Cable Layout Optimization Problems in the Context of Renewable Energy Sources

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)
genehmigte

Dissertation

von

Sascha Frederik Gritzbach

Tag der mündlichen Prüfung:	24. April 2023
Erste Referentin:	Prof. Dr. Dr. h. c. Dorothea Wagner
Zweiter Referent:	Univ.Prof. Dr. Martin Nöllenburg
Dritter Referent:	Prof. Dr. Veit Hagenmeyer

Danksagung

Mein erster Dank geht an Dorothea Wagner für die wissenschaftliche Betreuung dieses Promotionsprojekts und dafür, dass sie 2017 meiner Initiativbewerbung eine Chance gegeben hat. Mein zweiter Dank (und dann höre ich auf zu zählen!) geht ebenfalls an Dorothea Wagner und an die Organisationen und dazugehörigen Personen, die dieses Projekt finanziell abgesichert und wertvollen wissenschaftlichen Austausch durch Konferenzen, Workshops und vieles mehr unterstützt haben: die Helmholtz-Gemeinschaft in den Programmen Energiesystemintegration (ESI) und Energy System Design (ESD), sowie die Deutsche Forschungsgemeinschaft mit dem Graduiertenkolleg 2153: „Energiezustandsdaten – Informatik-Methoden zur Erfassung, Analyse und Nutzung“ unter der Leitung von Klemens Böhm. Danke an die beiden Referenten Martin Nöllenburg und Veit Hagenmeyer, dass sie sich bereiterklärt haben, diese Arbeit zu begutachten, und danke an Uwe Schöning, der mich während meiner Suche nach einer Promotionsstelle auf Schloss Dagstuhl aufmerksam gemacht hat, worüber ich letztlich meinen Weg zur Arbeitsgruppe von Dorothea Wagner gefunden habe.

Ich möchte mich bei all denen bedanken, mit denen ich über die Jahre zusammenarbeiten durfte: den Kolleginnen und Kollegen auf dem Gang, für die ich stellvertretend meine Bürokollegen Moritz Baum, Matthias Wolf und Tim Zeitz und die Energiekollegen Lukas Barth, Max Göttlicher, Franziska Wegner und (nochmal) Matthias Wolf nennen möchte; den Menschen, ohne die nichts funktionieren würde: Lilian Beckert, Andreas Hofmann, Isabelle Junge, Ralf Kölmel, Laurette Lauffer, Bernadette Lehmann und Tanja Wehrmann; meinen Ko-Autorinnen und Ko-Autoren Kemal Çakmak, Veit Hagenmeyer, Pascal Mehnert, Dominik Stampa, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner und Matthias Wolf, sowie Anselm Erdmann und Carolin Ulbrich, stellvertretend für die Kolleginnen und Kollegen aus ESI; und allen anderen, die nicht in eine dieser Gruppen fallen.

Danke an alle, die mich durch schlaue Hinweise und ausführliches Feedback beim Verfassen dieser Arbeit unterstützt haben: Pascal Beckedorf, Sonja Feger, Vivi Heinzel, Paul Jungeblut (insbesondere für das Finden einer Beweislücke), Jonas Sauer, Matthias Wolf (insbesondere auch für das Mithelfen beim Füllen der Lücke) und Anna Zinn. Danke an alle, mit denen ich mein Leid teilen und denen ich von spannenden Funden in der Literatur berichten durfte. Ganz besonderen Dank an alle, die mich die letzten Jahre und vor allem auch in der „heißen Phase“ emotional auf die verschiedensten Arten unterstützt haben: Pascal Beckedorf, Guido Brückner (für unzählige Stunden im „Tempel“), Sonja Feger (insbesondere auch dafür, dass wir uns gemeinsam durch die Pandemie kämpfen konnten) und Martha Frysztacki.

Und den wichtigsten Leuten ganz zum Schluss: Von tiefstem Herzen danke ich meinen Eltern Edwin und Elvira Gritzbach, meiner Oma Agnes Reinwald (danke, dass Ihr mir Deine Weihnachtsplätzchen sogar in die USA geschickt habt – einmal zu Weihnachten und einmal im Februar, weil ich nicht genug davon bekommen kann!) und Vivi, für die ich meine unendliche Dankbarkeit in Worte zu fassen nicht imstande bin! Danke, dass Ihr immer für mich da seid, auch wenn wir nicht am selben Ort sind.

Und weil mir beim Schreiben dieser Worte fast die Tränen vor Rührung kommen, let us switch to English and talk about “Cable Layout Optimization Problems in the Context of Renewable Energy Sources”, hoping that I have not forgotten anyone in these acknowledgments.

Abstract

One of the United Nations' 17 Sustainable Development Goals is to "ensure access to affordable, reliable, sustainable and modern energy" with a target to "increase substantially the share of renewable energy in the global energy mix" by 2030 [UN15]. Essential for achieving this target is the construction of new power plants of various types, among those are large-scale (offshore) wind and (open-field) solar power plants. These power plants share a general layout idea: Many decentrally placed generators harvest the renewable energy source and the generation is transmitted via a system of cables to transformers. From there the electricity is supplied to consumers. Historically many of these cable systems have been planned and layouted manually with the help of predefined templates. More recently, algorithmic support comes into focus in view of the growing sizes of the power plants.

In this doctoral thesis, we focus on bringing together the design of installation-cost-optimized cable layouts in wind and solar farms and a classical notion from Theoretical Computer Science: Network Flows.

At the core of these cable layout optimization problems is a special cost function that originates from the assumption that for any connection one of several available cable types as to be chosen. Each cable type comes with a cost per unit of length, so that the resulting cost function looks like a flight of stairs and is therefore called *step cost function.* Thus, both optimization problems considered here can be interpreted as Minimum-Cost Flow Problems with a Step Cost Function. We provide theoretical results on our optimization problems: We show strong NP-hardness and an analogon to the well-known Integer Flow Theorem. Those results both connect our problems to and set them apart from classical Minimum-Cost Flow Problems with linear costs.

Abstract

For cable layouts in wind farms, we engineer an algorithm based on Negative Cycle Canceling and evaluate its performance both theoretically and empirically, the latter by means of simulations on synthetic benchmark instances from the literature in comparison to existing solution approaches. We also show how our algorithm can be incorporated into a framework of Iterated Local Search. The evaluation of this extension is based on the synthetic instances, as well as on a case study on one of the biggest offshore wind farms in the world, the Hornsea One Wind Farm. Since the optimization problem we employ is rather simplistic compared to other models from the literature, we conclude our elaborations on wind farms with a structural and electrical analysis of the cable layouts computed by our algorithms. The goal is to investigate potential shortcomings of the simpler model, so that interested wind farm planners more easily judge the implementability of our cable layouts.

For cable layouts in solar farms, existing literature is surprisingly scarce compared to literature on wind farms. In order to facilitate future research in this field, we introduce one of the first problem formulations for the design of (installation-)cost-optimal cable layouts in solar farms. This formulation provides a multitude of possibilities for further extensions to cater for solar farm planners' needs. It also bears resemblance to various more classical optimization problems which may serve as inspiration for the development of algorithmic approaches. To support this development, we propose a framework for the generation of synthetic benchmark instances and populate this framework by parameters from real-world solar farms. The resulting instances are publicly available. Additionally, we state a Mixed-Integer Linear Program formulation, evaluate it on our instances and provide the results as a baseline for yet-to-be engineered algorithms.

Contents

1 Introduction

> Fortune is guiding our affairs better than we could have wished; for you see there before you, friend Sancho Panza, some thirty or more lawless giants with whom I mean to do battle.
>
> —*Don Quijote de la Mancha* [Cer15, p. 75][Cer61, p. 61]

What sounds like famous last words, is but an introduction to one of many adventures of Don Quijote de la Mancha in Miguel de Cervantes Saveedra's masterpiece. Ignoring all of his friend's warnings that the giants are merely windmills, Don Quijote, mounted on his horse Rocinante and armored with shield and lance, charges at the giants. With the wings of the windmills turning in the wind, the lance is shattered into pieces and both rider and horse are flung to the ground, leaving them "very much battered indeed" [Cer15, pp. 75–76][Cer61, pp. 61–62]. Thus was born the idiom "tilting at windmills" to describe the pursuit of a vain goal [Amm97].

