Engineering Negative Cycle Canceling for Wind Farm Cabling

Sascha Gritzbach
Karlsruhe Institute of Technology, Germany
sascha.gritzbach@kit.edu

Torsten Ueckerdt
Karlsruhe Institute of Technology, Germany
torsten.ueckerdt@kit.edu

Dorothea Wagner
Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Franziska Wegner
Karlsruhe Institute of Technology, Germany
franziska.wegner@kit.edu

Matthias Wolf
Karlsruhe Institute of Technology, Germany
matthias.wolf@kit.edu

Abstract

In a wind farm turbines convert wind energy into electrical energy. The generation of each turbine is transmitted, possibly via other turbines, to a substation that is connected to the power grid. On every possible interconnection there can be at most one of various different cable types. Each cable type comes with a cost per unit length and with a capacity. Designing a cost-minimal cable layout for a wind farm to feed all turbine production into the power grid is called the Wind Farm Cabling Problem (WCP).

We consider a formulation of WCP as a flow problem on a graph where the cost of a flow on an edge is modeled by a step function originating from the cable types. Recently, we presented a proof-of-concept for a negative cycle canceling-based algorithm for WCP [14]. We extend key steps of that heuristic and build a theoretical foundation that explains how this heuristic tackles the problems arising from the special structure of WCP.

A thorough experimental evaluation identifies the best setup of the algorithm and compares it to existing methods from the literature such as Mixed-integer Linear Programming (MILP) and Simulated Annealing (SA). The heuristic runs in a range of half a millisecond to under two minutes on instances with up to 500 turbines. It provides solutions of similar quality compared to both competitors with running times of one hour and one day. When comparing the solution quality after a running time of two seconds, our algorithm outperforms the MILP- and SA-approaches, which allows it to be applied in interactive wind farm planning.

2012 ACM Subject Classification Mathematics of computing → Network flows; Mathematics of computing → Graph algorithms; Mathematics of computing → Network optimization

Keywords and phrases Negative Cycle Canceling, Step Cost Function, Wind Farm Planning

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.55

Funding This work was funded (in part) by the Helmholtz Program Storage and Cross-linked Infrastructures, Topic 6 Superconductivity, Networks and System Integration, by the Helmholtz future topic Energy Systems Integration, and by the German Research Foundation (DFG) as part of the Research Training Group GRK 2153: Energy Status Data – Informatics Methods for its Collection, Analysis and Exploitation.

Acknowledgements We thank our colleagues Lukas Barth and Marcel Radermacher for valuable discussions and Lukas Barth for providing a code interface for simultaneous MILP-solver experiments.

1 Introduction

Wind energy becomes increasingly important to help reduce effects of climate change. As of 2017, 11.6 % of the total electricity demand in the European Union is covered by wind power [24]. Across the Atlantic, the state of New York aims at installing 2.4 GW of offshore wind energy capacity by 2030, which could cover the demand of 1.2 million homes [19].

In an offshore wind farm a set of turbines generate electrical energy. From offshore substations the energy is transmitted via sea cables to an onshore grid point. One of the biggest wind farms currently planned is Hornsea Project Three in the North Sea with up to 300 turbines and twelve substations [1]. To transport turbine production to the substations, a system of cables links turbines to substations (*internal cabling*) where multiple turbines may be connected in series. The designer of a wind farm has various cable types available, each of which with respective costs and thermal capacities. The latter restricts the amount of energy that can be transmitted through a cable. Planning a wind farm as a whole consists of various steps, including determining the locations for turbines and substations, layouting the connections from substations to the grid point, and designing the internal cabling. The planning process comes with a high level of complexity, which automated approaches struggle with [23]. Therefore, one might opt for decoupling the planning steps. We call the task of finding a cost-minimal internal cabling of a wind farm with given turbine and substation positions, as well as given turbine production and substation capacities, the Wind Farm Cabling Problem (WCP). Since WCP is a generalization of the $\mathcal{NP}$-hard problem Capacitated Minimum Spanning Tree [21], it is $\mathcal{NP}$-hard as well.

Due to the overall cost of a wind farm, using one day of computation time or more arguably is a reasonable way to approach WCP. Such computation times, however, are not appropriate for an interactive planning process: Imagine a wind farm planner uses a planning tool which allows altering turbine positions to explore their influence on possible cable layouts. In that case, computation times of at most several seconds are desirable.

1.1 Contribution and Outline

We extend our recent proof-of-concept, in which negative cycle canceling is applied to a formulation of WCP as a network flow problem (cf. Section 3) with a step cost function representing the cable types [14]. The idea of negative cycle canceling is to iteratively identify cycles in a graph in which the edges are associated with the costs of (or gains from) changing the flow. Normally, a cycle of negative total cost corresponds to a way to decrease the cost of a previously found flow. Due to the step cost function, however, not every negative cycle helps improve a solution to WCP. We explore this and other issues for negative cycle canceling that arise from the step cost function in the flow problem formulation for WCP. We present a modification of the Bellman-Ford algorithm [3, 9] and build a theoretical foundation that explains how the modified algorithm addresses the aforementioned issues, e. g., by being able to identify cycles that actually improve a solution. This modification works on a subgraph of the line graph (cf. page 6) of the input graph and can be implemented in the same asymptotic running time as the original Bellman-Ford algorithm.

We further extend that heuristic by identifying two key abstraction layers and applying different strategies in those layers. Using different initializations is hinted at in the section on future work in [14]. We follow this hint and design eight concrete initialization strategies. In another layer, we propose a total of eight so-called “delta strategies” that specify the order in which different values for flow changes are considered.

In [14] we compared the Negative Cycle Canceling (NCC) algorithm to a Mixed-integer Linear Program (MILP) using the MILP solver Gurobi with one-hour running times on benchmark sets from the literature [17]. We extend this evaluation by identifying the best of our variants and by comparing its results to the results of MILP experiments after running times of two seconds, one hour, and one day on the same benchmark sets. A running time of two seconds helps identify the usefulness of the NCC algorithm to an interactive planning process. The other running times stand for non-time-critical planning. We also compare the algorithm to an approach using Simulated Annealing [17] with different running times. The results show that our heuristic is very fast since it terminates on instances with up to 500 turbines in under two minutes. At two seconds our algorithm outperforms its competitors, making it feasible for interactive wind farm planning. Even with longer running times for the MILP- and SA-approaches, our algorithm yields solutions to WCP of similar quality but in tens of seconds.

In Section 2 we review existing work on WCP and negative cycle canceling. In Section 3 we define WCP as a flow problem. We give theoretical insights on the difference to standard flow problems and present and analyze our Negative Cycle Canceling algorithm in Section 4. An extensive experimental evaluation of the algorithm is given in Section 5. We conclude with a short summary of the results and outline possible research directions (see Section 6).

2 Related Work

In one of the first works on WCP, a hierarchical decomposition of the problem was introduced [4]. The layers relate to well-known graph problems and heuristics for various settings are proposed. Since then, considerable effort has been put into solving different variants of WCP. Exact solutions can be computed using Mixed-integer Linear Program (MILP) formulations including various degrees of technical constraints, e. g., line losses, component failures, and wind stochasticity [18]. However, sizes of wind farms that are solved to optimality in reasonable time are small. Metaheuristics such as Genetic Algorithms [25, 6] or Simulated Annealing [17] can provide good but not necessarily optimal solutions in relatively short computation times.

