00236-5

Amberg, A., Domschke, W., & Voß, S. (1996). Capacitated minimum spanning trees: Algorithms using intelligent search. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.3749&rep=rep1&type=pdf

Bauer, J., & Lysgaard, J. (2015). The offshore wind farm array cable layout problem: A planar open vehicle routing problem [Publisher: [Operational Research Society, Palgrave Macmillan Journals]]. *The Journal of the Operational Research Society*, *66*(3), 360–368. Retrieved May 6, 2022, from https://www.jstor.org/stable/24505324

BVG Associates. (2019). *Wind farm costs – guide to an offshore wind farm*. Retrieved June 15, 2022, from https://guidetoanoffshorewindfarm.com/wind-farm-costs

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points [Publisher: INFORMS]. *Operations Research*, *12*(4), 568–581. Retrieved June 23, 2022, from https://www.jstor.org/stable/167703

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271. Retrieved May 22, 2022, from https://ir.cwi.nl/pub/9256/9256D.pdf

Esau, L. R., & Williams, K. C. (1966). On teleprocessing system design, part II: A method for approximating the optimal network. *IBM Systems Journal*, *5*(3), 142–147. https://doi.org/10.1147/sj.53.0142

Fischetti, M., & Pisinger, D. (2016). Inter-array cable routing optimization for big wind parks with obstacles. *2016 European Control Conference (ECC)*, 617–622. https://doi.org/10.1109/ECC.2016.7810357

Fotedar, S. (2018, June 13). *Optimization of the offshore wind inter-array cable layout problem using heuristic based algorithms* (Master's thesis). University of Bergen. Retrieved June 17, 2022, from https://bora.uib.no/bora-xmlui/handle/1956/18063

Katsouris, G. (2015). *Infield cable topology optimization of offshore wind farms* (Master's thesis). Retrieved June 18, 2022, from https://repository.tudelft.nl/islandora/object/uuid%3A9f313e13-4570-48f8-8aff-3eef35bbad99

Kershenbaum, A. (1974). Computing capacitated minimal spanning trees efficiently [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230040403]. *Networks*, *4*(4), 299–310. https://doi.org/10.1002/net.3230040403

Kirk, D. (Ed.). (1994, January 19). *Graphics gems 3* (1st edition). Morgan Kaufmann. Retrieved June 17, 2022, from http://www.realtimerendering.com/resources/GraphicsGems/

KIS-ORCA. (2022). *Downloads | kis-orca*. Retrieved June 14, 2022, from https://kis-orca.org/downloads

Klein, A., & Haugland, D. (2017). Obstacle-aware optimization of offshore wind farm cable layouts. *Annals of Operations Research*, *272*(1), 373–388. https://doi.org/10.1007/s10479-017-2581-5

Öncan, T., & Altınel, İ. K. (2009). Parametric enhancements of the esau–williams heuristic for the capacitated minimum spanning tree problem. *Journal of the Operational Research Society*, *60*(2), 259–267. https://doi.org/10.1057/palgrave.jors.2602548

Papadimitriou, C. H. (1978). The complexity of the capacitated tree problem [_eprint: https://onlinelibrary *Networks*, *8*(3), 217–230. https://doi.org/10.1002/net.3230080306

Perez-Moreno, S. S., Dykes, K., Merz, K. O., & Zaaijer, M. B. (2018). Multidisciplinary design analysis and optimisation of a reference offshore wind plant [Publisher: IOP Publishing]. *Journal of Physics: Conference Series*, *1037*, 042004. https://doi.org/10.1088/1742-6596/1037/4/042004

Pérez-Rúa, J.-A. (2022). *A-framework-for-simultaneous-design-of-wind-turbines-and-cable-layout-in-offshore-wind* (Source code). GitHub repository. https://github.com/japerezrua/A-Framework-for-Simultaneous-Design-of-Wind-Turbines-and-Cable-Layout-in-Offshore-Wind

Pérez-Rúa, J.-A., & Cutululis, N. A. (2019). Electrical cable optimization in offshore wind farms—a review. *IEEE Access*, *7*, 85796–85811. https://doi.org/10.1109/ACCESS.2019.2925873

Pérez-Rúa, J.-A., & Cutululis, N. A. (2022). A framework for simultaneous design of wind turbines and cable layout in offshore wind [Publisher: Copernicus GmbH]. *Wind Energy Science*, *7*(2), 925–942. https://doi.org/10.5194/wes-7-925-2022

Peréz-Rúa, J.-A., Minguijón, D. H., Das, K., & Cutululis, N. A. (2019). Heuristics-based design and optimization of offshore wind farms collection systems. *Journal of Physics: Conference Series*, *1356*(1), 012014. https://doi.org/10.1088/1742-6596/1356/1/012014

Pillai, A., Chick, J., Johanning, L., Khorasanchi, M., & de Laleu, V. (2015). Offshore wind farm electrical cable layout optimization [Publisher: Taylor & Francis _eprint: https://doi.org/10.1 *Engineering Optimization*, *47*(12), 1689–1708. https://doi.org/10.1080/0305215X.2014.992892

Shamos, M. I. (1978). *Computational geometry.* (Doctoral dissertation). Yale University. Retrieved June 17, 2022, from http://euro.ecom.cmu.edu/people/faculty/mshamos/1978ShamosThesis.pdf