Solver-free heuristics to retrieve feasible points for offshore wind farm collection system

Juan-Andrés Pérez-Rúa

DTU Wind Energy, Roskilde, Denmark

ABSTRACT
A set of solver-free heuristics for the offshore wind collection system problem is presented. Currently, algorithms of this type are not able to cope with relevant constraints when explicitly minimizing for investment cost. The first algorithm is a two-step decision process of which the output is a tree network satisfying cable thermal limits, but vulnerable to violate planarity constraints. A cable crossings repair heuristic is then introduced in order to fix infeasible points from the first heuristic. Finally, the negative cycle cancelling refining heuristic takes over to improve feasible points. The latter iteratively swaps connections, intending to find cycles with negative costs to reduce investment expenditure. The applicability of the workflow is empirically demonstrated through a set of large-scale real-world offshore farms. Results indicate that: (i) feasible points can be retrieved in a computing time of seconds; and (ii) warm starting can help global optimization solvers to converge significantly faster for hard problems.

1. Introduction
The offshore wind industry has experienced a steep evolution during the last 12 years: from 2009 to 2019 it moved from being 1% to 10% of global wind installations (GWEC 2020). This is a strong proof of the maturity of the technology; nevertheless, there is room for technical improvement.

The balance of plant supports the operation of Wind Turbines (WTs) in Offshore Wind Farms (OWFs). It encompasses submarine cables, offshore substations (OSSs), converters, foundations, structures and control equipment. It represents around 30% of the full levelized cost of energy, with electrical systems being around 15% (ORE Catapult 2020).

The OWF collection system problem is defined as the design of the medium voltage network to interconnect WTs towards OSSs. It has been studied with particular attention over the past 10 years (Pérez-Rúa and Cutululis 2019). Finding the global optimum of this problem is generally NP-hard (Jothi and Raghavachari 2005).

Radial networks have been the prime subject of study in the context of computational optimization, but new methodologies are being proposed as well to cope with the design of networks with electrical redundancy as closed-loop topology with deterministic (Klein and Haugland 2020) and probabilistic (Pérez-Rúa et al. 2022) uncertainty modelling techniques. Mathematical programming with probabilistic incorporation of uncertainty allows formulating a compact model where investment costs and the reliability aspect can be optimized simultaneously. However, two major challenges arise: very low tractability and high dependence on data availability for trustable outcomes (Pérez-Rúa et al. 2022).
This article focuses on radial networks as, apart from being fully functional, cost-effective alternatives themselves (ESCA, SEAFISH, and RenewableUK 2021), they can also serve as a baseline for further refining processes to incorporate redundancies (if needed) based on case-by-case analysis. In general, three fundamental clusters of methods for tackling the OWF collection system problem are: global optimization, metaheuristics and heuristics.

Global optimization encompasses several modelling options, such as Binary Programming (BIP) (Cerveira et al. 2016), Mixed Integer Linear Programming (MILP) (Pérez-Rúa et al. 2019; Fischetti and Pisinger 2018; Fagerfjäll 2010) and MILP with decomposition techniques for stochastic programming (Lumbreras, Ramos, and Cerisola 2013). The main advantage of these approaches is the possibility of implementing exact solvers based on the branch-and-cut method; theoretically, being able to solve a problem to optimality supporting all the common engineering constraints. However, as pointed out by Lumbreras, Ramos, and Cerisola (2013) and Fischetti and Pisinger (2018), challenges emerge for large-scale problems, for which it becomes hard to obtain even feasible points, and computing time and memory requirements tend to be unpredictably large. While early works focused on developing analytical formulations for small-scale problems such as flow-based (Fagerfjäll 2010; Pillai et al. 2015) and hop-indexed (Cerveira et al. 2016) formulations, recent advancements in this category have dealt with increasing tractability for large problems by means of embedding global optimization within algorithmic frameworks, as in Fischetti and Pisinger (2018) and Pérez-Rúa et al. (2019). This has given place to new hybrid methods known as matheuristics. In spite of that, the intrinsic limitations from search tree optimization are still present.

In this context, solver-free methods are defined by approaches with alternative sets of policies to the branch-and-cut method, like stochastic metaheuristics, such as genetic algorithms (González-Longatt et al. 2012) or swarm optimization (Hou, Hu, and Chen 2016), and deterministic heuristics, such as Prim (Hou, Hu, and Chen 2015), Clarke and Wright (Bauer and Lysgaard 2015) or Esau–Williams (Sanchez Perez-Moreno et al. 2018). In general, metaheuristics utilize stochastic operators for constructing and improving a set of solutions iteratively. Therefore, they are likely to be able to obtain feasible points of good quality in a long computing time for large problems, but they lack optimality and worst-case time certificates. On the contrary, heuristics are provably fast, as they use a sequence of inexpensive decisions to (hopefully) construct a solution in a bounded time.

This article focuses on solver-free heuristics owing to their fast processing and capability of being combined with other methods because, during the pre-feasibility stage, many OWF collection system designs must be carried out, accounting for different parameters with associated uncertainties. Likewise, when provided with a feasible point, modern branch-and-cut solvers can improve their operation by implementing a warm-starting strategy. For industrial applications, heuristics can help by reducing investment and time costs stemming from expensive branch-and-cut solvers.

Minimum Spanning Tree (MST) algorithms have been almost entirely the chosen method within the heuristic category. An heuristic for designing a collection system without branching at WTs is proposed in Bauer and Lysgaard (2015). It supports crossing constraints but minimizes accumulated cable length without selecting cable type. A modification of the Prim algorithm to account for cable type selection is presented in Hou, Hu, and Chen (2015) for designing a tree with branching, but it relaxes the crossing restrictions. Other works neglect both investment minimization and crossings constraints, as in Dutta and Overbye (2012), where the Dijkstra algorithm is implemented to minimize total length, and Sanchez Perez-Moreno et al. (2018), Wade, Pereira, and Wade (2019) and Amaral and Castro (2017), where the equivalent is done through an Esau–Williams heuristic.

To the best of the author’s knowledge, only the sequence of works Gritzbach et al. (2018), Gritzbach et al. (2019), Gritzbach, Wagner, and Wolf (2020) have proposed heuristics to minimize the total investment of the wind farm collection system iteratively by applying minimum cost flow theory. Throughout these articles, different strategies are proposed to increase the likelihood of obtaining a global minimum within very short computing times. Nevertheless, those algorithms do not explicitly support the basic engineering constraints of an OWF collection system.
While fast, the heuristics proposed in the literature to address this problem present at least one of these disadvantages: (i) they struggle to satisfy the typical engineering constraints of the OWF collection system problem (non-redundant topology and no cable crossings); and (ii) they minimize for total length, and not directly the initial investment. The main contributions of this article are: (i) to propose solver-free heuristics that enforce the satisfiability of relevant constraints for the problem’s nature while minimizing investment; and (ii) to improve the performance of a branch-and-cut solver when implementing a global optimization (MILP) model.