We applied negative cycle canceling to a suitable flow formulation for WCP [14], but there is still an extensive agenda of open questions such as investigating the effect of other cable types, a comparison to existing heuristics, and using the solution as warm start for a MILP solver. Originally, negative cycle canceling is proposed in the context of minimum cost circulations when linear cost functions are considered [16]. The algorithm for the Minimum-Cost Flow Problem based on cycle canceling with strongly polynomial running time runs in $\mathcal{O}(nm(\log n)\min\{\log(nC), m\log n\})$ time on a network with n vertices, m edges, and maximum absolute value of costs C [12]. The bound for the running time of this algorithm was later tightened to $\Theta(\min\{nm\log(nC), nm^2\})$ [22]. Negative cycle canceling has also been used for problems with non-linear cost functions. Among these are multicommodity flow problems with certain non-linear yet convex cost functions based on a queueing model [20] and the Capacity Expansion Problem for multicommodity flow networks with certain non-convex and non-smooth cost functions [7]. A classic algorithm for finding negative cycles is the Bellman-Ford algorithm [3, 9] with heuristic improvements [13, 11]. An experimental evaluation of these heuristics and other negative cycle detection algorithms is given in [5].

A step cost function similar to the one in WCP appears in a multicommodity flow problem, for which exact solutions can be obtained by a procedure based on Benders Decomposition [10]. However, this procedure is only evaluated on instances with up to 20 vertices and 37 edges and

some running times exceed 13 hours. While our approach does not guarantee to solve WCP to optimality, our evaluation shows that the solution quality is very good compared to the MILP with running times not exceeding two minutes on wind farms with up to 500 turbines.

3 Model

The model presented in this paper is based on an existing flow model for WCP [14]. We briefly recall the model. Given a wind farm, let V_T and V_S be the sets of turbines and substations, respectively. We define a vertex set V of a graph by $V = V_T \cup V_S$. For any two vertices u and v that can be connected by a cable in the wind farm, we define exactly one directed edge $e = (u, v)$, where the direction is chosen arbitrarily. We obtain a directed graph $G = (V, E)$ with $V = V_T \cup V_S$ and $E \subseteq (V \times V) \setminus (V_S \times V_S)$ such that $(u, v) \in E$ implies $(v, u) \notin E$. There are no edges between any two substations since we consider the wind farm planning step in which all positions of turbines and substations, as well as the cabling from substations to the onshore grid point have been fixed. We assume that all turbines generate one unit of electricity. Note that our algorithm can be easily generalized to handle non-uniform integral generation. Substations have a capacity $\mathrm{cap}_{\mathrm{sub}} \colon V_S \to \mathbb{N}$ representing the maximum amount of turbine production they can handle and each edge has a length given by $\mathrm{len} \colon E \to \mathbb{R}_{\geq 0}$ representing the geographic distance between the endpoints of the edge.

A *flow* on G is a function $f \colon E \to \mathbb{R}$ and for an edge (u, v) with $f(u, v) > 0$ (resp. < 0), we say that $f(u, v)$ units of flow go from u to v (resp. $-f(u, v)$ units go from v to u). For a flow f and a vertex u we define the *net flow in* u by $f_{\mathrm{net}}(u) = \sum_{(w,u)\in E} f(w, u) - \sum_{(u,w)\in E} f(u, w)$. A flow f is *feasible* if the conditions on flow conservation for both turbines (Equation (1)) and substations (Equation (2)) are satisfied and if there is no outflow from any substation (Equations (3) and (4)).

$$f_{\mathrm{net}}(u) = -1 \qquad \forall u \in V_T, \tag{1}$$

$$f_{\mathrm{net}}(v) \leq \mathrm{cap}_{\mathrm{sub}}(v) \qquad \forall v \in V_S, \tag{2}$$

$$f(u, v) \geq 0 \qquad \forall (u, v) \in E : v \in V_S, \tag{3}$$

$$f(v, u) \leq 0 \qquad \forall (v, u) \in E : v \in V_S. \tag{4}$$

Let $c \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be a non-decreasing, left-continuous step function with $c(0) = 0$. This function represents the cable costs and $\sup\{x \in \mathbb{R}_{\geq 0} : c(x) < \infty\}$ is the maximum cable capacity, which we assume to be a natural number. Note that such a function is neither convex nor concave in general. The cost of a flow on a wind farm graph is then given by

$$\mathrm{cost}(f) = \sum_{e \in E} c\left(|f(e)|\right) \cdot \mathrm{len}(e). \tag{5}$$

The value of $c\left(|f(e)|\right)$ stands for the cost per unit length of the cheapest cable type with sufficient capacity to transmit $|f(e)|$ units of turbine production. With all that, WCP is the problem of finding a feasible flow f on a given wind farm graph that minimizes the cost. There is an analogon to the linear-cost integer flow theorem (e. g. [2, Thm. 9.10]) that guarantees an optimal flow with integral values.

▶ **Lemma 1.** *Suppose the cost function is discontinuous only at integers and there is a feasible flow. Then, there is a cost-minimal integral flow.*

4 Algorithm

Given a wind farm graph G we define the residual graph R of G with vertices $V(R)$ and edges $E(R)$ by $V(R) = V(G) \cup \{s\}$ and $E(R) = \{e, \bar{e} : e \in E(G)\} \cup \{(v, s), (s, v) : v \in V_S\}$ where $\bar{e}$ is the reverse of e. The new vertex s, the *super substation*, is a virtual substation without capacity, that is connected to all substations. The edges to and from s are used to model the substation capacity constraints and to allow the production of one turbine to be reassigned to another substation.

For a given feasible flow f in G of finite cost and $\Delta \in \mathbb{N}$ we further define *residual costs*, which represent by how much the cost for the edge changes if the flow on the edge is increased by Δ (cf. Figure 1 (a) – (d) for an example). Note that for negative quantities of flow this implies that the absolute value of the flow may be reduced or even the direction of the flow on an edge may change. More formally, we define $\gamma\colon E(R) \to \mathbb{R}$ by $\gamma(e) = \big(c(|f(e) + \Delta|) - c(|f(e)|)\big) \cdot \text{len}(e)$ for all $e \in E(R)$ that are neither incident to s nor lead to a substation where we alias $f(\bar{e}) = -f(e)$ for all $e \in E(G)$. By this definition the residual costs are infinite if $c(|f(e) + \Delta|) = \infty$, i. e., if the maximal capacity on e is exceeded. For $u \in V_S$ and $v \in V_T$, we set $\gamma(u, v) = \infty$ whenever $f(v, u) < \Delta$ because sending $f(u, v) + \Delta$ units from u to v would otherwise imply that flow leaves a substation. On edges into s, we set $\gamma(u, s) = 0$ if and only if $f(u, s) + \Delta \leq \text{cap}_{\text{sub}}(u)$ and $\gamma(u, s) = \infty$ otherwise. On edges leaving the super substation, we set $\gamma(s, u) = 0$ if and only if $f(u, s) \geq \Delta$ and $\gamma(s, u) = \infty$ otherwise to prevent flow from leaving the substation.