Tackling the climate crisis is not like tilting at windmills.[1] In fact, windmills—or rather wind turbines—are considered part of the solution: The European Commission has set a goal of 60 GW of installed offshore wind energy capacity by 2030 and of 300 GW by 2050, estimated to cost up to 800 billion euros [EC20]. As of 2020, a total of 12 GW of offshore capacity is in service in the EU-27 states and another 10 GW in the United Kingdom (UK). By the end of 2022, additional 2.8 GW and 2.5 GW are expected to be commissioned in the EU-27 and the UK respectively [Tel+20].

[1] There are actually two meanings to "tilting at windmills", the other being "fighting an imagined enemy" [Amm97], which is closer to the origin in Cervantes' masterpiece. It goes without saying that the climate crisis is real.

The fastest growing renewable energy source however is Photovoltaic (PV) [Cha+22]. In March 2022, the cumulative installed PV capacity reached 1 TW_p[2] worldwide. The European Union contributes approximately one sixth of the total installed capacity with 165 GW_p at the end of 2021, seeing a 25 % increase in newly installed capacity (26 GW_p) compared to the year before [Cha+22]. Photovoltaic power stations come in different scales and forms such as tiny systems powering a road sign via small rooftop systems on private homes to utility-scale solar farms. The world's biggest solar farm (as of February 2020) is the Bhadla Solar Park with a capacity of more than 2.2 GW [San20].

Utility-scale solar farms and offshore wind farms are conceptually surprisingly similar: A multitude of decentralized generators provide electrical energy that is transmitted to a central point (e. g. an offshore substation) from where it is fed into the electrical grid to be made available for consumers. Responsible for the transmission is a system of cables with potentially additional electrical equipment.

The focus of this work is on the optimization of the layout of such a system of cables. We consider the high-level scenario that for each connection of any two devices, a planner of such a decentralized power station can choose one cable from a set of cable types. Each cable type has a thermal capacity and a per-unit cost and the costs of a cable layout arise solely from the cumulated costs of all cables across all connections.[3] This yields a step cost function when we consider the costs as a function of transmitted power. Such a step cost optimization function is algorithmically challenging.

The overarching motivation of this dissertation is to bring Energy Informatics and Theoretical Computer Science closer together: The goal is to solve cable layout optimization problems with the toolbox centered around network flow algorithms.

We engineer an algorithm based on Negative Cycle Canceling, which is a classical minimum-cost flow algorithm, to tackle the cable layout optimization in wind farms, and we show that our algorithm is well-suited to deal with the challenges arising from the aforementioned step cost function. For solar farms, we provide one of the first problem definitions for cable layout optimization in order to facilitate future algorithmic research in this field.

More details on the contribution of this dissertation are provided in Chapter 2 following descriptions on the inner workings of wind and solar farms as well as an extensive review of the literature.

[2] Watt-Peak (W_p) is a non SI-unit which describes the rated power of a solar module under so-called *Standard Test Conditions* [Mer19, p.15].

[3] While this may sound restrictive, there is actually some flexibility on what can be represented by the costs of such cable types, see our discussion in Section 4.10.

2 Literature Overview and Our Contribution

In Chapter 1 we mentioned that "utility-scale solar farms and offshore wind farms are conceptually surprisingly similar" in the sense that the output of decentralized generators must be transmitted to a central point. In this chapter, we shed light into the differences by explaining in more detail, how these power plants work. For both types, we give an overview of algorithmic research and state the contribution of our work. As such, this chapter is divided into two main parts: Section 2.1 on offshore wind farms and Section 2.2 on solar farms. Section 2.3 gives an overview of optimization problems that are similar to the cable layout problems in wind or solar farms or that have been tackled using similar techniques as the ones applied in this work. The chapter concludes in Section 2.4 with an outline how the remainder of this thesis is structured.

2.1 Offshore Wind Farms

In a wind farm, the decentralized generators are wind turbines, which convert wind energy into electrical energy. Between these turbines, a system of cables (referred to as *internal cabling, inter-array cable routing* or simply *cable layout*) transmits the generated power to one of possibly multiple substations. These substations are the gateway to feed the generated power into the electrical grid where it is available to consumers. In the case of offshore wind farms, a high-capacity export cable connects the offshore substation(s) to an onshore grid point.

The planning process of a new wind farm consists of multiple phases and includes, among others, the placement of turbines, the design of the turbine foundations, and the design of the cable routing. The placement of the substation(s) and the connexion to the grid point may also be part of the process or it may be already be established [FP19]. The survey [Hou+19] mentions two works on co-optimization but seems rather critical. Due to the complexity in each phase alone, research focuses mainly on one phase at a time [VSA14].

For literature on the optimal placement of turbines (*micro-siting*), we refer the interested reader to the survey [Hou+19]. In this phase, planners face a dilemma: While more turbines closer together increase the nominal output of a wind farm, in reality turbines placed downwind from other turbines experience reduced wind speeds and therefore lower energy yields. This is known as the *wake effect* [Hou+19, Sec. 3.1].

To learn about the design of turbine foundations, the reader may want to refer to [FP19].

We focus on the design of the internal cabling, as this is the topic of this work. Historically, planners choose from one of several standard patterns for the cable layout. These patterns can be evaluated under various technical aspects [QAAM07] and the planner picks the best layout according to their priorities.

With the emergence of algorithmic approaches in wind farm planning, the set of turbines and substations are modelled as vertices of a graph embedded into the Euclidean plane with edges representing possible connections between them. Anecdotally, the edge set is frequently assumed complete, meaning that any two vertices can be connected in the cable layout. A variety of exact, heuristic and metaheuristic optimization approaches have been developed to find the best possible layout with respect to different cost functions or constraints included in the respective optimization problem. A recent survey [PC19] gives an overview of different approaches, highlighting the differences in objective functions and constraints being accounted for in the literature. The authors note that roughly half of the papers in their consideration employ metaheuristics, so our overview starts with those as well:

On the metaheuristic side, Genetic Algorithms are the most used technique (34 % of all literature surveyed in [PC19]). To name but a few examples: They are used to find the best radial layout (i. e., the layout consists of disjoint paths originating at a substation) under minimal investment costs for cables and substations [Dah+15, GWRT12] or to find the best branched layout with a non-predetermined positions of the substations [Sed+16]. Particle Swarm Optimization involving Minimum Spanning Tree computations has been employed to find a tree-layout in which no cables may cross [HHC16]. In this setting, multiple cables between two turbines are allowed at the same time. To the contrary, only one cable is allowed between any two turbines in [LRWW17] and cable crossings are allowed. The authors propose to use Simulated Annealing (SA) to compute arbitrary layouts and evaluate their algorithm against a Mixed-Integer Linear Program (MILP) formulation on their own set of synthetic benchmark instances with up to 500 turbines using cable types based on a real-world wind farm. The use of synthetic instances enables the authors to evaluate their approach on instances bigger than any existing wind farm. Real-world instances are used for the evaluation of a Large Neighborhood Search in [CP22] where the focus of the optimization model lies on avoiding obstacles and ensuring that branches rooted at substations connect approximately equal number of turbines. A computational study comparing various metaheuristics is provided in [CFF20] for a model involving one substation only but including various constraints: tree-structure[1], avoidance of cable crossings, and a limit on the degree of the substation.

[1] Strictly speaking, forests in case of multiple substations. For simplicity, however, we stick to the word "tree".