2. Problem definition and modelling

The aim is to design the collection system electrical network for an OWF, i.e. to interconnect the $n_T$ WTs to the available OSSs, $n_S$, using a list $C$ of cables available, while minimizing the total investment cost without electrical redundancies and allowing branching at WTs.

Let the OSSs and WTs be defined by the sets $S = \{1, \ldots, n_S\}$ and $T = \{1 + n_S, \ldots, n_S + n_T\}$, respectively. The full set of points is denoted as $N = S \cup T$, i.e. each point is assigned a unique natural number identifier. The Euclidean distance between points $i \in N$ and $j \in N$, is denoted as $d_{ij}$. The weighted complete directed graph $G = (N, A, D)$ gathers all relevant graph-related parameters, where $N$ represents the vertex set, $A$ the set of arcs arranged as a pair-set $a \in A : a = (i, j)$, expressing the power transfer direction from $i$ to $j$ for non-negative flow, and $D$ the set of distances $d_a$.

Let $n_C$ cables be available. The capacity of a cable $c \in C$ is $q_c$ measured in terms of the maximum number of WTs supported upstream (with respect to flow towards the OSSs). Furthermore, let $Q$ be the set of thermal capacities sorted as in $C$ (non-decreasing order). Each cable type $c$ has a cost per unit length, $w_c \in W$, in such a way that $C$, $Q$ and $W$ are all co-monotonic.

Generally, a standard feasible collection system design includes the following engineering constraints.

[C1] (Hard) A tree topology must be enforced. This means that there must be only one electrical path from each WT towards an OSS.
[C2] (Hard) The thermal capacity of cables must not be exceeded.
[C3] (Hard) Cables must not lie over each other (no cable crossings) due to practical installation aspects.
[C4] (Soft) The number of main feeders, i.e. cables reaching an OSS directly, might be limited.

Constraints [C1] and [C2] define a canonical computer science problem, known as the Capacitated Minimum Spanning Tree (C-MST), an NP-hard problem (Jothi and Raghavachari 2005). On top of the previous two constraints, planarity constraint [C3] is forced owing to practical limitation aspects present during the construction stage of this type of project. Finally, spatial constraint [C4] is generally not binding for a large enough maximum capacity $Q = \max Q$; therefore, in most cases it is relaxed as in this article.

3. Solver-free heuristics

The proposed framework containing the algorithms presented in this section is schematized in Figure 1. To tackle a given problem instance, the Two-Step Heuristic (TSH) is first run in order to get an initial point, which is then evaluated for the crossing constraint [C3]. Secondly, in case these are not satisfied, the greedy algorithm Cable Crossings Repair Heuristic (CCRH) is called, seeking to eliminate those invalidities promptly and hoping for good empirical results. Lastly, the retrieved feasible point is intended to be refined by means of the Negative Cycle Cancelling Refining Heuristic (NCCRH).
3.1. The Two-Step Heuristic (TSH)

The first step of this algorithm consists in determining the topology of the network based on accumulated cable length minimization. In this work, a classic C-MST framework (Kershenbaum and Chou 1974) with the Esau–Williams mechanism ruling the iteration process is implemented. This algorithm supports constraints [C1] and [C2]; however, it usually fails when introducing [C3].

The second step calculates the number of upstream WTs connected through every edge after the first step. This is followed up with the selection of the cheapest cable type for each connection. For a detailed description of the TSH, see Pérez-Rúa and Cutululis (2022). The final output is a matrix of connections $T$, where each row represents an edge, with the first two columns defining the connected nodes, and the last one for the cable type selected.

3.2. Cable Crossings Repair Heuristic (CCRH)

After the TSH, if [C3] is violated, the CCRH is implemented with input $T$. The pseudocode is presented in Algorithm 1. The algorithm’s mechanism is to swap infeasible edges with others that do not introduce violations to the design in a sequential deterministic manner. The process starts in line 5, where a list of edges $Crossings$ with at least one crossing is generated, each element being defined as a tuple of edge identifier and crossings number.

After sorting out $Crossings$ for prioritizing the elimination of the largest number of crossings at once, in line 6, the algorithm is successfully terminated in line 8 as no cables cross with each other, returning a crossing-free matrix $T$. On the contrary, if after exhausting all sequences of trials there is at least one crossing, the process is stopped in line 12, with the outcome stamped as infeasible.

In line 14, an edge from $Crossings$ is fetched, which is then temporarily eliminated from the network in line 15, creating a matrix $T_p$. In the next two lines, those nodes removed from the network due to edge deletion are stored in $Nodes$, and a candidate edges list $Candidates$ to integrate them back into the system is created, respectively. $Candidates$ cannot contain any potential edge overlapping with the existing topology in $T_p$.
Algorithm 1: Pseudocode of the cable crossings repair heuristic for the offshore wind collection system problem

1. Get edge matrix, $T$
2. $pot_{el} \leftarrow 0$
3. while True do
4.   if $pot_{el} == 0$ then
5.     Obtain list of edges with at least one crossing in $T$, Crossings
     (An element $l \in$ Crossings is a tuple $(e, \eta)$, where $e \in T$ is an edge, and $\eta > 0$ the number of crossings with it)
6.   end if
7.   Sort elements in Crossings by non-increasing order of number of crossings, $\eta$
8.   if len(Crossings) == 0 then
9.     Break. All crossings eliminated
10. end if
11. if $pot_{el} == \text{len}(\text{Crossings})$ then
12.     Break. Not all crossings eliminated
13. end if
14. Eliminate $\leftarrow \text{Crossings}[pot_{el}][0]$
15. $T_p \leftarrow \text{del}(T, \text{Eliminate})$
16. Find nodes out of the tree $T_p$, Nodes
17. Find candidate edges (sorted in non-decreasing order of length) connecting to Nodes that do not cross with any edge in $T_p$, Candidates
18. $pot_{add} \leftarrow 0$
19. while True do
20.   infeasible $\leftarrow \text{False}$
21.   $T_p \leftarrow \text{add}(T_p, \text{Eliminate}, \text{Candidates}[pot_{add}])$
22.   if Not satisfied $[\text{C1}]$ and $[\text{C2}]$ on $T_p$ then
23.     infeasible $\leftarrow \text{True}$
24.     $T_p \leftarrow \text{del}(T_p, \text{Eliminate})$
25.     $pot_{add} + = 1$
26.   end if
27.   if $pot_{add} == \text{len}(\text{Candidates})$ then
28.     $pot_{el} + = 1$
29.     Break. Need to find new candidates
30. end if
31. else
32.     Get new tree with new concatenated edge from $T_p$, as $T$
33. end if
34. end while
35. end while
36. Return new edges matrix, $T$