In a nutshell, the Negative Cycle Canceling (NCC) algorithm (Algorithm 1) starts with an initial feasible flow and some value of Δ, computes the residual costs, and looks for a negative cycle[1]. If the algorithm finds a negative cycle, it cancels the cycle, i. e., it changes the flow by adding Δ units of flow on all (residual) edges of the cycle. Note that this may decrease the actual amount of flow on edges of G. Then this procedure is repeated with the new flow and a (possibly new) value of Δ. If no negative cycle is found, a new value of Δ is chosen and new residual costs are computed. This loop is repeated until all sensible values of Δ have been considered for a single flow, which is then returned by the algorithm. This flow is of integer-value, since the initial flow is designed to only have integer values and we solely consider natural values for Δ. Without loss of generality we can restrict ourselves to integer flows according to Lemma 1, even though our algorithm does not necessarily find an optimal solution of WCP. One question we answer is to what extent the algorithm benefits from different initial flows and different orders in which the values of Δ are chosen. We present different strategies in Sections 4.3 and 4.4.

The details of the algorithm (cf. Sections 4.1 and 4.2) address problems that arise from the special structure of WCP, namely the non-linear cost function c. Firstly, in classical min-cost flow problems, when c is linear, the cost for changing flow by a certain amount is proportional to the amount of flow change (and the length of the respective edge) and does not depend on the current amount of flow on that edge. Hence, there is no need for computing residual costs for different values of Δ. Secondly, *short* cycles, i. e., cycles of two edges, may have non-zero total cost in WCP (cf. cycle uv_2u in Figure 1 (d)). Canceling such a cycle, however, does not change the flow and therefore does not improve the solution. Hence, only cycles of at least three edges (*long* cycles) are interesting to us because they do not contain both an edge

[1] A *cycle* is a sequence of consecutive edges such that the first edge starts at the same vertex where the last edge ends and such that no two edges start at the same vertex. That is, all cycles are simple. A cycle is said to be *negative* if the sum of residual costs over all edges is negative.

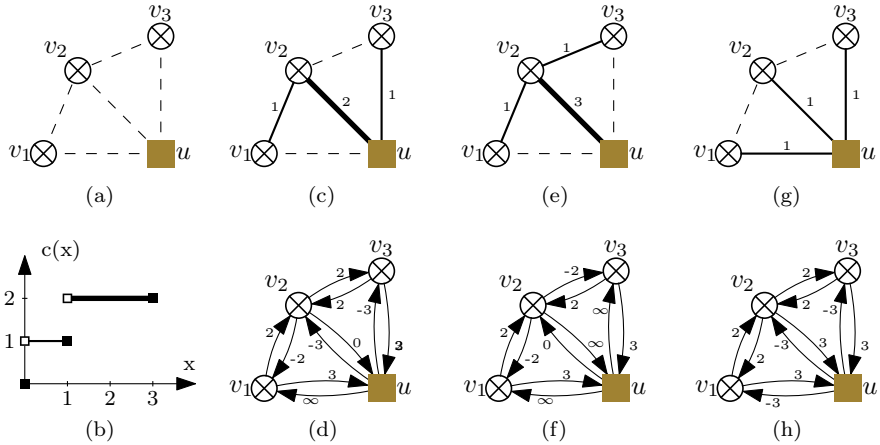

Figure 1 Examples of flows and corresponding residual graphs. (a) shows a wind farm graph. Edges between turbines are of length 2, edges between the substation u and any turbine are of length 3. (b) depicts a cost function induced by two cable types. (c) displays a feasible flow. Dashed lines do not carry any flow. The thickness of solid lines represents the necessary cable type to carry the respective flow. (d) is the residual graph for the flow in (c) and $\Delta = 1$. The super substation is omitted for ease of presentation. There are three negative cycles: uv_2u, uv_2v_1u, and uv_3v_2u. (e) shows the flow obtained by sending one unit of flow along uv_3v_2u in (c). (f) is the residual graph for (e) and $\Delta = 1$. (g) depicts the flow obtained by sending one unit of flow along uv_2v_1u in (c). (h) displays the residual graph for (g) and $\Delta = 1$.

and its reverse. Finding any negative cycle can be done in polynomial time but finding long negative cycles is $\mathcal{NP}$-hard for general directed graphs [15, Theorem 4 for $k = 3$]. Thirdly, the order of canceling cycles matters (Figure 1 (c) – (g)). In (d), there are two long negative cycles: uv_2v_1u and uv_3v_2u. After canceling uv_3v_2u (Figure 1 (e)), the other cycle uv_2v_1u is not negative anymore. Ultimately, Figure 1 (e) and (f) show that the non-existence of negative cycles in (all) residual graphs does not imply that the underlying flow is optimal – contrary to min-cost flow problems with linear cost functions. In other words, there are flows that represent local but not global minima.

4.1 Detecting Long Negative Cycles

We assume that the reader is familiar with the standard Bellman-Ford algorithm [3, 9], which is a common approach to finding negative cycles. We observed in preliminary experiments that it mostly reports short cycles even if long cycles exist. The reason is that negative residual costs on an edge are repeatedly used if the cost of the reverse edge is, say, zero. In that case, the negative residual cost strongly influences the distance labels on close vertices and overshadows long cycles (see cycle uv_2u in comparison to cycle uv_2v_1u in Figure 1 (d)).

One solution is to prohibit propagating the residual cost of an edge over its reverse edge. To this end, we employ the Bellman-Ford algorithm on the subgraph L of the directed line graph[2] of R which we obtain from the line graph by removing all edges representing U-turns, i. e., edges of the form $(e, \bar{e})$ for $e \in E(R)$. We define the cost of an edge (e_1, e_2) in L as $\gamma(e_2)$. At every vertex e of L we maintain a distance label $\ell(e)$ initialized as $\gamma(e)$. Thus, throughout

[2] The *line graph* $L(G)$ of a directed graph G shows which edges are incident to each other. It is defined by $V(L(G)) = E(G)$ and $E(L(G)) = \{((u, v), (v, w)) \colon (u, v), (v, w) \in E(G)\}$.

the Bellman-Ford algorithm, $\ell(e)$ represents the length of some walk[3] in L starting at any vertex of L and ending at e. By construction of L, the label $\ell(e)$ also stands for some walk in R which ends at the target vertex of e and which does not traverse an edge of R directly after its reverse. Consequently, a cycle C in L corresponds to a closed walk W without U-turns of the same cost in R. In particular, W is not a short cycle, which is what we wanted. It may still occur, however, that W includes an edge and its reverse. In that case, W consists of more cycles that may be negative themselves. Therefore, we decompose the closed walk W into cycles, which, in turn, can be canceled one after another. For more details, refer to Section 4.2.

A downside of running the Bellman-Ford algorithm on the line graph is that more labels have to be stored and the running time of the algorithm is in $\mathcal{O}(|V(L)| \cdot |E(L)|)$, which is worse than the running time on R. We present how to implement an algorithm that directly works on R, that is equivalent to the Bellman-Ford algorithm on L, and that has the asymptotic running time as the original Bellman-Ford algorithm on R. When running the Bellman-Ford algorithm on L, there is one label for every vertex of L. Each of those labels gives rise to a label on an edge of R. The labels at incoming edges of $v \in V(R)$ are used to compute the labels at outgoing edges of v. Let (v, w) and (v, x) be two edges leaving v. Let us assume that (x, v) has the smallest label of all edges entering v. Then, (x, v) is used to relax (v, w). But it cannot be used to relax (v, x). To do so, we need the second smallest label of all edges entering v. This yields the following observation.