The cable types used in [LRWW17] stem from the proposed Black Nubble Wind Farm in Maine, USA [DO11] and have been translated into a suitable cost function in [BVMO11]. These papers lead us to heuristic and exact approaches: Dutta et al. [DO11] propose multi-level clustering to group the wind turbines according to their Euclidean distance. In each level, turbines within a cluster are connected by a star layout to the respective cluster's representative turbine. The approach is evaluated on a wind farm with 22 turbines and one substation. Berzan et al. [BVMO11] propose a hierarchical decomposition depending on the number of substations and, in the case of only one, the number of cables connected to it. Their model also includes maximum capacities at substations but does not mention cable crossings. When only one cable type is available, the subproblems arising from the decomposition are related to other optimization problems for which exact and heuristic algorithms already exist. A sequential heuristic for the highest-level problem ("Full-Farm Problem") is evaluated on instances with up to 1000 vertices. For the case that multiple cable types are available, the authors propose a Divide-and-Conquer algorithm and an Integer Linear Program to solve the "Circuit Problem", i. e., finding a cost-minimal tree-cable-layout among a set of turbines, out of which only one is connected directly to the single substation. While the solution must be a tree, the Integer Linear Programm does not seem to include any constraints to enforce this layout. It is noted that the Integer Linear Program is "too slow beyond" eight turbines [BVMO11, p. 12]. Bauer et al. [BHMP00] compute radial layouts without crossings by a reduction from the Planar Vehicle Routing Problem and by Integer Linear Programming with only binary variables. Cerveira et al. [CBP14] use a problem formulation that translates into a Capacitated Minimum Spanning Tree and establish an MILP formulation that includes both the number of turbines in a subtree as well as the resulting electric current. The simulation on a 25-turbine wind farm is terminated after 14 hours with a gap of 5.6 %[2]. The computational difficulty of large MILP formulations is overcome in [FP18] by combining heuristic steps with an MILP solver to find close to optimal solutions for the "Full-Farm Problem" on real-world based instances with up to 100 turbines. Their model includes constraints to enforce a tree-structure, a maximum number of cables connected to any substation, and the absence of cable crossings. The model also includes Steiner vertices, i. e., vertices that are neither turbines nor substations, which are used to avoid obstacles on the seabed.

To add an element of reliability to the layout, some literature considers finding cable layouts with loops, see the survey [PC19, Chapter II, Section A.4]. For the most part, however, the focus in the literature lies on tree-layouts, as we have outlined above.

While several of the works on tree-layouts implicitly model the flow inside a cable (e. g., [FP18, CBP14]—the first measures flow based on arbitrary but fixed turbine productions, and the second with electric current proportional to the number of turbines connected via that cable), the flow is uniquely determined due to the tree-structure. As such, the task of designing a cable layout can also be interpreted as a classical Minimum-Cost Flow Problem (albeit with possibly non-classical cost functions or side constraints), which opens a toolbox of many well-researched classical algorithms. The interpretation as a Minimum-Cost Flow Problem, in which the cost

[2]We define the notion of MIP gaps in Section 4.3

function is a "stair-case" can also be found in [LRWW17] but their flow problem is solved using Simulated Annealing and an MILP formulation. We are not aware that the adaptability of classical flow problems to compute cable layouts in wind farms has been investigated—be it theoretically or by experimental evaluation.

Contribution 1. *We adapt the Minimum-Cost Flow Problem formulation from [LRWW17] and prove both strong NP-hardness[3] and an adaptation of the well-known Integer-Flow Theorem.*

Contribution 2. *We engineer an algorithm based on the classical Negative Cycle Canceling (NCC)-technique and investigate its performance both theoretically and by experimental evaluation.*

We give a thorough explanation of this fundamental technique in Section 3.3. One key theorem in the theory of Negative Cycle Canceling is a characterization of (cost-)optimal flows: A flow is optimal if and only if no negative cycles exist in a suitable auxiliary graph (Theorem 1). NCC has been applied in non-classical Minimum-Cost Flow settings. In some settings the optimality criterion remains true [OM00], in other only a local version can be shown [MS07, SMG08]. In [SMG08], NCC is thus enhanced by strategies to deal with local minima. The cost functions in those works differ from our step cost function and thus, it is not clear if the optimality criterion holds in our setting.

Contribution 3. *We describe and investigate the challenges arising from our cost function and find that the classical optimality criterion does not hold anymore. We show how to deal with these challenges by running the negative cycle detection algorithm on the linegraph of the input graph.*

The evaluation of our algorithm on the synthetic instances from [LRWW17] shows that our algorithm finds competitive solutions compared to an MILP solver within tens seconds on instances with up 500 turbines—in most cases even faster.

It is noted on [CFF20, p. 1] that in "practical applications involving preliminary what-if analyses, however, one is mainly interested in finding reasonably good (not necessarily optimal) solutions in very short computing times". Our algorithm fits this description nearly perfectly. A crucial point here may be, that the model in [CFF20] includes further technical constraints, mainly the tree-structure and the ban of cable crossings. Cable crossings are possible but need better insultation and higher maintenance cost in case of failures [BHMP00, p. 1]. Along similar lines, further electrical parameters might make the cable layouts computed by our algorithm unsuitable for real-world implementation.

Contribution 4. *We analyse the cable layouts computed by our algorithm under topological and electrical aspects. This brings the necessity to translate the cable layouts into electrical models, for which we propose a workflow that is also able to determine electrical input parameters from abstract input instances and cable types.*

[3] To be precise, Lehmann et al. [LRWW17] claim that the resulting Minimum-Cost Flow Problem is NP-hard due to the "stair-case" cost function. The connection to the referenced literature that would allow a straightforward reduction, however, is not imminently clear. Thus, we provide a self-contained proof.

The analysis shows that only one in approximately 56 layouts contains cycles and one in approximately 13 layouts contains cable crossings. The electrical parameters are in line with reference values from the literature, which gives evidence that the layouts computed by our algorithm may be suitable for electrical implementation.

The experimental evaluation reveals that while our algorithm computes competitive solutions in short amount of times, it also get stuck in local minima from which it cannot recover.

Contribution 5. *We embed our Negative-Cycle-Canceling-based algorithm into a framework of Iterated Local Search (ILS). For this ILS we develop strategies to escape local minima. The framework is again evaluated on the synthetic benchmark instances from [LRWW17].*

Contribution 6. *We also carry out a case study to evaluate the ILS. This study involves an instance resembling one of the biggest offshore wind farms in the world: Hornsea One.*

The evaluation and the case study show that the additional computation time is well invested. The share of instances from the ILS with better solutions than the MILP is greatly increased compared to our NCC algorithm, while still using less overall computation time than the MILP. Additionally, the number of instances with cable crossings in the best solutions found is reduced.

The evaluations of both our algorithms, the algorithm based on Negative Cycle Canceling as well as its incorporation into Iterated Local Search, prove that flow-based algorithms are indeed approaches to finding wind farm cable layouts worthwhile their consideration.

2.2 Solar Farms

As outlined in Section 2.1, in a wind farm the turbines are connected to each other with an eventual connection to a substation. In solar farms, the cable layout is more hierarchical with more than only two meaningful types of components as we explain in the following. The description of solar farms and the subsequent overview of the literature is based on joint work with Dominik Stampa and Matthias Wolf [GSW22b]. Further details on solar farms can be found in [ABB19, Mer19].

In a solar farm, the decentralized generators are solar cells, which convert sunlight into tiny amounts of electric current. The cells are connected forming a PV module. These modules, in turn, are connected in series to form a string which is mounted on a rack. For our purposes, we consider PV strings as the smallest building block, since we are mainly interested in optimizing the cable layout. The strings supply electricity in form of Direct Current (DC), which is converted to Alternating Current (AC) in inverters. Strings can have their own inverters (*string inverters*, connected to one or at most to a few strings) or a larger number of strings is connected to only a few *central inverters*. In general, inverters have more functions than only conversion. They are used for monitoring and safety purposes and generally also incorporate control elements such as maximum power point trackers, by which the DC voltage in the connected components is adjusted according to environmental conditions to maximize electric power harvest.

Solar farms typically operate at low-voltage levels (the DC side is usually aimed at a maximum voltage of 1 kV or 1.5 kV), so that step-up transformers are needed to feed the generated power into the grid. Since inverters have only a finite number of input circuits, additional devices are installed between strings and inverters. These devices have different names, depending on the monitoring and safety equipment installed in them and on the components they connect. *Y-connectors* are the most simple device. They normally connect just two strings with no additional equipment (except maybe for fuses [Eve16]). *Combiner boxes* connect a larger number of strings (or Y-connectors) and have additional safety and monitoring equipment. *Recombiner boxes* have the same equipment as combiner boxes but connect combiner boxes instead of strings. Which kind of components, in particular between strings and inverters, are used ultimately depends on a decision by the solar farm planners. In any case, the components need to be connected by cables. For electrical reasons the cable layout should be balanced to some extent, e. g. to avoid reverse current. In particular, the layered structure should be respected, for example, connecting a recombiner box (to which multiple strings are connected) and a single string to an inverter should be avoided. This layered structure is visualized on an abstract level in Figure 5.1.