Lines 19 to 35 are for the inner loop in charge of exploiting the list Candidates. Line 21 temporarily incorporates into $T_p$ the candidate edge Candidates$[pot_{add}]$ in the index Eliminate, then in line 22 it is assessed if constraints [C1] and [C2] are respected in the tentatively formed network. If that is not the case, the network is infeasible, and therefore in line 24 the candidate edge Candidates$[pot_{add}]$ is disconsidered. The next candidate edge indexed by potadd is retrofitted to $T_p$, repeating the previous examination (line 25). When all alternatives in Candidates are considered, a new set of them must be computed by exploring Crossings and having a new potential to eliminate (the inner while loop is broken, line 28). The edge is only permanently eliminated (from line 15) if constraints [C1] and [C2] are satisfied, when this is swapped with edge Candidates$[pot_{add}]$ (line 31). The inner while loop is interrupted, and a new list Crossings is created, restarting the traversing counters, potel and potadd.
Algorithm 2: Pseudocode of the classic negative cycle cancelling refining heuristic

1. Find a feasible point, \( \Lambda \)
2. while There exists a negative cost cycle in \( \mathcal{G}_\Lambda \) do
3. Find cycle with negative cost in \( \mathcal{G}_\Lambda \) based on \( \mathcal{P}_\Lambda, \mathcal{L} \)
4. Find \( \Delta = \min_{a \in \mathcal{L}} u_{\lambda a} \)
5. Push \( \Lambda \) on \( \mathcal{L} \) with \( \Delta \) units
6. end while

3.3. Negative Cycle Cancelling Refining Heuristic (NCCRH)

Given a feasible network \( T \), the NCCRH is called to decrease investment cost. A detailed explanation of this heuristic is deployed in this section.

3.3.1. Background

The Negative Cycle Cancelling Algorithm (NCCA), also known as the augmenting cycle method, was originally derived to solve a standard network problem, the Minimum Cost Flow (MCF) problem (Ahuja et al. 1993). The classic version of the MCF problem consists in supplying the sinks from the sources by a flow \( \Lambda \) in the cheapest possible way, given a directed graph \( \mathcal{G}_c = (\mathcal{V}_c, \mathcal{A}_c, \mathcal{D}_c, \mathcal{U}_c, \mathcal{P}_c) \), where \( \mathcal{V}_c \) is the nodes set, each with a demand \( \mathcal{D}_c \), \( \mathcal{A}_c \) the arcs set \( (a \in \mathcal{A}_c) \), \( \mathcal{D}_c \) the lengths set for all arcs \( (d_{ca} \text{ length of arc } a) \), \( \mathcal{U}_c \) the capacities set for all arcs \( (u_{ca} \text{ capacity of arc } a) \), \( \mathcal{P}_c \) the costs set per unit of flow for all arcs \( (p_{ca} \text{ linear cost of arc } a) \). Formally, the problem can be formulated as a linear program as

\[
\begin{align*}
\min & \quad \sum_{a \in \mathcal{A}_c} p_{ca} \cdot \lambda_a \\
\text{s.t.} & \quad f_{\Lambda}(i) = b_i \quad \forall \, i \in \mathcal{V}_c \\
& \quad 0 \leq \lambda_a \leq u_{ca} \quad \forall \, a \in \mathcal{A}_c.
\end{align*}
\]

The flow balance equation for node \( i \in \mathcal{V}_c \) in (2) is given as

\[
f_{\Lambda}(i) = \sum_{a_i \in \delta^{-}(i)} \lambda_{a_i} - \sum_{a_j \in \delta^{+}(i)} \lambda_{a_j},
\]

where \( \delta^{-}(i) \) and \( \delta^{+}(i) \) are the incoming and outgoing arcs to \( i \), respectively. In (3), \( \lambda_a \) is a continuous variable representing the flow through \( a \in \mathcal{A}_c \).

By exploiting the complementary slackness conditions of the problem from (1) to (3) and its dual, Algorithm 2 guarantees the optimum solution (see Ahuja et al. [1993] for the proof of correctness).

A feasible flow \( \Lambda \) satisfying (2) and (3) is obtained in line 1 of Algorithm 2, for example by solving a maximum flow problem (Ahuja et al. 1993). The residual graph \( \mathcal{G}_\Lambda \), indicating how flow excess can be moved in \( \mathcal{G}_c \) given the present flow \( \Lambda \), is obtained in line 2.

The definition of \( \mathcal{G}_\Lambda \) is presented in (4), where the nodes set \( \mathcal{V}_\Lambda \) is equal to the original graph joined with the fictitious root node \( i_r \); the arcs set \( \mathcal{A}_\Lambda \) is defined as the arcs of the original graph where flow is lower than their capacity, along with the inverse arcs \( (\bar{a}) \) of the original graph where flow is strictly greater than zero, plus arcs from the fictitious root node \( i_r \) to all nodes.

\[
\begin{align*}
\mathcal{G}_\Lambda &= (\mathcal{V}_\Lambda, \mathcal{A}_\Lambda, \mathcal{U}_\Lambda, \mathcal{P}_\Lambda) \\
\mathcal{V}_\Lambda &= \mathcal{V}_c \cup \{i_r\} \quad \text{(fictitious root node)} \\
\mathcal{A}_\Lambda &= \{a : a \in \mathcal{A}_c \land \lambda_a < u_{ca}\} \cup \{\bar{a} : \bar{a} \in \mathcal{A}_c \land \lambda_{\bar{a}} > 0\} \cup \{a : a = (i_r, j) \land j \in \mathcal{V}_c\} \\
\mathcal{U}_\Lambda &= \{u_{ca} - \lambda_a : a \in \mathcal{A}_c \land \lambda_a < u_{ca}\} \cup \{\lambda_{\bar{a}} : \bar{a} \in \mathcal{A}_c \land \lambda_{\bar{a}} > 0\} \cup \{\infty : a = (i_r, j)\} \\
\mathcal{P}_\Lambda &= \{p_{ca} : a \in \mathcal{A}_c \land \lambda_a < u_{ca}\} \cup \{-p_{ca} : \bar{a} \in \mathcal{A}_c \land \lambda_{\bar{a}} > 0\} \cup \{0 : a = (i_r, j)\}.
\end{align*}
\]
\( U_A \) is called the residual capacities set, composed respectively of the remaining capacity of the arcs with flow lower than their capacity, of the flow of arcs when it is greater than zero, and of infinite capacity for the arcs rooted at \( i_r \). Finally, for the costs set \( P_A \), cost is equal for arcs with flow lower than capacity, negative in the inverse arcs when flow in the original graph is greater than zero, plus the costs of the arcs rooted at \( i_r \) being zero.