▶ **Observation 2.** *For each vertex v of R only the two smallest labels of incoming edges of v are required to correctly update the labels on outgoing edges of v.*

Consequently, throughout our modified version of the Bellman-Ford algorithm, we maintain two distance labels $\ell_1(v)$ and $\ell_2(v)$, and two parent pointers $\text{parent}_1(v)$ and $\text{parent}_2(v)$ for every $v \in V(R)$, respectively. As above, $\ell_i(v)$ with $i = 1, 2$ stands for the length of a U-turn-free walk whose first edge is arbitrary and whose last edge is $(\text{parent}_i(v), v)$. That means that the parent pointers hold the edges that have been used to build the values of the distance labels. The algorithm ensures that $\text{parent}_1(v) \neq \text{parent}_2(v)$ and $\ell_1(v) \leq \ell_2(v)$ for every $v \in V(R)$. If, during a relaxation step, several incumbent labels and a newly computed candidate label have the same value, we break ties in favor of the older labels – as in the original algorithm. With Observation 2 we show reduced bounds for the number of iterations and the overall running time compared to a straightforward implementation on L.

▶ **Theorem 3.** *If after $2 \cdot |V(R)|$ iterations there is an edge whose label can still be reduced, then there is a negative cycle in L.*

▶ **Corollary 4.** *A negative cycle in L can be computed in $\mathcal{O}(|V(R)| \cdot |E(R)|)$ time if one exists.*

4.2 Algorithm in Detail

The previously described Bellman-Ford algorithm on L is encapsulated in Algorithm 1. We first compute some initial flow (line 1) using one of eight initialization strategies presented in Section 4.3. In line 3 we compute the residual graph R using a given flow f and a given Δ and run the modified Bellman-Ford algorithm (line 4). In the repeat-loop, we consider one edge after another and check in line 7 if it can be relaxed (again) after the Bellman-Ford algorithm. In that case, we extract a walk W in R with negative costs leading to that edge

[3] A *walk* is a sequence of consecutive edges. A walk is called *closed* if the start vertex of the first edge equals the target vertex of the last edge. In particular, every cycle is a closed walk.

by traversing parent pointers. However, canceling W directly may not always improve the costs of the flow as W may still contain an edge and its reverse. We decompose W into a set of simple cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ in line 8 and cancel each cycle independently if it is long and has negative costs (lines 9 to 12). Note that even though W has negative costs, it may happen that only short cycles in $\mathcal{C}$ have negative costs and all long cycles have non-negative costs. In this case we search for another negative cycle in L (line 13).

If no negative cycle in the current graph L is canceled, a new value for Δ is determined according to the *delta strategy* (cf. Section 4.4) in line 14 and new residual costs γ are computed. Line 14 also checks if every possible value for Δ has been used after the last update of f without improving the solution. If so, f is returned.

We apply two well-known speed-up techniques to the Bellman-Ford algorithm. Firstly, if one iteration does not yield any update of any label, then the computation is aborted and no negative cycle can be found in the current residual graph. Secondly, after sorting edges by start vertices, we track whether the labels at a vertex v have been updated since last considering its outgoing edges. If not, then there is no need to relax the outgoing edges.

Algorithm 1 Negative Cycle Canceling.

```
   Input: Graph G, costs c, edge lengths len
   Result: A feasible flow f in G
 1 f := InitializeFlow(G, len), Δ := InitialDelta
 2 while Δ ≠ NULL do
 3     (R, γ) := ComputeResidualGraph(G, c, f, Δ)
 4     RunBellmanFord(R, γ)
 5     found := false
 6     foreach e ∈ E(R) do
 7         W := FindNegativeClosedWalk(R, e)
 8         C := DecomposeWalkIntoCycles(W)
 9         foreach C ∈ C do
10             if |C| ≥ 3 and γ(C) < 0 then
11                 f := AddFlowOnCycle(f, C, Δ)
12                 found := true
13         if found then break
14     Δ := NextDelta(Δ, found)
15 return f
```

4.3 Initialization Strategies

Before we can start searching for and canceling negative cycles, we need some feasible initial flow. To obtain such a flow, we consider eight strategies, which all roughly work as follows. We pick a turbine u whose production has not been routed to a substation yet. We then search for a shortest path P from u to a substation v with free capacity using Dijkstra's algorithm [8]. The search only considers edges on which the production of the turbine can be routed, i. e., it ignores congested edges. We then route the production of u along P to v.

We consider two metrics to compute shortest paths. Either we use the lengths defined by len (cf. Section 3) or we assume a length of 1 for every edge. Turbine production can either be routed to a nearest or a farthest (in the sense of the respective metric) substation with

free capacity. There are two ways in which the flow is updated: The simpler variant routes only the production of u along P, i. e., the flow along P is increased by 1. The other variant greedily collects as much production from u and other turbines on P as possible without violating any capacity constraints. The resulting flows are integral since the substation capacities and the maximum cable capacity are natural numbers. If no feasible flow of finite cost is found during the initialization, the algorithm returns without a result.

This yields eight initialization strategies, which we name as follows. The base part of each name is either `BFS` if unit distances are used or `Dijkstra` (abbr. `Dijk`) if the distances given by len are used. This part is followed by a suffix specifying the target substation: `Any` (abbr. `A`) for the nearest and `Last` (abbr. `L`) for the farthest substation. An optional prefix of `Collecting` (abbr. `C`) means that the production is greedily collected along shortest paths. For example, `CollectingDijkstraLast` (abbr. `C-Dijk-L`) iterates over all turbines and for each turbine u it finds the substation v such that the shortest path given by len from u to v is longest among all substations. Along a shortest path from u to v, turbine production is collected greedily.

4.4 Delta Strategies

A delta strategy consists of two parts: an initial value for Δ and a function that returns the value of Δ for the following iteration. We discuss eight delta strategies. The simplest one starts with $\Delta = 1$ and increments Δ until a negative cycle is canceled. Then, Δ is reset to 1. We call this strategy `Inc` (as in increasing). Similarly, `Dec` (as in decreasing) starts with the largest possible value for Δ, which is twice the largest cable capacity. Then, Δ is decremented until a cycle is canceled and reset to the largest value. The third strategy `IncDec` behaves like `Inc` until a negative cycle is canceled. Then, it decrements Δ until $\Delta = 1$ and behaves like `Inc` again. To improve performance, all Δ can be skipped during incrementation up to the last value of Δ for which a negative cycle was canceled. The fourth strategy `Random` returns random natural numbers between one and the maximum possible value for Δ. Between any two cycle cancellations, no value is repeated.

For each strategy, we consider the following modification: After canceling a negative cycle, we retain the current value of Δ, recompute the residual costs with the new flow, and run the Bellman-Ford algorithm again. We repeat this, until Δ does not yield a negative cycle. In that case, Δ is changed according to the respective delta strategy. We call the strategies after the modification `StayInc`, `StayDec`, `StayIncDec`, and `StayRandom` (or `S-Inc`, `S-Dec`, `S-IncDec`, and `S-Random` for short).