With all those restrictions in mind, solar farms appear to be mostly constructed on flat ground following one (of maybe several) pre-specified templates. However, there are exceptions: The Monte Mele photovoltaic plant is situated on the slope of a hill on Sicily, Italy, and has irregular distances between its strings [Alp22]. For a solar farm of the size of the Monte Mele plant with its capacity of 718 kW, drawing a cable layout by hand might be feasible. For larger solar farms (as mentioned before, the world's largest stands at 2245 MW as of February 2020 [San20]), algorithmic approaches computing near-to-cost-optimal cable layouts might be the way to go.

Various aspects of solar farms have been a target for optimization in the literature. Solar cells can be manifactured from different so-called photovoltaic absorber materials that influence the performance of a cell, for a review see [KR18]. The efficiency of transformers can be influenced by using appropriate control methods, which are realized in a prototype [Liu+19]. Shifting the focus to the overall electrical system of a solar farm and its operation, a variety of maximum power point tracking techniques can be employed to maximize the power output of the farm [BMB20]. Concerning the early stages of the planning process, a fuzzy Analytic Hierarchy Process has been proposed to find an optimal site for a solar farm [TSMA17].

A more holistic view on solar farm design is employed at Siemens Energy [BEPS14]: In a multi-criteria decision support system a set of pre-computed designs for a given site can be compared and investigated by planners visually and with respect to different "key performance indicators" so that planners "obtain an overview on the whole solution space" [BEPS14, p. 338]. The designs are computed using several (unspecified) heuristics in a three-stage process involving three subproblems: placing service ways, placing strings in the area between ways, and inverter placement. Given the positions of strings and inverters, a cable layout is computed in a "single-objective manner, [minimizing] cable cross sections such that specified losses are not exceeded" [BEPS14, p. 337]. The exact optimization problem is not stated.

Using a different setting, a formalization of the optimization problem of finding a cost-minimal cable layout is given in [Luo+21]. Computing cable layouts and the placement of combiner boxes assigned to a single inverter is modelled as a generalized Capacitated Minimum Spanning Tree Problem and solved by a branch-and-price-and-cut algorithm. In this setting, strings are placed on a grid and edge lengths are given by the ℓ_1 metric. A capacitated spanning tree connecting strings and the inverter yields a cable layout in which a combiner box is placed at each child string of the inverter such that the capacity of the combiner box is not exceeded. As such, all strings are also candidate positions for the placement of combiner boxes. The costs arise from a linear-cost flow on edges between any two strings and from a step cost function (similar to the "stair-case" cost function arising from cable types in wind farms) between combiner boxes and inverter. This step cost function models the installation costs of the combiner boxes.

As seen above, there are further types of components that a solar farm planner may want to install. For those cases, a more general formulation of the optimization problem is needed.

Contribution 7. *We introduce the Solar Farm Cable Layout Problem (SoFaCLaP), a problem formulation that accounts for multiple types of electrical components in solar farms.*

SoFaCLaP is based on a network flow on a layered graph where each layer corresponds to one type of component. The solution space entails both the cable layout itself as well as the choice where to place the electrical components from a set of candidate positions. The problem formulation also includes cable types (as seen in the wind farms) and capacities for all types of components. The cable layout is enforced to be a forest such that the hierarchical structure of the components is observed: Any vertex may only have at most one outgoing connection and this connection must be to a vertex of the next layer. There is a high degree of flexibility for solar farm planners on modelling decisions, including but not limited to the types of components to include.

Contribution 8. *We propose an MILP formulation for SoFaCLaP to provide a first solution method that can serve as a baseline for other algorithms proposed later.*

Contribution 9. *We describe a framework to generate synthetic benchmark instances and provide a set of instances based on parameters from real-world solar farms. These instances are used to evaluate the MILP formulation and can be used for the development and evaluation of further algorithmic approaches. Both the set of instances as well as the results from the evaluation are publicly available [GSW22a, GSW22c].*

2.3 Similar Problems

In this section, we name further optimization problems that bear some resemblance to the cable layout optimization problems in solar and wind farms, as well as corresponding literature. In particular for solar farms, this references may be the starting point for further research, to which this thesis provides the problem formulation and means for the experimental evaluation.

In solar farms, the combination of a tree-layout with a set of optional vertices that can be used for routing the strings' generation to transformers bears resemblance to the Steiner Tree Problem on graphs. If the graph is directed and rooted, this problem is known as the Steiner Arborescence Problem on graphs [Lju21]: In a directed graph with given arc weights, given terminals, and a given root, find a directed subtree that connects all terminals to the root (*arborescence*) in a cost-minimal manner where the cost of a solution is given as the sum of all the weights of the arcs in the arborescence. The most obvious difference is that the cost of an edge does not depend on the number of terminals connected to the root via this edge. Special cases of Steiner Trees with costs depending on the transmitted flow have been considered as early as [Gil67]: A set of points in the plane shall be connected by straight lines forming a tree and the cost of a line arises from a given global function that maps flow to cost per unit of length. Separate structural results are provided for affine and convex cost functions. A special case with multiple sources and one sink (which is closer to solar farms) is considered in [Vol+13] where it is named the Gilbert Arborescence Problem. The problem is defined on general Minkowski spaces with special consideration given to cost functions of the form $w(t) = d + ht^{\alpha}$ with $d, h > 0$ and $\alpha \in (0, 1]$.

The layered structure of solar farms can be found in Multi-Level Facility Location Problems. A survey is provided in [OCL18]. Customers on the lowest level need to be connected to facilities on the next level, which in turn need to be connected to facilities on the level after that. The question arises which facilities to open and how to connect the open facilities. Generally speaking, costs can arise from opening facilities and connections (fixed) and linear transportation costs (variable). The authors make no mention of any work that considers more than two levels and capacities at the facilities, which is the case in our solar farm model. For the case of two levels with both upper and lower bounds at the facilities, several approximation and bicriteria approximation algorithms exist, see the survey [Rez22].

While the literature on wind farms mentioned in Section 2.1 consider offshore farms, there is also literature for the onshore case [Her+17]. In that model, the edge set of possible connections contains separate subsets for connections above- and underground. Tree-layouts are enforced but multiple cables of the same type may connect the same vertices. The authors propose a Quadratic Program and an equivalent but bigger Integer Program to model power losses into the cost function.

An emerging type of offshore wind farms are floating wind farms [Tel+20]. Such wind farms, in which cables "have a dynamic section that moves with the floating substructures [i. e., turbines and substations]" are considered in [LDM21]. The authors propose a metaheuristic to find cable layouts in a setting with stochastic wind speeds and directions. We come back to these two variants of wind farms in Section 4.10 to discuss to what extend our algorithm may be able to handle the aforementioned models.

A conceptually similar problem to the design of cable layouts of wind farms is the design of collector systems for offshore production of natural gas: The production from several offshore gas fields needs to be transported through pipelines to a delivery point on the shore [Rot+70]. The optimization problem enforces a tree-structure and the variables represent how drilling platforms are connected by which capacity the installed pipeline has. This is in line with the cable types used in wind farms. Brimberg et al. [Bri+03] formulate an MILP and employs both exact and metaheuristic solution approaches. A more extensive MILP formulation is provided in [Zha+17]. It includes for example facilities that increase the pressure in the pipelines.

Another conceptually similar problem is the design of water distribution networks: Vertices with water supplies and demands are connected by edges and for each edge one type of pipe out of several available ones must be chosen [DS16]. Contrary to wind farms, pipes must be installed on all edges. The model in [DS16] also includes the "energy conservation law", similar to Kirchhoff's voltage law in electrical circuits. The design problem of choosing installation-cost-optimal pipes to satisfy all supplies and demands is solved by an Iterated Local Search. Further metaheuristic approaches are referenced in [DS16] as well. Exact approaches have also been proposed in the literature, for example involving a Mixed-Integer Non-Linear Program [CCCR21].

Two more general problems in which production from many decentral vertices must be routed to a central sink are the Multilevel Capacitated Minimum Spanning Tree Problem and the Single-Sink Buy-at-Bulk Network Design Problem. In the former, the layout must be a tree and costs arise from installing one out of several available capacities on each edge. The objective is to minimize the total amount of installed capacity [GRG03]. The latter problem uses multiple cable types (as in our solar and wind farm cable layout problems), whose costs follow economies of scale [Rez22]. It is mentioned that this problem is also known under the names Single-Source Network Loading Problem and Local Access Network Design Problem. Models exist for both the splittable and unsplittable case, the latter meaning that the flow from one source must be routed along a single path to the sink. This category of network design problem can be combined with facility location problems so that decisions on which facilities to open and how to route demand to them have to be made. Again, we refer the interested reader to the survey in [Rez22]. We are not aware of any algorithmic approaches employing Minimum-Cost Flow Algorithms to tackle any of the problems mentioned in this paragraph.