In line 3 of Algorithm 2, a negative cost cycle \( \mathcal{L} \) (if any) must be found with initial point \( i_r \). This is possible by means of the shortest path algorithm, Bellman–Ford (Bellman 1958; Ford 1956) in \( \mathcal{O}(nm) \), where \( n \) is the number of nodes, and \( m \) the number of arcs. If an \( \mathcal{L} \) is present in \( \mathcal{G}_A \), then a surplus flow \( \Delta \) (from line 4) equal to the minimum value of the residual capacities on arcs in \( \mathcal{L} \) is pushed in the cycle in line 5. Ultimately, the algorithm is terminated if a cycle with negative cost does not exist.

### 3.3.2. Disparities between the MCF problem and the OWF collection system problem

The NCCA provides the global optimum for the classic MCF problem. However, in spite of its similarities to the OWF collection system, the following two major disparities preclude its application to this problem.

- The costs in set \( P_c \) are linear functions of the flow \( \Lambda \) in the MCF problem. For this problem, the cost is a non-convex step function as illustrated in Figure 2, where \( k \) is the number of WTs connected through an arc (equivalent to \( \lambda_a \)), and \( g(k) \) represents the cost function for all \( a \in \mathcal{A}_c \).
- Constraints [C1] and [C3] are not required in the MCF problem.

### 3.3.3. The NCCRH for the OWF collection system problem

Owing to the main differences between the OWF collection system problem and the MCF problem, modifications to the classic NCCA must be introduced. The goal is to propose a new algorithm (the NCCRH) to tackle the former problem, notwithstanding the impossibility of assuring either optimality or theoretically successful convergence, but with a good experimental performance on real-world problems.

In comparison with the sequence of works Gritzbach et al. (2018), Gritzbach et al. (2019) and Gritzbach, Wagner, and Wolf (2020), this article explores the capability of supporting constraints [C1] and [C3] by explicitly forbidding their violations, and by introducing a specific strategy to push flow through the network, equivalent to swapping connections. Likewise, a tailor-made algorithm to design a network satisfying [C1] to [C3] (presumably of good quality) is used as the starting point for the NCCCH algorithm.
The input directed graph is defined as \( G_c = (V_c, A_c, D_c) \), where \( V_c = N \), the arcs set \( A_c \) consists of the arcs in \( A \) (Section 2) excluding their inverses \( A^- \), and \( D_c \) the set of distances for each arc in \( A_c \):

\[
A_c = A^+ = \{ a : a \in A \land \tilde{a} \notin A^+ \} \\
D_c = \{ d_a : a \in A_c \} \\
A^- = \{ a : \tilde{a} \in A^+ \}. \tag{5}
\]

The residual graph in this case is not a function of the flow \( \Lambda \), but rather a constant network given by (6). Nodes set \( V_R \) includes the original set \( V_c \), a cluster transfer node \( i_o \) to model surplus flow interchange between OSSs, and the fictitious root node \( i_r \). Arcs set \( A_R \) is composed of arcs with tails at OSS nodes \( j \in S \) and heads at the cluster transfer node \( i_o \), the inverses of these arcs, plus sets \( A^+ \) and \( A^- \), and plus arcs connecting the fictitious root node \( i_r \) to the rest of nodes of the residual graph.

Comparing (6) with (4), it is noticeable that the residual capacities set \( U_\Lambda \) and costs set \( P_\Lambda \) have not been defined for this problem. As mentioned above, the network associated with \( G_R \) is constant for flow \( \Lambda \). The feasibility to push a surplus flow \( \Delta \) needs to be assessed with a residual cost function \( r(\lambda_a, \Delta) \), as initially proposed by Gritzbach et al. (2018), which accounts for the cable capacities intrinsically.

\[
G_R = (V_R, A_R) \\
V_R = V_c \cup \{ i_o \} \ (\text{cluster transfer node}) \cup \{ i_r \} \\
A_R = A^{0-} \cup A^{0+} \cup A^+ \cup A^- \cup A^r \tag{6} \\
A^{0-} = \{ a : a = (j, i_o) \land j \in S \} \\
A^{0+} = \{ a : a = (i_o, j) \land j \in S \} \\
A^r = \{ a : a = (i_r, j) \land j \in V_R \setminus \{ i_r \} \}. 
\]

In Section 3.3.1, \( \Lambda \) is defined as the flow set, where each element contains the non-negative flow \( \lambda_a \) for each arc \( a \in A^+ \). Within this context, \( \lambda_a \) may be equal to any real value, in such a way that \( \lambda_a \geq 0 \) if flow goes from node \( i \) to \( j \), where \( a = (i, j) \in A_c \), otherwise \( \lambda_a < 0 \) if flow goes from \( j \) to \( i \). Let \( \Lambda \) be redefined under this principle. Additionally, a mirroring flow set through each arc belonging to \( A^+ \) and \( A^- \) is formalized in (7), meaning that all arcs in \( A^+ \) with their corresponding complements in \( A^- \) are assigned the same flow.

\[
\Lambda_n = \Lambda \cup \{ \lambda_a \leftarrow \lambda_b : a = \bar{b}, b \in A^+, a \in A^- \}. \tag{7}
\]

The residual cost function \( r(\lambda_a, \Delta) \) is applied to each arc \( a \in A_R \), given a potential positive surplus flow \( \Delta \), and information about flow \( \lambda_a \) contained in \( \Lambda_n \). This function is defined in (8).