5 Experimental Evaluation

In the previous sections, we introduce a heuristic with various strategies for the WCP. We first use statistical tests to evaluate these strategies and identify the best ones. Using the result we compare the best *variant* (i. e., best combination of initialization and delta strategy) with different base line algorithms for the WCP namely solving an exact MILP formulation and a Simulated Annealing algorithm [17]. In preliminary experiments a comparison of the MILP solvers Gurobi and CPLEX shows that Gurobi tends to produce better solutions. We therefore use Gurobi to solve the MILP formulation. However, getting an optimal solution takes too long in almost all instances. Thus, we restrict Gurobi to different maximum running times.

For our evaluation we use benchmark sets for wind farms from the literature [17] consisting of wind farms of different sizes and characteristics: small wind farms with exactly one substation ($\mathcal{N}_1$: 10–79 turbines), wind farms with multiple substations ($\mathcal{N}_2$: 20–79 turbines,

Table 1 Comparison of delta strategies over all initialization strategies. An entry in row i and column j shows on how many instances strategy i produces better solutions than strategy j. Values are marked by a star if they are significant with $p < 10^{-2}$ and by two stars if $p < 10^{-4}$. The best strategy is marked in green.

	`Inc`	`Dec`	`IncDec`	`Random`	`S-Inc`	`S-Dec`	`S-IncDec`	`S-Random`
`Inc`	—	64.1 %**	46.2 %	58.1 %**	53.5 %	64.0 %**	52.0 %	56.9 %*
`Dec`	35.9 %	—	34.3 %	43.1 %	37.1 %	49.2 %	35.1 %	39.6 %
`IncDec`	53.8 %	65.7 %**	—	59.1 %**	55.2 %	64.3 %**	52.7 %	57.9 %**
`Random`	41.9 %	56.9 %*	40.9 %	—	46.9 %	56.3 %*	44.1 %	44.8 %
`S-Inc`	46.5 %	62.9 %**	44.8 %	53.1 %	—	59.1 %**	46.4 %	52.9 %
`S-Dec`	36.0 %	50.8 %	35.7 %	43.7 %	40.9 %	—	38.6 %	42.1 %
`S-IncDec`	48.0 %	64.9 %**	47.3 %	55.9 %	53.6 %	61.4 %**	—	55.4 %
`S-Random`	43.1 %	60.4 %**	42.1 %	55.2 %	47.1 %	57.9 %**	44.6 %	—

$\mathcal{N}_3$: 80–180 turbines, $\mathcal{N}_4$: 200–499 turbines), and complete graphs ($\mathcal{N}_5$: 80–180 turbines). Our code is written in C++14 and compiled with GCC 7.3.1 using the `-O3 -march=native` flags. All simulations are run on a 64-bit architecture with four 12-core CPUs of AMD clocked at 2.1 GHz with 256 GB RAM running OpenSUSE Leap 15.0. We use Gurobi 8.0.0 and all computations are run in single-threaded mode to ensure comparability of the different algorithms.

5.1 Comparing Variants of our Algorithm

In a first step, we want to determine which delta strategy works best. To this end, we randomly select 200 instances per benchmark set. We run our algorithm on each instance with every pair of delta and initialization strategy. We first observe that all variants are fast, with running times between tenths of milliseconds to 4.5 minutes on large instances in the worst case. Since all variants have similarly small running times, we base our decision which variant to choose solely on their solution qualities.

To compare the variants in terms of solution quality, we compute for each delta strategy i and instance m the mean $X_m^{(i)}$ of the solution values over all eight initialization strategies. This gives us 1000 data points per delta strategy. For delta strategies i, j we perform a Binomial Sign Test counting instances with $X_m^{(i)} < X_m^{(j)}$ and $X_m^{(j)} < X_m^{(i)}$, that means for this test we are rather interested in whether strategy i performs better than strategy j on instance m and not by how much i is better than j on m. Table 1 summarizes the results of all tests after Bonferroni-correction by 112 (the number of tests from both delta and initialization strategies). The percentage given in an entry in row i and column j states on how many instances i performes strictly better than j after averaging over all initialization strategies. Note that entries (i, j) and (j, i) need not represent 1000 instances, as two variants may return equal solution values.

In the row `IncDec`, all values are above 50 %, four of which are significant at the 10^{-4}-level. The smallest value (52.7 % in column `StayIncDec`) stands for 481 instances on which `IncDec` performs better than `StayIncDec`. To the contrary, there are 431 instances on which `StayIncDec` yields better solutions (cf. entry 47.3 % in row `StayIncDec` and column `IncDec`). While the differences between the four delta strategies involving `Inc` and `IncDec` are not statistically significant, `IncDec` does seem to have a slight advantage over the others. Hence we consider `IncDec` as the best delta strategy.

In Figure 2 (left), for the dark green curve all instances are ordered by $X_m^{(\texttt{Random})}/X_m^{(\texttt{IncDec})}$ in ascending order. For a given value α on the abscissa, the curve shows the relative cost factor of the instance at the α-quantile in the computed order. The other curves work

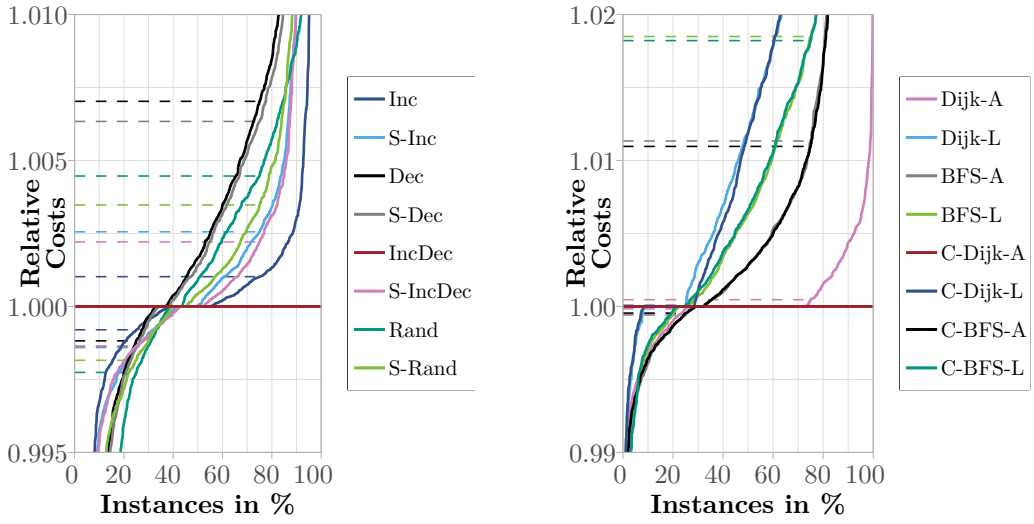

Figure 2 Evaluation of the NCC Algorithm using different strategies. For each strategy and for each instance, the ratio of the best solution value found by that NCC variant to the best solution value found by the reference variant (marked in red) are computed. They are shown in increasing order. The dashed lines represent the 25% and 75% quantiles of the instances. *Left:* The delta strategies are presented relative to the `IncDec` strategy. Solution values represent the average over all initialization strategies. *Right:* The initialization strategies are presented relative to the `CollectingDijkstraAny` strategy with fixed delta strategy `IncDec`.

accordingly. We see, for example, that `IncDec` works strictly better than `StayInc` on 50.9 % of all instances and on 7.3 % of all instances `IncDec` outperforms `Inc` by at least 0.5 % in cost ratio. The minimum ratios range between 0.868 (`Random`) and 0.918 (`StayIncDec`) and the maximum ratios are between 1.069 (`StayDec`) and 1.126 (`StayIncDec`).