2.4 Thesis Outline

Chapter 3 introduces our notation and fundamental concepts on which we build in the remainder of this dissertation.

Chapter 4 presents our research on the Wind Farm Cabling Problem (WCP), covering Contributions 1 to 6. WCP is formally stated as a Minimum-Cost Flow Problem in Section 4.1, with strong NP-hardness and the analogon to the Integer Flow Theorem proven in Section 4.2 to complete Contribution 1. Our NCC algorithm is explained in Section 4.4 where we also discuss and address the challenges imposed by the step cost function as mentioned in Contributions 2 and 3. The second part of Contribution 2, the experimental evaluation, is provided in Section 4.5. We compare our algorithm to an MILP formulation (presented in Section 4.3) solved by the commercial solver Gurobi and an approach using Simulated Annealing from the literature. Our NCC algorithm is embedded into an Iterated Local Search in Section 4.6 and subsequently evaluated in Section 4.7, which covers Contribution 5. The case study on the Hornsea One wind farm as mentioned in Contribution 6 can be found in Section 4.8 and the analysis of cable layouts under topological and electrical aspects as promised in Contribution 4 is presented in Section 4.9. Following a discussion of our findings in relation to past and possible future research in Section 4.10, we conclude with a brief summary and outlook in Section 4.11.

Chapter 5 presents our research on the Solar Farm Cable Layout Problem, covering Contributions 7 to 9. We introduce SoFaCLaP in Section 5.1 in accordance with Contribution 7 and provide complexity results in Section 5.2. The MILP formulation alluded to in Contribution 8 is presented in Section 5.3 and the experimental evaluation is given in Section 5.5. The benchmark instances we generate for the evaluation and the generation process itself are presented in Section 5.4.2, which fulfills Contribution 9. Again, we conclude with a discussion of our model in Section 5.6 and an outlook for future research in Section 5.7.

Chapter 6 provides a short summary of this dissertation.

3 Fundamentals

Let us continue by introducing our notation and basic concepts the reader may find helpful to understand the elaborations in the following chapters.

3.1 Graph Theory

For our purposes, a (directed) graph G is a tupel $(V(G), E(G))$ (or simply (V, E) if the context is unambiguous) consisting of a set V called *vertices* and a set of *edges* $E \subseteq V \times V \setminus \{(u, u) \colon u \in V\}$. A *subgraph* of a graph G is any graph (V', E') such that $V' \subseteq V(G)$ and $E' \subseteq E(G)$. Given an edge $e = (u, v)$ (or simply $e = uv$) we call u the *startvertex* and v the *endvertex* of e and say that u and v are *connected* by e. The edge $\bar{e} = (v, u)$ is called the *reverse edge* of $e = (u, v)$ if both exist. A vertex u is *incident* to an edge e if $u \in e$. The *out-/in-degree* of a vertex u is the number of edges for which u is the start-/endvertex, respectively. Two edges e_1, e_2 are *adjacent* if there is a vertex that is incident to both and two vertices are *adjacent* if there is an edge that is incident to both.

A *walk* is a sequence of edges $W = (e_1, \dots, e_k)$ with $k \in \mathbb{N}$ such that for each $i \in [k-1] := \{1, \dots, k-1\}$ it holds that the endvertex of e_i is the startvertex of e_{i+1}. We call k the *length* of W and say that W *starts* at the startvertex of e_1 and *ends* at the endvertex of e_k. A walk W in a graph G gives rise to a subgraph of G with edge set $E' = \{e_1, \dots, e_k\}$ and vertex set consisting of exactly the vertices incident to edges in E'. We call a walk *closed* if the endvertex of e_k is the startvertex of e_1. A *cycle* is a closed walk in which no two edges share a startvertex; it is usually denoted by C. A *path* $P = (e_1, \dots, e_k)$ is a walk in which no two edges share a startvertex and in which the endvertex of e_k and the startvertex of e_1 are distinct.

Given a directed graph G, the *line graph* $\mathcal{L}(G)$ is a representation of the adjacency of edges. It is defined by $V(\mathcal{L}(G)) = E(G)$ and an edge (e_1, e_2) exists in $\mathcal{L}(G)$ if and only if the endvertex of e_1 is the startvertex of e_2 in G.

3.2 Network Flows

Our definition of flows in graphs is a mixture of the definitions in [AMO93, Weg20] tailored to our needs. We will give a short overview of the changes from [AMO93], as we build on algorithms presented there.

Let G be a graph. We assume that the graph does not include anti-parallel edges, i. e., $(u, v) \in E$ implies $(v, u) \notin E$, and that it is weakly-connected, i. e., between any two vertices there is a sequence of consecutively adjacent edges.

Network flows are a handy notion to model how abstract or material things are moving or are being moved through a graph, like people walking in a pedestrians' zone, goods being transported between factories, data being sent via the internet or, in our case, electrical power being transmitted from generators to substations. In all of these cases, a certain thing (a person, a package of data, a unit of power) originates at a source (a restaurant, a computer, a generator) and is routed to a target (a bus-stop, another computer, a substation), possibly via other entities of the graph (a certain junction of streets, a third computer, an electrical bus).

We regard a subset of vertices $V^\uparrow$ as *source vertices* with a "production" function $p\colon V^\uparrow \to \mathbb{N}$ representing how many entities leave a vertex or, in our electrical setting, how much power is generated. For our purposes, the production of different source vertices need not be distinguishable (unlike people walking in the street), so we refer to the electrical power more generally as *(generated) output*. Another subset of vertices $V^\downarrow$ with $V^\downarrow \cap V^\uparrow = \emptyset$, called the *target vertices*, serve as destinations. They have a capacity function $\mathrm{cap}_V\colon V^\downarrow \to \mathbb{N}$ limiting the amount of output from the sources that can "exit" the graph at the target vertices. All other vertices neither generate additional output nor can take output out of the graph. In the literature, they are referred to as "*transshipment nodes*" [AMO93, p. 5].

How the generated output from the source vertices moves through the graph to the target vertices is modelled by a function $f\colon E \to \mathbb{R}$. By our assumption that any two adjacent vertices are connected by only one edge, the sign of $f(u, v)$ represents if there is movement from u to v or vice versa. If for an edge (u, v) we have that $f(u, v) > 0$, we say that $f(u, v)$ units of output go from u to v. Analogously, $-f(u, v)$ go from v to u if $f(u, v) < 0$. For notational ease, we may alias $f(v, u) = -f(u, v)$ for an edge (u, v) even though the left-hand side is not defined by the function f.

Given such a function f, we define the movement through a vertex by its *net flow*

$$f_{\mathrm{net}}(u) = \sum_{(w,u)\in E} f(w, u) - \sum_{(u,w)\in E} f(u, w). \tag{3.1}$$

To limit the movement through the graph, edges may have capacities for either direction. For each edge (u, v) we have capacities $\mathrm{cap}_E(u, v), \mathrm{cap}_E(v, u) \in \mathbb{N}_0$ so that not more than $\mathrm{cap}_E(u, v)$ may go from u to v and not more than $\mathrm{cap}_E(v, u)$ may go from v to u.

Now, we can define what a flow is. Given a graph as defined in this section with source and target vertices with their respective output and capacities, as well as capacities for the edges, we say that $f : E \to \mathbb{R}$ is a *flow* on G if all source vertices' productions are injected into the graph, i. e.,

$$f_{\text{net}}(u) = -p(u) \qquad \forall u \in V^{\uparrow}, \tag{3.2}$$

if the capacities of target vertices is not exceeded, i. e.,

$$f_{\text{net}}(u) \leq \text{cap}_V(u) \qquad \forall u \in V^{\downarrow}, \tag{3.3}$$

if transshipment nodes work as neither sources nor targets, i. e.,

$$f_{\text{net}}(u) = 0 \qquad \forall u \in V \setminus (V^{\uparrow} \cup V^{\downarrow}), \tag{3.4}$$

and if capacities on edges are respected, i. e.,

$$-\text{cap}_E(v, u) \leq f(u, v) \leq \text{cap}_E(u, v) \qquad \forall uv \in E. \tag{3.5}$$

We refer to Equations (3.2) to (3.4) as the *flow conservation constraints*.