The residual cost function equals zero for all arcs in \( A^{0-} \) or in \( A^r \), and for those arcs in \( A^{0+} \) when the total incoming flow to the OSS is greater than or equal to \( \Delta \). The latter avoids outgoing flow from the OSS. For the arcs in \( A^+ \) for which the absolute value of the surplus flow plus the flow through them is lower than or equal to the capacity of the biggest cable available, the residual cost is equal to \( g^+ \) (9). Similarly, arcs in \( A^- \) for which the absolute value of the flow through them minus \( \Delta \) is lower than or equal to the capacity of the biggest cable available, their residual cost is given by \( g^- \) (10). Finally, the residual cost is set to infinity for infeasible arcs to avoid impractical situations, such as
Algorithm 3: Pseudocode of the negative cycle cancelling refining heuristic for the offshore wind collection system problem

1. Form input direct graph using (5), \( G_c = (V_c, A_c, D_c) \)
2. Form residual graph using (6), \( G_R = (V_R, A_R) \)
3. Find an initial feasible point based on \( T, \Lambda \)
4. Get mirroring flow set using (7), \( \Lambda_n \)
5. \( \Lambda_t \leftarrow \Lambda \)
6. \( \text{potflow} \leftarrow 0 \)
7. \( \Delta \leftarrow \text{unique}(|\Lambda|) \)
8. while True do
9. \( \Delta \leftarrow \Delta[\text{potflow}] \)
10. \( \mathcal{P}_\Delta \leftarrow \{r(\lambda_a, \Delta) : a \in A_R\} \)
11. Get list of cycles with more than two arcs based on \( \mathcal{P}_\Delta, \mathcal{L} \)
12. for \( \mathcal{L} \in \mathcal{L} \) do
13. if \( \mathcal{L} \) has negative cost then
14. if Satisfied \( [C1] \) and \( [C3] \) on \( \Lambda_t \) using \( G_c \) then
15. \( \Lambda \leftarrow \Lambda_t \)
16. Update \( \Lambda_n \) with (7)
17. \( \Delta \leftarrow \text{unique}(|\Lambda|) \)
18. \( \text{potflow} \leftarrow -1 \)
19. Break. Flow improved
20. end if
21. \( \Lambda_t \leftarrow \Lambda \)
22. end if
23. end for
24. \( \text{potflow}^+ = 1 \)
25. if \( \text{potflow} == \text{len}(\Delta) \) then
26. Break. Algorithm terminated
27. end if
28. end while
29. Return feasible flow, \( \Lambda \), and associated active cables in \( G_c, T \)

outgoing flow from an OSS or cable thermal capacity exceedance.

\[
\begin{align*}
\text{r}(\lambda_a, \Delta) &= \begin{cases} 
0, & (a \in \mathcal{A}^o^-) \lor (a \in \mathcal{A}^o^+ : a = (i,j), j \in S \land \Delta \leq f_\Lambda(j)) \lor (a \in \mathcal{A}^r) \\
g^+, & a \in \mathcal{A}^+ \land |\lambda_a + \Delta| \leq Q \\
g^-, & a \in \mathcal{A}^- \land |\lambda_a - \Delta| \leq Q \\
\infty, & (a \in \mathcal{A}^o^+ : a = (i,j), j \in S \land \Delta > f_\Lambda(j)) \lor (a \in \mathcal{A}^+ \land |\lambda_a + \Delta| > Q) \lor (a \in \mathcal{A}^- : a = (i,j), i \in T \land |\lambda_a - \Delta| > Q) \lor (a \in \mathcal{A}^- : a = (i,j), i \in S \land \Delta > \lambda_a).
\end{cases}
\end{align*}
\]

(8)

\[
\begin{align*}
g^+ &= g(|\lambda_a + \Delta|) - g(|\lambda_a|) \\
g^- &= g(|\lambda_a - \Delta|) - g(|\lambda_a|).
\end{align*}
\]

(9)

(10)

Algorithm 3 presents the working principles of the NCCRH. Lines 1 and 2 initialize the required inputs \( G_c \) (input directed graph) and \( G_R \) (residual graph), respectively. This is continued by the derivation of a feasible flow \( \Lambda \) by means of \( T \). In line 4, the mirroring flow set \( \Lambda_n \) is obtained, and then an auxiliary set \( \Lambda_t \) is defined. The last initialization step in line 7 is to get the set of potential surplus flows \( \Delta \), which consists of the unique positive arc flows in the feasible flow set \( \Lambda \).
Table 1. Input parameters for computational experiments.

| Instance | OWF       | $n_T$ | ($Q$)          | $W$ (M€ /km)      |
|----------|-----------|-------|----------------|-------------------|
| 1        | Horns Rev 1 | 80    | {7, 11, 13}    | {0.37, 0.39, 0.43}|
| 2        |           |       |                |                   |
| 3        |           |       |                |                   |
| 4        | Ormonde   | 30    | {5, 10}        | {0.41, 0.61}      |
| 5        |           |       |                |                   |
| 6        | DanTysk   | 80    | {4, 6, 8}      | {0.37, 0.39, 0.43}|
| 7        |           |       |                |                   |
| 8        | Thanet    | 100   | {7, 15}        | {0.38, 0.63}      |
| 9        |           |       |                |                   |
| 10       | Random O  | 74    | {7, 11, 13}    | {0.37, 0.39, 0.43}|
| 11       | Random I  | 74    | {4, 9}         | {0.38, 0.63}      |

The job of the process between lines 8 to 29 is to swap one active arc $a$ such as $\lambda_a \neq 0$ with an inactive one ($\lambda_a = 0$), thus preserving the satisﬁability of $[C_1]$, restricted to the no violation of $[C_3]$. Constraint $[C_2]$ is satisﬁed implicitly by the residual cost function (8).

Line 9 chooses one potential surplus ﬂow $\Delta$ at the time, which is then utilized in line 10 to create the set of residual costs $P_\Lambda$, such as $p_{ca} \in P_\Lambda$ is the residual cost of arc $a$ based on its ﬂow $\lambda_a \in \Lambda_n$. The methodology described in Gritzbach et al. (2018) to ﬁnd negative cycles $L$ with at least three arcs is implemented, as cycles with length equal to two bring in contradicting behaviour.

A cycle $L \in L$ is selected, and in case it has a total negative cost (otherwise try another cycle within the same potential surplus ﬂow or with the next in $\Delta$), then $\Delta$ units of surplus ﬂow are pushed on $L$ in the ﬂow set $\Lambda_1$ (line 14). Pushing surplus ﬂow means in this context that, if $a \in L \land a \in A^+$, then the ﬂow is pushed forwards through $a$, i.e. $\lambda_{\bar{a}} \leftarrow \lambda_a + \Delta$ or, on the contrary, if $a \in L \land \bar{a} \in A^+$, then the ﬂow is pushed backwards through $\bar{a}$, i.e. $\lambda_{a} \leftarrow \lambda_{\bar{a}} - \Delta$. This guarantees ﬂow conservation.