Next, we want to find an initialization strategy after fixing `IncDec` as the delta strategy. We pair each initialization strategy with `IncDec` on the same 1000 instances. Table 2 summarizes the results of all pairwise tests after Bonferroni-correction with factor 112. We see that initialization strategies that route turbine production to the nearest substations outperform their counterparts choosing the farthest substations. Among the former, strategies using Euclidean distances work significantly better than strategies involving BFS. In Figure 2 (right) we depict ratios of solution values compared to `CollectingDijkstraAny`. The minimum ratios are between 0.65 and 0.72 for all initialization strategies other than `DijkstraAny`, which is at 0.971. The maximum ratios range between 1.05 and 1.10 for `DijkstraAny`, `BFSLast`, and `CollectingBFSLast` and are around 1.17 for all others. We see that for the main part there is hardly any difference between collecting strategies and their non-collecting counterparts. It stands out that on roughly 40 % of all instances `CollectingDijkstraAny` is better than `BFSAny` and `CollectingBFSAny` by 0.5 % and better than `BFSLast` and `CollectingBFSLast` by 1 %. `CollectingDijkstraAny` has a slight but not significant advantage over `DijkstraAny`. We therefore declare `CollectingDijkstraAny` paired with `IncDec` as our best variant.

Table 3 shows the running time characteristics of `CollectingDijkstraAny` paired with `IncDec`. Running times range between tenths of milliseconds to under two minutes.

Table 2 Comparison of the initialization strategies when the delta strategy `IncDec` is fixed. An entry in row i and column j shows on how many instances strategy i produces better solutions than strategy j. Values are marked by a star if they are significant with $p < 10^{-2}$ and by two stars if $p < 10^{-4}$. The best strategy is marked in green.

	`Dijk-A`	`BFS-A`	`C-Dijk-A`	`C-BFS-A`	`Dijk-L`	`BFS-L`	`C-Dijk-L`	`C-BFS-L`
`Dijk-A`	—	65.5 %**	47.5 %	66.8 %**	88.9 %**	76.7 %**	85.3 %**	76.1 %**
`BFS-A`	34.5 %	—	32.7 %	50.2 %	70.2 %**	66.2 %**	68.5 %**	64.4 %**
`C-Dijk-A`	52.5 %	67.3 %**	—	66.9 %**	88.7 %**	78.1 %**	88.4 %**	77.9 %**
`C-BFS-A`	33.2 %	49.8 %	33.1 %	—	71.5 %**	64.7 %**	69.6 %**	65.7 %**
`Dijk-L`	11.1 %	29.8 %	11.3 %	28.5 %	—	41.9 %	46.7 %	42.6 %
`BFS-L`	23.3 %	33.8 %	21.9 %	35.3 %	58.1 %**	—	56.6 %*	51.5 %
`C-Dijk-L`	14.7 %	31.5 %	11.6 %	30.4 %	53.3 %	43.4 %	—	43.5 %
`C-BFS-L`	23.9 %	35.6 %	22.1 %	34.3 %	57.4 %*	48.5 %	56.5 %*	—

Table 3 Minimum, average and maximum of running times in milliseconds of the best NCC variant `IncDec`, `CollectingDijkstraAny`. Running time measurement starts before the initial flow is computed and ends with the termination of the algorithm prior to outputting the solution.

	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\mathcal{N}_5$
min	0.46	3.40	183	3.0k	1.9k
avg	36.0	53.2	717	27.3k	14.0k
max	215	314	3.1k	89.6k	106k

5.2 Comparing our Best Variant with Gurobi

We compare our algorithm in its best variant, i. e., `CollectingDijkstraAny` with `IncDec`, with Gurobi on a MILP formulation which uses a binary variable for each edge and cable type to model the step cost function. We randomly select 200 instances per benchmark set from the benchmark sets in [17].

In Figure 3 we plot the ratio of the best solution value found by our algorithm to Gurobi's best solution at running times of two seconds, one hour, and one day for each benchmark set separately. As mentioned before, these running times represent both interactive and non-time-critical planning. Since our algorithm terminates in under two minutes, the comparisons in Figure 3 (middle and right) use the solution our algorithm provides at termination. While discussing the plots, we also discuss the so-called relative gaps, a standard notion from Mixed-integer Linear Programming. From the one-day MILP experiments we know a proven lower bound (lb) on the optimal solution value for each instance. For any instance, any maximum running time and for both the MILP and the NCC algorithm take best solution value (ub) found at the maximum running time and compute $^{\text{ub}-\text{lb}}/_{\text{ub}}$. This value is in the unit interval and gives information on how "bad" the solution value can be compared to the (unknown) optimal value. Note, however, that a solution might be optimal even though the gap is positive. We refer to the relative gaps as MILP *gap* and *NCC gap*, respectively.

After two seconds our algorithm outperforms Gurobi on all benchmark sets except $\mathcal{N}_4$ and produces better solutions on 89 % of the instances across all benchmark sets. On $\mathcal{N}_1$ the NCC gaps are on average 14.1 % with a maximum of 24.8 % compared to MILP gaps of 16.9 % on average and at most 43 %. For $\mathcal{N}_3$, the NCC gaps are on average 37.8 % with a spread of only seven percentage points, compared to a mean of 34.6 % and a maximum of 45.4 % for the MILP gap. The values for $\mathcal{N}_2$ range between those for $\mathcal{N}_1$ and $\mathcal{N}_3$. The ratios of solution values range between 0.699 and 1.019 for $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$. On $\mathcal{N}_4$, which contains the largest instances, our algorithm computes better solutions on 62 % of the instances.

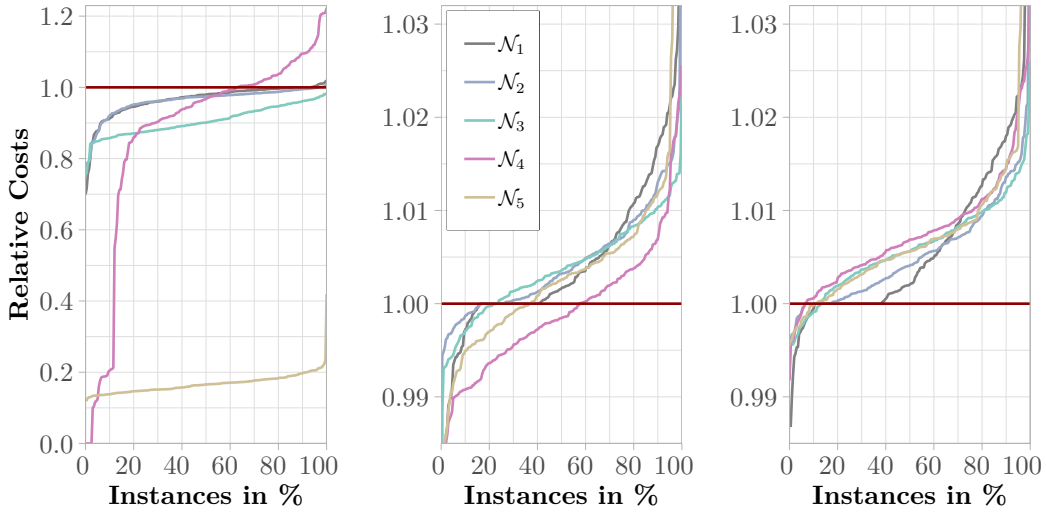

Figure 3 Comparison of the NCC algorithm to Gurobi on 200 instances per benchmark set. The ordinate shows the ratio of objective values at various maximum running times of our algorithm to objective values of Gurobi. Running times: *Left:* two seconds, *Middle:* one hour, *Right:* one day.