Remark. Our definition of flows deviates from the classical definition in [AMO93] in order to facilitate our use of Negative Cycle Canceling in a very crucial point. Yet, it is equivalent for our purposes. The definition in [AMO93] assumes that the graph contains both directed edges (u, v) and (v, u) whenever u and v are adjacent, i. e., the graph is *bidirected*. Flow values on edges must be non-negative. To convert to our flow definition, the difference of the two flow values is computed. However, if the flow values on both (u, v) and (v, u) were positive, our algorithm would have problems with reducing this redundant flow Section 4.4.3. The other and less consequential difference is that target vertices in [AMO93] have a fixed demand while ours have a capacity. The remedy to convert our model into theirs is to add another vertex connected to all target vertices with proper demand and edge capacities. We allude to the equivalence in the construction in the proof of Theorem 2.

Next, we associate costs with flows. Let $\text{cost} : E \times \mathbb{R} \to \mathbb{R}_{\geq 0}$. We say that the cost to send x units of flow from u to v along the edge (u, v) is $\text{cost}((u, v), x)$, keeping in mind the convention on how to interpret negative flow values. With that, the cost of a flow f is defined as

$$\text{Cost}(f) = \sum_{e \in E} \text{cost}(e, f(e))$$

and the goal of the Minimum-Cost Flow Problem is to find a flow of minimal cost on a given graph (with given output and capacity functions).

In our setting, cost is the product of edge lengths and the step cost function arising from cable types. Classically, the cost function is $\text{cost}(e, f(e)) = |f(e)| \cdot c_e$ for some non-negative constant c_e. With the aforementioned model in [AMO93] using edges in both directions, this yields a linear cost function and many efficient algorithms exist. One such algorithm is Negative Cycle Canceling (NCC).

3.3 Negative Cycle Detection and Canceling

We describe the fundamentals of the NCC algorithm in its classical setting with linear cost flows according to [AMO93].

For the moment, we go back to the flow model on graphs with bidirected edges as outlined in the remark in Section 3.2. Given a bidirected graph G with a flow f on G, the *residual graph*[1] R_f with respect to f is defined by $V(R_f) = V(G)$ and $E(R_f) = \{(u,v), (v,u) \colon (u,v) \in E(G)\}$. The edge (u,v) in R_f has cost c_{uv} and a so-called *residual capacity* $r_{uv} = \mathrm{cap}_E(u,v) - f(u,v)$ with c_{uv} and $\mathrm{cap}_E(u,v)$ as defined in Section 3.2. For the reverse edge (v,u) we define $c_{vu} = -c_{uv}$ and $r_{vu} = f(u,v)$. Edges with zero residual capacity are removed. The residual capacities state by how much the flow can be increased ("augmented") in the direction of the respective edge so that the flow does not violate any edge capacities in the original graph.

Given a cycle $C = (e_1, \ldots, e_k)$ in some residual graph R_f with associated costs c_e, we say that C is *negative* if $\sum_{e \in C} c_e < 0$. These negative cycles are of particular importance for the optimality of a flow:

Theorem 1 ([AMO93, Thm. 9.1] Negative Cycle Optimality Conditions)**.** *A flow f is an optimal solution to the Minimum-Cost Flow Problem if and only if the corresponding residual graph R_f does not contain any negative cycles.*

This yields what is called the *cycle-canceling algorithm* [AMO93, p. 317]: Starting with some flow in the graph, repeatedly find a negative cycle C in the residual graph and augment the flow on that cycle by as much as the residual capacities of edges on the cycle permit, namely by $\min\{r_e \colon e \in C\}$. This augmentation step is called *canceling a negative cycle*. Terminate once no negative cycle exists in the residual graph corresponding to the current flow. By Theorem 1, the resulting flow is optimal.

Given bounds on the edge capacities and costs, say $\mathrm{cap}_E \leq U$ and $c_e \leq C$, this algorithm needs $\mathcal{O}(UC \cdot |E|)$ iterations. This bound can be reduced to a polynomial bound by carefully selecting which cycles to cancel. We refer the reader to the discussion in [AMO93, pp. 319, 342].

Negative cycles can be detected by means of the (Moore-)Bellman-Ford algorithm, attributed to [Moo59, Bel58, For56], shown in Algorithm 3.1. Its original purpose is to compute shortest paths in graphs. Our presentation follows [KV00, CLRS09]. Let G be a graph with a weight function $c \colon E \to \mathbb{R}$ and $s \in V$ be a designated starting vertex, from which shortest paths to all other vertices should be computed.[2]

The update of a distance label in Line 5 is also called *relaxation* of edge (u,v). The traversal of parent pointers in Line 8 reveals a walk containing a negative cycle.

Obviously, Algorithm 3.1 runs in time $\mathcal{O}(|V| \cdot |E|)$ which yields a running time of the cycle-canceling algorithm of $\mathcal{O}(UC \cdot |V| \cdot |E|^2)$. Straightforward observations yield speed-ups of the algorithm, albeit not in a worst-case analysis: Not all edges need to be considered in Line 3.

[1] In the reference, this graph is called the *residual network* [AMO93, p. 298].

[2] To be precise, only negative cycles reachable from s can be found but this is covered by our connectivity assumptions.

Algorithm 3.1: Bellman-Ford algorithm

Input: Graph G with edge weights c and starting vertex s
Output: Distance labels $\text{dist}(u)$ and parent pointers $\text{parent}(u)$ for all vertices

```
1 dist(s) = 0, dist(v) = ∞ for all u ∈ V \ {s}
2 forall i = 1, ..., |V| − 1 do
3     for each (u, v) ∈ E do
4         if dist(v) > dist(u) + c(uv) then
5             dist(v) = dist(u) + c(uv)
6             parent(v) = u
7 if dist(v) > dist(u) + c(uv) for some edge (u, v) then
8     G contains a negative cycle. Traverse parent pointers backwards from u.
```

More precisely, an edge (u, v) in Line 3 only needs to be considered if the label at u has been updated since the last consideration of (u, v) [Moo59]. Consequently, the algorithm can be aborted early if no changes have been made to distance labels within one iteration of the outer loop [Moo59]. We will make use of these observations in our NCC algorithm (Section 4.4).

Other speed-up techniques by which the number of iterations of the outer loop can be reduced by constant factors: At most $\lceil |V|/2 \rceil$ iterations can be achieved by fixing a linear order on the vertices and partitioning edges according to the order of the incident vertices [Yen70][CLRS09, Problem 24-1] and using a random order yields $\lceil |V|/3 \rceil$ iterations in expectation [BE12].

We refer the interested reader to a more general discussion on "label-correcting shortest path algorithms" in [AMO93, Sec. 5.3] and historical discussions in [AMO93, p. 156] and [Sch12]. A presentation and experimental evaluation of other detection algorithms for negative cycles can be found in [CG99].

3.4 Binomial Sign Test

This section is based on Appendix A of the arXiv version of joint work with Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf [Gri+19].

The algorithms presented in this work are built in a way that allows different *strategies* to be used at various key steps. One can think of it as trying to find the way out of a maze, one strategy would be to always turn left, another one to alternate between turning left and right. The goal of a statistical evaluation would be to find statistical evidence whether one strategy is better than the other. A corresponding algorithm that uses these strategies is to start walking until one encounters a junction. At the junction, turn as the strategy commands and continue walking to the next junction.

In our experimental evaluation, we compare different sets of strategies, to which we refer as *variants* of our algorithm to find cable layouts. One building block of our evaluation are statistical tests, in particular the (one-sided) *Binomial Sign Test for two dependent samples* [She11, p. 303].

We explain this test in a general setting here and specify how we apply the test in more detail below. Generally speaking, we compare k variants of an algorithm. We apply each variant to each instance. For every instance m, we denote the total cost of the resulting flow computed by variant i on instance m by $X_m^{(i)}$.

For any ordered pair of two variants (i, j) running on a fixed instance m, we calculate its solution difference $D = X_m^{(i)} - X_m^{(j)}$ and increment—depending on the sign of D—either n_i or n_j where, for example, n_i counts the instances in which i performed strictly better than j. If both variants were equally good, then $n_i \sim \text{Bin}(n_i + n_j, 0.5)$, i. e., n_i is binomially distributed on $n_i + n_j$ trials and probability 0.5.