Constraints $[C_1]$ and $[C_3]$ are evaluated for the network stemming from arcs with ﬂow different from zero in $\Lambda_1$ according to $G_c$. If both constraints are satisﬁed, then $\Lambda$, $\Lambda_n$, $\Delta$ and $pot_{flow}$ are reset along with the whole previous process. Otherwise, the next cycle $L$ is studied. The algorithm is terminated in line 27, when all the potential surplus ﬂows have been exhausted. The output is a (possibly) improved ﬂow $\Lambda$ and associated improved edges matrix $T$.

4. Computational experiments

4.1. Methodology

This article, in addition to feasibility generation, explores the beneﬁt of warm starting. This strategy is particularly useful for mixed integer problems as the collection system for OWFs, since generally it helps an exact solver to activate internal heuristics, causing faster convergence.

The testbed of Table 1 is implemented with the aim of quantifying both functionalities in large- scale real-world cases, extracted from Pérez-Rúa et al. (2019) and Fischetti and Pisinger (2018), plus two extra synthetic OWFs with random patterns of WTs from Pérez-Rúa and Cutululis (2022). Data related to the real-world OWFs under study are available in ESCA, SEAFISH, and RenewableUK (2021).

The global optimization model to benchmark the solver-free heuristics with the workflow illustrated in Figure 1, and to assess the beneﬁts of warm starting an exact solver, is deployed in online supplemental data A, which can be accessed at https://doi.org/10.1080/0305215X.2022.2108027. The selected solver is the branch-and-cut implemented in IBM® ILOG® CPLEX® Optimization Studio V12.10 with default parameters (IBM 2021).

The experiments have been carried out on an Intel® Core™ i7-6600U CPU running at 2.80 GHz with four logical processors and 16 GB of RAM. The stopping criteria for all experiments using CPLEX is a relative gap less than or equal to 1%.
Table 2. Performance of the solver-free heuristics.

| Instance | Sol. qual. (M€) | Time (ms) | Diff. w.r.t. best^b (%) | Sol. qual. (M€) | Time (s) | Diff. w.r.t. best^b (%) | Sol. qual. (M€) | Time (s) (lt) (%) | NCCRH Gain with |
|----------|----------------|-----------|------------------------|----------------|---------|------------------------|----------------|------------------|----------------|
| 1 Inf-6cr. | 246.00 | 23.35 | 15.48 | 20.17 | 22.56 | 86.73 (5it) | 16.11 | −3.38 |
| 2 Inf-10cr. | 262.00 | 30.88 | 26.70 | 36.76 | 30.86 | 47.59 (2it) | 36.67 | −0.06 |
| 3 Inf-15cr. | 272.00 | 33.57 | 58.73 | 42.97 | 33.37 | 40.45 (1it) | 42.12 | −0.60 |
| 4 8.18 | 20.00 | 0.86 | 8.18 | 0.90 (0it) | 0.86 | |
| 5 8.52 | 29.00 | 2.04 | 8.50 | 1.75 (1it) | 1.80 | −0.23 |
| 6 Inf-9cr. | 176.00 | 45.09 | 8.75 | 16.45 | 45.09 | 12.96 (0it) | 16.45 | |
| 7 Inf-9cr. | 188.00 | 58.79 | 8.61 | 19.03 | 58.79 | 14.08 (0it) | 19.03 | |
| 8 24.19 | 186.00 | 8.87 | 23.24 | 93.76 (2it) | 4.59 | −3.93 |
| 9 Inf-1cr. | 189.00 | 26.40 | 1.60 | 1.42 | 26.40 | 32.18 (0it) | 1.42 | |
| 10 Inf-7cr. | 135.00 | 53.08 | 1.77 | 12.17 | 51.93 | 46.94 (2it) | 9.74 | −2.17 |
| 11 Inf-2cr. | 118.00 | 65.69 | 0.94 | 6.55 | 65.14 | 28.88 (2it) | 5.66 | −0.84 |

^b Best feasible point after CPLEX termination.

Table 3. Benefits of warm starting (WS) the MILP solver (CPLEX).

| Instance | Glob. opt. (no WS) | Glob. opt. (with WS) |
|----------|-------------------|---------------------|
|          | Obj. (M€) | Time (min) | Obj. improv. (%) | Time improv. w/o heur. (%) | Time improv. w. heur. (%) |
| 1        | 19.43 | 1.60 | 0.00 | −41.88 | 64.85 |
| 2        | 22.58 | 1.08 | 0.00 | −37.04 | 78.01 |
| 3        | 23.59 | 8.01 | −0.47 | −33.21 | −12.52 |
| 4        | 8.13 | 0.20 | −0.25 | −25.00 | −17.33 |
| 5        | 8.39 | 0.25 | −0.48 | −16.00 | −4.14 |
| 6        | 38.72 | 1.19 | 0.28 | 136.13 | 166.79 |
| 7        | 49.39 | 2.79 | 0.06 | −5.73 | 7.93 |
| 8        | 22.22 | 307.00 | 0.00 | −36.32 | −35.81 |
| 9        | 26.11 | 2.46 | −0.31 | −65.45 | −42.43 |
| 10       | 47.33 | 16.56 | −0.02 | −19.50 | −14.59 |
| 11       | 61.65 | 10.53 | 0.00 | −41.12 | −36.38 |

4.2. Performance analysis of the solver-free heuristics

The performance of the TSH, CCRH and NCCRH algorithms is available in Table 2. Only the problem instances 4, 5 and 8 lead to feasible points when applying the TSH algorithm. For the other instances, up to 15 cable crossings (instance 3, 15cr.) appear after the very rapid computing time of this heuristic (in the order of a few hundreds of milliseconds). For instances 4, 5 and 8, the feasible points have a solution value greater than the best known point (objective value after correspondingly solving the global optimization model with CPLEX in each case, see Table 3) by 0.86%, 2.04% and 8.87%, respectively. The solution quality depends on the correlation between the cost functions determined by length and investment.

The CCRH algorithm repairs all the infeasible points after the TSH. The longest computing time 58.73 s (for problem instance 3, where the greatest number of cable crossings is present) with an overall average of 15.32 s. A large variation is observed in the percentage deviation with respect to the best known point; extreme values are 42.97% (also for problem instance 3) and 1.42% (conversely the case with the lowest number of crossings, instance 9). The relation can be appreciated between the number of crossings after the TSH and the solution quality after the CCRH. The lowest deviations with respect to the best known point emerge in instances 9 (1.42%) and 11 (6.55%), where the number of crossings are one and two, respectively. A very strong correlation of 0.91 results between the number
The NCCRH manages to refine feasible points in 64% of the problem instances. The computing time is in the order of dozens of seconds, with a maximum of 93.76 s (instance 8) and an average of 49.44 s. The number of successful iterations, i.e. the number of times the flow is improved (line 20 in Algorithm 3) is also available (two iterations for problem instance 8). For the four problem instances where NCCRH does not improve the solution, the computing time spent trying to do so is a maximum of 32.18 s and an average of 15 s. The percentage improvement after running the NCCRH algorithm is presented in the last column of Table 2, with a maximum improvement of 3.93% and an average of 1.60% for the successful cases. In absolute terms, the improvement is the order of hundreds of thousands of euros in a timescale of seconds.