However, on six instances Gurobi does not find a solution. The instances on which Gurobi is better are on average larger than the other instances in $\mathcal{N}_4$. There are 16 instances on which the ratio of solution values exceeds 1.1 with a maximum of 1.223. On those very large instances, detecting negative cycles takes longer and fewer iterations are performed in two seconds. The NCC gaps spread between 31.4 % and 57.4 % with an average of 42.6 %. The MILP gaps are even worse with a mean value of 48.2 % and 18 instances above 88.5 %. On the complete graphs of $\mathcal{N}_5$, our algorithm produces solutions that are at least 75 % cheaper than Gurobi's on all but one instance (which has a ratio of 0.420). The gaps, however, are on average at 53.2 % for the NCC algorithm and at 99.2 % for Gurobi. The large spread in solution values indicates that both Gurobi and our algorithm seem unable to reliably find good solutions within a maximum running time of two seconds.

Within one hour (middle plot in Figure 3) Gurobi finds better solutions than our algorithm on a majority of the instances in all benchmark sets except $\mathcal{N}_4$. There our algorithm still yields better solutions in 57.5 % of the instances. However, our algorithm is within 1 % of Gurobi's best solution on 85.9 % and within 2 % on 96.8 % of all instances. Only on four of 1000 instances (all in $\mathcal{N}_5$), the ratio exceeds 1.10 with a maximum of 1.203. That means, our algorithm is comparable to Gurobi in solution quality but much faster since it terminates in under two minutes. We make similar observations for running times of one day (right plot in Figure 3). While our algorithm is at least as good as Gurobi only on between 7 % ($\mathcal{N}_4$) and 38.5 % ($\mathcal{N}_1$) of the instances, it is within 1 % of Gurobi's solution on 77.6 % of all instances. Again, there are only four instances with a ratio worse than 1.10 with a maximum of 1.210. Our algorithm does not profit from long running times since it gets stuck in local minima. Thus, the MILP solver is the better choice if more time is available. Between running times of one hour and day, the gaps look vastly the same and there is hardly any difference between NCC gaps and MILP gaps. They range between zero and 25.0 % on $\mathcal{N}_1$, clot around 28 % for $\mathcal{N}_3$ and $\mathcal{N}_4$ and around 34 % for $\mathcal{N}_5$ with seven outliers to the worse by the NCC algorithm.

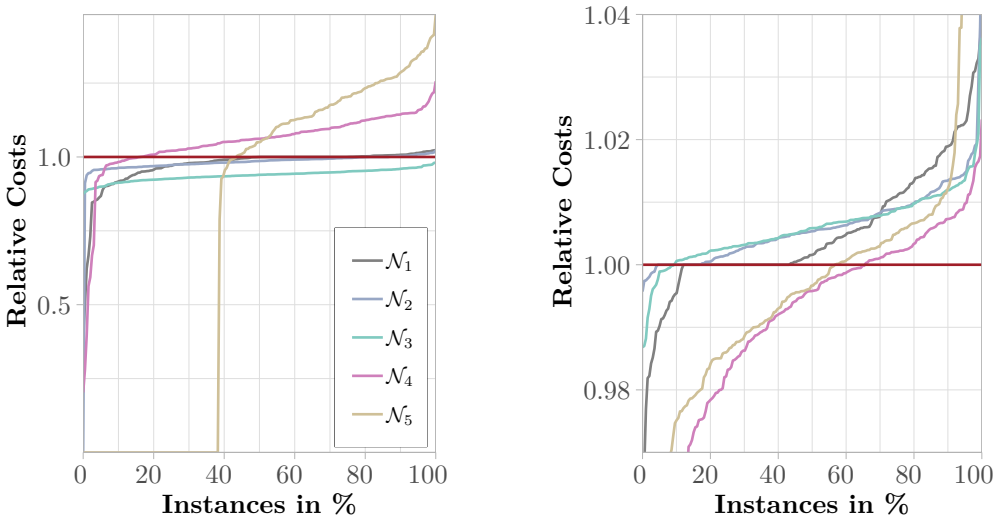

Figure 4 Comparison of Negative Cycle Canceling algorithm to the Simulated Annealing algorithm on 200 instances per benchmark set. The ordinate represents the ratio of objective values at different maximum running times of our algorithm to objective values of the Simulated Annealing algorithm. *Left:* Running time of two seconds. *Right:* Running time of one hour.

5.3 Comparison to Metaheuristic Simulated Annealing

We compare our best algorithm variant with the best variant of a Simulated Annealing (SA) algorithm [17]. We run the SA algorithm on 200 randomly selected instances per benchmark set (independently selected from other experiments). We compare the best solutions found after two seconds and one hour (Figure 4). After two seconds, our algorithm outperforms the SA algorithm on all instances from $\mathcal{N}_3$ and on 74.5 % and 86.5 % on $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. The minimum ratios are 0.381 for $\mathcal{N}_1$ and around 0.90 for $\mathcal{N}_2$ and $\mathcal{N}_3$ with one instance in $\mathcal{N}_2$ where the SA algorithm does not find a solution. The maximum ratio on those benchmark sets is at most 1.024. On the larger instances of $\mathcal{N}_4$ and $\mathcal{N}_5$, our algorithm presumably cannot perform sufficient iterations, as the SA algorithm is better on 70 % of those instances. Yet, the SA algorithm does not find feasible solutions on 38.5 % of instances from $\mathcal{N}_5$. The ratios have a wide spread: from 0.203 to 1.256 for $\mathcal{N}_4$ and 0.788 to 1.482 for $\mathcal{N}_5$ (save for the instances without a solution from the SA algorithm). After one hour, the SA algorithm provides better solutions than our algorithm on 83 % and 90.5 % of instances from $\mathcal{N}_2$ and $\mathcal{N}_3$, respectively. Our algorithm, however, stays within 1 % in solution quality on 77 % on the benchmark sets $\mathcal{N}_1$–$\mathcal{N}_3$. Again, our algorithm seems to be stuck in local minima. On $\mathcal{N}_4$ and $\mathcal{N}_5$, our algorithm performs better than the SA algorithm on 65.5 % and 56.5 %, respectively. Apparently, the SA algorithm needs more time to explore the solution space. The minimum ratios of solution values are as low as 0.716 for $\mathcal{N}_1$ and between 0.908 and 0.996 for the other benchmark sets. The maximum ratios are at most 1.055 for all benchmark sets except $\mathcal{N}_5$ (1.179). This supports our findings from the MILP experiments that our algorithm is competitive to other approaches to solving WCP within very short amounts of time. In view of an interactive planning process, it stands out that the SA algorithm struggles to find solutions quickly in dense graphs.