We perform $k(k-1)$ tests, one for each ordered pair of variants, and always test the null hypothesis $\text{H}_0 : \theta = 0.5$ against the alternative hypothesis $\text{H}_1 : \theta > 0.5$ where θ is the probability in the underlying hypothesized distribution $n_i \sim \text{Bin}(n_i + n_j, \theta)$. The resulting p-values are Bonferroni-corrected by the number of tests. In this setting, we interpret rejecting H_0 as algorithm variant i performing better than algorithm variant j.

While this test does not give any indication how much better a variant is over the other, its undeniable advantage lies in the fact that its only prerequisites are of rather straightforward nature and not difficult to comply with. For more details, we refer the reader to the elaborations in [She11].

4 The Wind Farm Cabling Problem

The first renewable power plant with decentralized generators we consider in this work are (offshore) wind farms. In Section 2.1 we have given insights into the structure of wind farms, outlined the planning process, and reviewed the literature with a focus on cable layout optimization. In this chapter, we build on a Minimum-Cost Flow Problem for cable layout optimization from the literature. In this problem statement, the positions of turbines and substations are fixed, as well as the export cable(s) from the substations to an onshore grid point.

The definition of the cable layout optimization problem comes next (Section 4.1) followed by theoretical results on the problem complexity and on integral solutions in Section 4.2. We describe an MILP formulation as one solution approach in Section 4.3. In Section 4.4 we present our algorithm based on Negative Cycle Canceling and analyse it from a theoretical perspective. An experimental evaluation on synthetic benchmark instances follows in Section 4.5. With the lessons learned in the evaluation, we propose extending our NCC algorithm to an Iterated Local Search in Section 4.6 and evaluate all algorithms experimentally in Section 4.7 and by a case study (Section 4.8). An analysis of the cable layouts computed by our algorithms with respect to structural and electrical properties is provided in Section 4.9, followed by a thorough discussion of the model, our algorithms, and possible adaptations to other settings from the literature in Section 4.10. We conclude with a summary and an outlook in Section 4.11.

4.1 The Optimization Problem

In this section we define the Wind Farm Cabling Problem (WCP) as an optimization problem involving network flows and give theoretical insights into its structure. These elaborations are based on joint work with Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf [Gri+18, Gri+19]. The problem statement is in essence due to [LRWW17].

We consider the planning step in which positions of turbines and substations are fixed and the goal is to determine a suitable internal cable layout. Given this setting, 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

$$V = V_T \cup V_S, \qquad E \subseteq (V \times V) \setminus (V_S \times V_S)$$

such that $(u, v) \in E$ implies $(v, u) \notin E$. There are no edges between any two substations since we only consider the internal cabling and assume that the cabling from substations to the onshore grid point has been determined. The substations are the target vertices in our flow model (Section 3.2) and the turbines are the source vertices. We assume uniform generation across all turbines, i. e., $p \equiv 1$.

Substations have a capacity $\text{cap}_{\text{Sub}} \colon V_S \to \mathbb{N}$ representing the maximum amount of turbine generation they can handle. Edges have a (uniform) capacity $\text{cap}_E \in \mathbb{N}_0 \cup \{\infty\}$, which we set to the maximum cable capacity (Equation (4.6)), and a length given by $\text{len} \colon E \to \mathbb{R}_{\geq 0}$ which may, but not need to, represent the geographic distance between the respective endpoints of the edges.

On such a wind farm graph G, a *flow* according to the definition in Section 3.2 is a function $f \colon E \to \mathbb{R}$ that satisfies the so-called flow conservation constraints at both turbines and substations (Equations (3.2) to (3.4))

$$f_{\text{net}}(u) = -1 \qquad \forall u \in V_T, \tag{4.1}$$

$$f_{\text{net}}(v) \leq \text{cap}_{\text{Sub}}(v) \qquad \forall v \in V_S, \tag{4.2}$$

that does not exceed the capacities on edges, i. e.,

$$|f(e)| \leq \text{cap}_E(e) \qquad \forall e \in E, \tag{4.3}$$

and that allows no outflow from substations, i. e.,

$$f(u, v) \geq 0 \qquad \forall (u, v) \in E : v \in V_S, \tag{4.4}$$

$$f(v, u) \leq 0 \qquad \forall (v, u) \in E : v \in V_S, \tag{4.5}$$

where the *net flow* f_{net} at a vertex u is defined as in Equation (3.1). The last three inequalities can be obtained from Equation (3.5) by suitable adjustments of the edge capacity function. Equations (4.4) and (4.5) represent the idea that once power reaches a substation, it is transmitted from that substation via the export cable to the onshore grid point.

For an edge (u, v) we continue to understand $f(u, v) > 0$ (resp. < 0) as $f(u, v)$ units of flow going from u to v (resp. $-f(u, v)$ units going from v to u). Thus, a negative net flow at a vertex as seen in Equation (4.1) means that more flow leaves a vertex than enters it.[1]

[1] Note that in [Gri+18, Gri+19] a flow was defined as any function on the edges of a graph and a flow was defined as *feasible* if Equations (4.1), (4.2), (4.4) and (4.5) hold. Here, feasibility is included in the very definition of "flow". The edge capacity constraint (Equation (4.3)) was only modelled in the cost function: A feasible flow would have finite cost if and only if its absolute value is bounded by the maximum cable capacity. We believe that the definitions used here are more straightforward as unhelpful distinctions are dropped.

Along each edge we may place a single cable, whose type is chosen from a finite set of cable types. Each cable type κ is uniquely determined by its capacity $\text{cap}_K(\kappa) \in \mathbb{N}_0 \cup \{\infty\}$ and its cost per unit length $c_K(\kappa) \in \mathbb{N}_0$. We therefore identify each cable type κ with the pair $(\text{cap}_K(\kappa), c_K(\kappa))$ and define the set K of all allowed cable types represented by these pairs. For notational consistency we assume that K also contains the two special cable types $(0, 0)$ and (∞, ∞) called *trivial cable types*. The former represents the case that no cable is built along an edge and the latter the case that no cable has sufficient capacity. The *maximum cable capacity*

$$\sup\{\text{cap}_K(\kappa) \colon (\text{cap}_K(\kappa), c_K(\kappa)) \in K, c_K(\kappa) < \infty\} \tag{4.6}$$

is the highest capacity of any cable type with finite cost. Based on the cable types we define a cost function $c \colon \mathbb{R} \to \mathbb{N}_0 \cup \{\infty\}$ by

$$c(x) = \min\{c_K(\kappa) : (\text{cap}_K(\kappa), c_K(\kappa)) \in K, |x| \leq \text{cap}_K(\kappa)\} \qquad \forall x \in \mathbb{R}, \tag{4.7}$$

i. e., we choose the cheapest cable type that has sufficient capacity to transport $|x|$ units of flow. We refer to c as a *step cost function* due to Proposition 1. But first, we may impose further assumptions on the set of cable types without losing generality by artificially introducing an order on the cable types: Let $K = \{\kappa_1, \ldots, \kappa_{|K|}\}$ such that $\text{cap}_K(\kappa_i) \leq \text{cap}_K(\kappa_{i+1})$. We may assume that the capacities are indeed pairwise different and that $c_K(\kappa_i) < c_K(\kappa_{i+1})$. Otherwise there would be a cable type that never realizes the minimum in Equation (4.7) and that can therefore be removed from K.

Proposition 1. *Let c be defined as in Equation* (4.7)*. Then, the restriction of c to $\mathbb{R}_{\geq 0}$ is piecewise constant, non-decreasing, and left-continuous and it holds that $c(0) = 0$.*

Proof. Since K includes the trivial cable type $(0, 0)$ and since the per unit cost of cable types is non-negative, it holds that $c(0) = 0$. By definition, c is constant on the intervals $(\text{cap}_K(\kappa_i), \text{cap}_K(\kappa_{i+1})]$ for any $i \in [|K| - 1]$ and these intervals span $\mathbb{R}_{\geq 0}$. Let $x, y \in \mathbb{R}_{\geq 0}$ with $x < y$. For any cable type κ with $|y| \leq \text{cap}_K(\kappa)$ it also holds $|x| \leq \text{cap}_K(\kappa)$ and thus $c(x) \leq c(y)$. Hence, c is non-decreasing on $\mathbb{R}_{\geq 0}$. Let $(x_n)_{n \in \mathbb{N}}$ be a sequence of non-negative reals converging from the left to some $x \in \mathbb{R}_{\geq 0}$, i. e., $x_n \to x$ as $n \to \infty$ and $x_n \leq x$ for all $n \in \mathbb{N}$. Then, $|x_n| \leq |x|$ and $|x_n| \to |x|$ as $n \to \infty$. Since c is piecewise constant, we have that $c(x_n) = c(x)$ for sufficiently big n, implying left-continuity. □

Given such a step cost function obtained from cable types, we define the cost of a flow on a wind farm graph as

$$\text{Cost}(f) = \sum_{e \in E} c(f(e)) \cdot \text{len}(e). \tag{4.8}$$

With that, given a wind farm graph and a set of cable types, the Wind Farm Cabling Problem consists of finding a flow on the wind farm graph of minimal cost.