Graphical results of the designed collection systems for problem instance 10 are illustrated in Figure 3. The design after the TSH algorithm in Figure 3(a) poses seven cable crossings (observe for example, three crossings with the connection from node 1, OSS, to WT 59), while both constraints [C1] (note that there is only one electrical path from each WT towards an OSS) and [C2] (each connection meets the capacity set \( Q \)) are satisfied. The CCRH algorithm eliminates all crossings while still satisfying [C1] and [C2], see Figure 3(b). This comes at the expense of increasing the accumulated length of cables by 4.95%, implying as well an increase of total investment. The NCCRH succeeds in decreasing investment costs by 2.17%, through swapping connections, as for instance by eliminating the connection from 1 to 17 (in Figure 3(b)), and by creating a connection from 1 to 26, with subsequent upgrading of cable from 26 to 74 (in Figure 3(c)). The second change is the replacement of the connection from 1 to 19 by 1 to 71 using the same cable type.

Finally, Figure 4 compares, for instance 8, the speed (rate-of-change) of the incumbent curve without warm starting from CPLEX—calculated as \( \Delta \text{Cost}/\Delta \text{Time} \) of the dashed red line in Figure 5—and the speed of the NCCRH, computed as \((23.24-24.19)/93.76 \approx -0.61 \text{ M€/min.} \) CPLEX’s incumbent speed is higher during the first 5 min when the solution rapidly evolves from low (more than three times the final solution) to high quality values (around 7% greater than the final solution).
Nonetheless, after this period, for the time when CPLEX’s incumbent is still more than or equally expensive to the output of the NCCRH—between minutes 6 and 29 of Figure 4—it is noticeable that this heuristic is faster than CPLEX in the vicinities where both have similar input and output values. This result validates the purpose of the NCCRH, which is indeed able to refine high quality solutions in competitive computing times. Likewise, it shows the benefits of the sequential application of the TSH+NCCRH heuristics requiring only around 1.5 min to come up with a solution (23.24 M€) that CPLEX finds in 29 min.

4.3. Benefits of warm starting for the MILP model

Improvements of solution quality and computing time are summarized in Table 3. For the latter, the results are split into two columns, one for the time improvement without the overhead from the heuristics of Figure 1, and the other for when this is included. Without the heuristics time overhead, for the large majority of cases, warm starting helps to obtain at least equally good solutions in shorter times. A reduction in computing time between 5.73% to 65.45% is achieved for the testbed. However, an improvement for instance 6 does not result. This could be explained by the intrinsic diverse nature of combinatorial problems, where the performance of solvers can vary significantly.

Figure 4. Incumbent evolution speed for problem instance 8.

Figure 5. Incumbent and dual bound time evolution without and with warm starting of CPLEX for problem instance 8.
between instances. On the other hand, it is evident that for some instances whose solution time without implementing warm starting is rather short (a couple of minutes), the total time is longer when implementing such a strategy owing to the extra time incurred by the heuristics (instances 1, 2 and 7). Nevertheless, for the harder instances, a considerable pay-off is noticeable. One example is instance 8, where the time is reduced by almost 36%. In the best case, the reduction is roughly 42%, which validates the benefit of this approach.

Instance 8 requires 307 min to be solved with CPLEX without warm starting. The real-time evolution of the best known feasible point (incumbent) and the best known achievable solution (dual bound) is illustrated in Figure 5. Interestingly, warm starting seems to help the solver in such a way that the incumbent curve shifts towards the left, resulting in a faster convergence of 197 min. Conversely, the dual (lower) bound curve appears to be unaffected, which is an indication that supplementary methods should be proposed to speed up its behaviour.

5. Conclusions
This article demonstrates the ability to obtain feasible points empirically for large-scale real-world OWF collection systems, within a computing time in the order of dozen of seconds, and a solution quality with a spread of between roughly 1% and 42% to the best known feasible point. Two algorithms are presented, the CCRH, which is a greedy heuristic aiming to eliminate the largest number of crossings at once, and the NCCRH, inspired by previous work in which the classic minimum cost flow algorithm is modified, taking into consideration the particular properties of the OWF collection system problem. The CCRH algorithm behaves satisfactorily for all the studied problem instances, being able to repair all infeasible designs from the TSH. Likewise, the NCCRH improves solutions in 64% of the cases, with an enhancement up to almost 4%. Finally, the retrieved feasible points are fed into the branch-and-cut solver as a warm-starting solution, which demonstrates the benefit of accelerated convergence (up to 42% time reduction), and actually to come up with slightly better final solutions.

No theoretical guarantees can be provided for the proposed heuristics. This is largely due to the NP-hard nature of the OWF collection system problem. The empirical results, however, suggest promising contributions from the algorithms towards retrieving high-quality feasible points.

Future work can focus on: new versions of the CCRH to account for investment minimization while removing infeasibilities; strategies to detect longer negative cycles and swapping of several connections simultaneously; or inclusion of geographical constraints as forbidden areas and seabed topography.

Note
1. A cycle is defined, for example, as a sequence of arcs \( ((i,j), (j,u), (u,v), (v,i)) \), such that the head of an arc is equal to the tail of the next arc, and the initial node of the path is equal to the final one.

Acknowledgments
The author thanks Professors Mathias Stolpe and Nicolaos A. Cutululis for their inputs and feedback on the manuscript.

Disclosure statement
The author reports that there are no competing interests to declare.

Funding
This research has been supported by the Danmarks Frie Forskningsfond, DFF [grant no. 1127-00188B], under the project ‘Integrated Design of Offshore Wind Power Plants’, and by the internal CCA TOPFARM project.
Data availability statement

The data that support the findings of this study are available upon request.

References

Ahuja, Ravindra K., Thomas L. Magnanti, James B. Orlin, and K. Weihe. 1993. *Network Flows: Theory, Algorithms and Applications*. 1st ed. Lebanon, IN: Pearson Education.

Amaral, Luis, and Rui Castro. November 2017. “Offshore Wind Farm Layout Optimization Regarding Wake Effects and Electrical Losses.” *Engineering Applications of Artificial Intelligence* 60 (2016): 26–34.