6 Conclusion

Based on recently presented ideas [14] we propose and compare numerous variants of a Negative Cycle Canceling heuristic for the Wind Farm Cabling Problem. While all variants run in the order of milliseconds up to 4.5 minutes, they differ significantly in quality. We identify the best variant and use it to compare our heuristic to the MILP solver Gurobi and a Simulated Annealing algorithm from the literature. With these comparisons we are able to solve several open questions [14]. While the MILP solver Gurobi has the potential to find optimal solutions if it runs long enough, our heuristic is able to find solutions of comparable quality in only a fraction of the time. Our algorithm beats Gurobi in finding good solutions in a matter of seconds. We make similar observations when we compare ourselves with a Simulated Annealing approach.

Moving forward, one may investigate how to improve the solution quality of our heuristic. Visually comparing flows from our algorithm and other solution methods may help to identify what kind of more complex circulations improve the solution. It then remains to investigate how these circulations can be detected. Also, methods for escaping local minima such as temporarily allowing worse solutions could help to improve our algorithm. It also remains open whether one can prove any theoretical guarantees on the solution quality or the number of iterations. Along the same lines, any theoretical insights on why one delta or initialization strategy works better than another, or on the order in which cycles should be canceled could help improve the NCC algorithm.

In a broader algorithmic view, the heuristic can be easily generalized to minimum-cost flow problems with other types of cost functions provided that one searches for integral flows. It would be interesting to see how well the heuristic performs there.

References

1 4C Offshore Ltd. *Hornsea Project Three Offshore Wind Farm*, 2018. Accessed: 2018-08-15. URL: www.4coffshore.com/windfarms/hornsea-project-three-united-kingdom-uk1k.html.

2 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, Upper Saddle River, NJ [u.a.], 1993.

3 Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958. doi:10.1090/qam/102435.

4 Constantin Berzan, Kalyan Veeramachaneni, James McDermott, and Una-May O'Reilly. Algorithms for cable network design on large-scale wind farms. MSRP technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 2011. http://thirld.com/files/msrp_techreport.pdf, Accessed: 2016-01-04.

5 Boris V. Cherkassky and Andrew V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2):277–311, June 1999. doi:10.1007/s101070050058.

6 Ouahid Dahmani, Salvy Bourguet, Mohamed Machmoum, Patrick Guerin, Pauline Rhein, and Lionel Josse. Optimization of the Connection Topology of an Offshore Wind Farm Network. *IEEE Systems Journal*, 9(4):1519–1528, 2015. doi:10.1109/JSYST.2014.2330064.

7 Mauricio C. de Souza, Philippe Mahey, and Bernard Gendron. Cycle-based algorithms for multicommodity network flow problems with separable piecewise convex costs. *Networks*, 51(2):133–141, 2008. doi:10.1002/net.20208.

8 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959. doi:10.1007/BF01386390.

9 Lester R. Ford, Jr. and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, NJ, USA, 2010.

10 Virginie Gabrel, Arnaud Knippel, and Michel Minoux. Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25(1):15–23, 1999. doi:10.1016/S0167-6377(99)00020-6.

11 Andrew V. Goldberg and Tomasz Radzik. A heuristic improvement of the Bellman-Ford algorithm. *Applied Mathematics Letters*, 6(3):3–6, 1993. doi:10.1016/0893-9659(93)90022-F.

12 Andrew V. Goldberg and Robert E. Tarjan. Finding Minimum-cost Circulations by Canceling Negative Cycles. *Journal of the ACM*, 36(4):873–886, October 1989. doi:10.1145/76359.76368.

13 Donald Goldfarb, Jianxiu Hao, and Sheng-Roan Kai. Shortest Path Algorithms Using Dynamic Breadth-First Search. *Networks*, 21(1):29–50, 1991. doi:10.1002/net.3230210105.

14 Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. Towards negative cycle canceling in wind farm cable layout optimization. In *Proceedings of the 7th DACH+ Conference on Energy Informatics*, volume 1 (Suppl 1). Springer, 2018. doi:10.1186/s42162-018-0030-6.

15 Longkun Guo and Peng Li. On the Complexity of Detecting k-Length Negative Cost Cycles. In *Combinatorial Optimization and Applications, COCOA 2017*, volume 10627 of *Lecture Notes in Computer Science*, pages 240–250. Springer International Publishing, 2017. doi:10.1007/978-3-319-71150-8_21.

16 Morton Klein. A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems. *Management Science*, 14(3):205–220, 1967. doi:10.1287/mnsc.14.3.205.

17 Sebastian Lehmann, Ignaz Rutter, Dorothea Wagner, and Franziska Wegner. A Simulated-Annealing-Based Approach for Wind Farm Cabling. In *Proceedings of the Eighth International Conference on Future Energy Systems*, e-Energy '17, pages 203–215, New York, NY, USA, 2017. ACM. doi:10.1145/3077839.3077843.

18 Sara Lumbreras and Andres Ramos. Optimal Design of the Electrical Layout of an Offshore Wind Farm Applying Decomposition Strategies. *IEEE Transactions on Power Systems*, 28(2):1434–1441, 2013. doi:10.1109/TPWRS.2012.2204906.

19 New York State Energy Research and Development Authority. *New York State Offshore Wind Master Plan*, 2017. Accessed: 2018-08-15. URL: https://www.nyserda.ny.gov/-/media/Files/Publications/Research/Biomass-Solar-Wind/Master-Plan/Offshore-Wind-Master-Plan.pdf.

20 Adam Ouorou and Philippe Mahey. A minimum mean cycle cancelling method for nonlinear multicommodity flow problems. *European Journal of Operational Research*, 121(3):532–548, 2000. doi:10.1016/S0377-2217(99)00050-8.

21 Christos H. Papadimitriou. The complexity of the capacitated tree problem. *Networks*, 8(3):217–230, 1978. doi:10.1002/net.3230080306.

22 Tomasz Radzik and Andrew V. Goldberg. Tight bounds on the number of minimum-mean cycle cancellations and related results. *Algorithmica*, 11(3):226–242, March 1994. doi:10.1007/BF01240734.

23 Pedro Santos Valverde, António J. N. A. Sarmento, and Marco Alves. Offshore Wind Farm Layout Optimization – State of the Art. *Journal of Ocean and Wind Energy*, 1(1):23–29, 2014.

24 WindEurope asbl/vzw. *Wind in power 2017*, 2018. Accessed: 2018-08-15. URL: https://windeurope.org/wp-content/uploads/files/about-wind/statistics/WindEurope-Annual-Statistics-2017.pdf.

25 Menghua Zhao, Zhe Chen, and Frede Blaabjerg. Optimization of Electrical System for a Large DC Offshore Wind Farm by Genetic Algorithm. In *Proceedings of NORPIE 2004*, pages 1–8, 2004.