Bauer, Joanna, and Jens Lysgaard. 2015. “The Offshore Wind Farm Array Cable Layout Problem: A Planar Open Vehicle Routing Problem.” *Journal of the Operational Research Society* 66 (3): 360–368.

Bellman, Richard. 1958. “On a Routing Problem.” *Quarterly of Applied Mathematics* 16 (1): 87–90.

Cerveira, Adelaide, Amaro F. de Sousa, E. J. Solteiro Pires, and Jose Baptista. 2016. “Optimal Cable Design of Wind Farms: The Infrastructure and Losses Cost Minimization Case.” *IEEE Transactions on Power Systems* 31 (6): 4319–4329.

Dutta, Sudipta, and Thomas J. Overbye. 2012. “Optimal Wind Farm Collector System Topology Design Considering Total Trenching Length.” *IEEE Transactions on Sustainable Energy* 3 (3): 339–348.

ESCA, SEAFISH, and RenewableUK. 2021. “KIS-ORCA.” https://kis-orca.org/.

Fagerfjäll, Patrik. 2010. “Optimizing Wind Farm Layout—More Bang for the Buck Using Mixed Integer Linear Programming.” MSc Thesis/Technical Report. Chalmers University of Technology.

Fischetti, Martina, and David Pisinger. 2018. “Optimizing Wind Farm Cable Routing Considering Power Losses.” *European Journal of Operational Research* 270 (3): 917–930.

Ford Jr, Lester R. 1956. *Network Flow Theory*. Technical report. Santa Monica, CA: Rand Corp.

González-Longatt, Francisco M., Peter Wall, Pawel Regulski, and Vladimir Terzija. 2012. “Optimal Electrical Network Design for a Large Offshore Wind Farm Based on a Modified Genetic Algorithm Approach.” *IEEE Systems Journal* 6 (1): 164–172.

Gritzbach, Sascha, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. 2018. “Towards Negative Cycle Canceling in Wind Farm Cable Layout Optimization.” *Energy Informatics* 1 (1): 183–193.

Gritzbach, Sascha, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. 2019. “Engineering Negative Cycle Canceling for Wind Farm Cabling.” In 27th Annual European Symposium on Algorithms (ESA 2019). Wadern, Germany: Leibniz International Proceedings in Informatics. doi:10.4230/LIPIcs.ESA.2019.55.

Gritzbach, Sascha, Dorothea Wagner, and Matthias Wolf. 2020. “Negative Cycle Canceling with Neighborhood Heuristics for the Wind Farm Cabling Problem.” In *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*, 299–307. New York: Association for Computing Machinery (ACM).

GWEC. 2020. *Global Offshore Wind Report 2020*. Technical report. Brussels, Belgium: GWEC. https://gwec.net/wp-content/uploads/2020/12/GWEC-Global-Offshore-Wind-Report-2020.pdf.

Hou, Peng, Weihao Hu, and Z. Chen. 2015. “Offshore Wind Farm Cable Connection Configuration Optimization Using Dynamic Minimum Spanning Tree Algorithm.” In *50th Power Engineering Conference (UPEC)*, 1–6. Piscataway, NJ: IEEE. doi:10.1109/UPEC.2015.7339896.

Hou, Peng, Weihao Hu, and Zhe Chen. 2016. “Optimisation for Offshore Wind Farm Cable Connection Layout Using Adaptive Particle Swarm Optimisation Minimum Spanning Tree Method.” *IET Renewable Power Generation* 10 (5): 694–702.

IBM. 2021. *IBM ILOG CPLEX Optimization Studio. CPLEX User Manual*. Technical report. Armonk, NY: IBM. https://www.ibm.com/docs/en/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcplex.pdf.

Jothi, Raja, and Balaji Raghavachari. 2005. “Approximation Algorithms for the Capacitated Minimum Spanning Tree Problem and Its Variants in Network Design.” *ACM Transactions on Algorithms (TALG)* 1 (5): 265–282.

Kershenbaum, A., and Wushow Chou. 1974. “A Unified Algorithm for Designing Multidrop Teleprocessing Networks.” *IEEE Transactions on Communications* 22 (11): 1762–1772.

Klein, Arne, and Dag Haugland. 2020. “Optimization of Reliable Cyclic Cable Layouts in Offshore Wind Farms.” *Engineering Optimization* 53 (2): 258–276. doi:10.1080/0305215X.2020.1717482

Lumbereras, Sara, Andres Ramos, and Santiago Cerisola. 2013. “A Progressive Contingency Incorporation Approach for Stochastic Optimization Problems.” *IEEE Transactions on Power Systems* 28 (2): 1452–1460.

ORE Catapult. 2020. “Wind Farm Costs.” Swindon, UK: BVG Associates. https://guidetoanoffshorewindfarm.com/wind-farm-costs.

Pérez-Rúa, Juan-Andrés, and Nicolaos A. Cutululis. 2019. “Electrical Cable Optimization in Offshore Wind Farms—A Review.” *IEEE Access* 7 (1): 85796–85811.

Pérez-Rúa, Juan-Andrés, and Nicolaos Antonio Cutululis. 2022. “A Framework for Simultaneous Design of Wind Turbines and Cable Layout in Offshore Wind.” *Wind Energy Science* 7 (2): 925–942.

Pérez-Rúa, Juan-Andrés, Sara Lumbereras, Andrés Ramos, and Nicolaos A. Cutululis. 2022. “Reliability-Based Topology Optimization for Offshore Wind Farm Collection System.” *Wind Energy* 25 (1): 52–70.
Pérez-Rúa, Juan-Andrés, Mathias Stolpe, Kaushik Das, and Nicolaos A. Cutululis. 2019. “Global Optimization of Offshore Wind Farm Collection Systems.” *IEEE Transactions on Power Systems* 35 (3): 2256–2267.

Pillai, A. C., J. Chick, L. Johannings, M. Khorasanchi, and V. De Laleu. 2015. “Offshore Wind Farm Electrical Cable Layout Optimization.” *Engineering Optimization* 47 (12): 1689–1708.

Sanchez Perez-Moreno, S., K. Dykes, K. O. Merz, and M. B. Zaaijer. 2018. “Multidisciplinary Design Analysis and Optimisation of a Reference Offshore Wind Plant.” *Journal of Physics: Conference Series* 1037 (4): Article ID 042004. doi:10.1088/1742-6596/1037/4/042004.

Wade, Bryce, Ricardo Pereira, and Cameron Wade. 2019. “Investigation of Offshore Wind Farm Layouts Regarding Wake Effects and Cable Topology.” *Journal of Physics: Conference Series* 1222 (1): Article ID 012007. doi:10.1088/1742-6596/1222/1/012007.