4.2 On the Complexity of WCP and Integral Solutions

In this section we establish complexity results on WCP and link them to complexity results for standard Minimum-Cost Flow Problems, in which the cost function c is linear. In particular, we show that WCP is strongly NP-hard (Theorem 3). We also provide a proof that the so-called Integer Flow Theorem (e. g. [AMO93, Thm. 9.10]) remains true in the setting of WCP. The latter result facilitates the engineering of heuristic solution methods.

Theorem 2. *It is possible to determine in polynomial time, if there is a flow on a given wind farm graph with a given set of cable types.*

Proof. We provide a reduction from the Maximum Flow Problem [AMO93, p. 168]: Note again that their flow model is different to ours as described in Section 3.2. Let $G' = (V(G'), E(G'))$ be a bidirected graph, i. e., $(u, v) \in E(G')$ implies $(v, u) \in E(G')$, with non-negative edge capacities $\mathrm{cap}_E(e) \in \mathbb{N}_0$, and let $x, y \in V(G')$ be two distinguished vertices, called the source and the sink. Maximize the *flow value* μ such that there is a flow f' with $f'_{\mathrm{net}}(x) = -f'_{\mathrm{net}}(y) = -\mu$ and $f'_{\mathrm{net}}(u) = 0$ for all other vertices, as well as $0 \leq f'(e) \leq \mathrm{cap}_E(e)$ for all $e \in E(G')$.[2] The Maximum Flow Problem can be solved in polynomial time [AMO93, Thm. 7.10].

We modify a given wind farm graph as follows: Add all reverse edges and set their capacity to the maximum cable capacity. For all edges of the form (u, v) with $u \in V_S$ set the capacity to zero. Add a source vertex x and insert all edges between x and any turbine u, with $\mathrm{cap}_E(x, u) = 1$ and $\mathrm{cap}_E(u, x) = 0$. Add a sink vertex y and insert all edges between y and any substation v, with $\mathrm{cap}_E(y, v) = 0$ and $\mathrm{cap}_E(v, y) = \mathrm{cap}_{\mathrm{Sub}}(v)$.

The maximum flow value in this network is at most $|V_T|$, since the edges of the form (x, u) induce a cut of capacity $|V_T|$ [AMO93, Property 6.1]. The maximum flow value is therefore exactly $|V_T|$ if and only if a flow exists in the wind farm graph:

Given a witness f' to the maximum flow value of $|V_T|$, we obtain a flow f on the wind farm graph by $f(u, v) = f'(u, v) - f'(v, u)$ for all $(u, v) \in E(G)$. Equations (4.1) and (4.2) follow from the capacities on edges incident to the source and the sink, respectively, Equation (4.3) holds as f' respects the maximum cable capacity, and the constraints forbidding outflow from substations (Equations (4.4) and (4.5)) hold because the respective edges have zero capacity in the modified graph.

Given a flow f on the wind farm graph, define $f'(u, v) = f(u, v)$ and $f'(v, u) = 0$ if $f(u, v) \geq 0$ and define $f'(v, u) = -f(u, v)$ and $f'(u, v) = 0$ otherwise. Furthermore, define $f'(x, u) = 1$ and $f'(u, x) = 0$ for all turbines u and $f'(v, y) = f_{\mathrm{net}}(v)$ and $f'(y, v)$ for all substations v. Then, f' yields a flow value of exactly $|V_T|$. □

However, if cost-minimization is taken into account, the problem becomes hard. This is a key difference to linear-cost flow problems, which can be solved in strongly polynomial time [AMO93, Section 10.5].

[2] The definition of the net flow follows our definition in a straightforward way. Note that the sign is inverted when comparing to the original definition [AMO93, Equation (6.1b)].

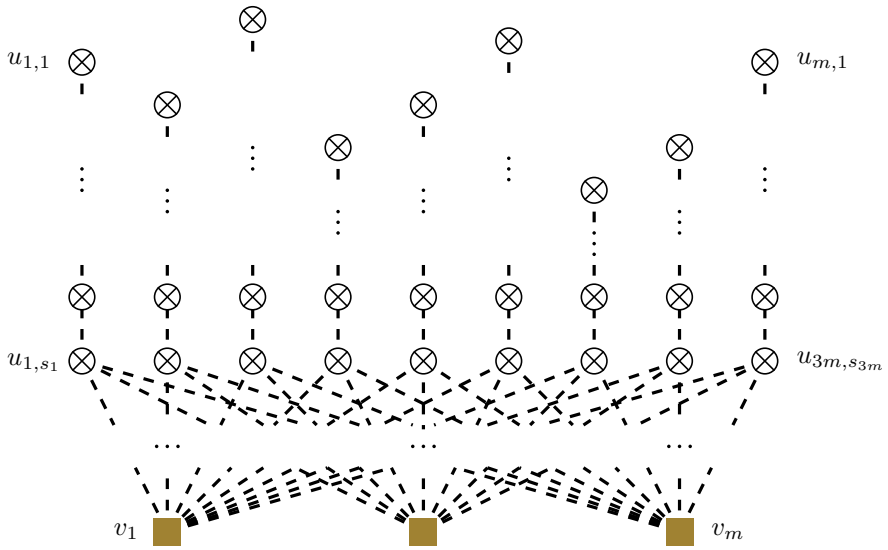


Figure 4.1: Visualization of a WCP instance constructed from a 3-Partition instance. Turbines are placed as $3m$ paths, each containing s_i turbines for the respective $i \in [3m]$. The last vertex of each path has edges to each substation v_j for $j \in [m]$. Ellipses indicate the inner vertices of the paths and the edges between turbines and substations.

Theorem 3. *WCP is strongly NP-hard, even if edge lengths and substation capacities are uniform, and there is only one non-trivial cable type, which has infinite capacity.*

In Theorem 3 we interpret WCP as a decision problem: Given an instance and a threshold value, does there exist a flow of cost below the threshold.

Proof. We reduce from the strongly NP-complete 3-Partition problem [GJ79, SP15]: Let $m, T \in \mathbb{N}$ and let $S \coloneqq \{s_1, \dots, s_{3m}\}$ be a multiset of natural numbers such that $T/4 < s < T/2$ for all $s \in S$ and $\sum_{s \in S} s = mT$. Can S be partitioned into triplets $S_1, \dots, S_m$ such that $\sum_{s \in S_i} s = T$ for all $i = 1, \dots, m$?

Given an instance of 3-Partition, we construct a wind farm graph as follows and as visualized in Figure 4.1: For every $i \in [3m]$ there is a path P_i: $u_{i,1}, \dots, u_{i,s_i}$ of turbines. Let there also be substations $v_1, \dots, v_m$, each of which has a capacity of T. In addition to the edges in the paths, the graph includes the edges $u_{i,s_i}v_j$ for all $i \in [3m], j \in [m]$. All edges have a length of 1 and the only non-trivial cable type is $(\mathrm{cap}_K(\kappa), \mathrm{c}_K(\kappa)) = (\infty, 1)$. This cable type implies that $c(x) = 1$ if $x \neq 0$ (and $c(0) = 0$). Thus, the cost of a flow on the wind farm graph equals the number of edges with non-zero flow.

We show that the instance of 3-Partition is a yes-instance if and only if there is a flow on the wind farm graph of cost at most mT.

Given a yes-instance with tripels S_j ($j \in [m]$), we define a flow as follows. On the paths P_i define $f(u_{i,j}u_{i,j+1}) = j$ for all $i \in [3m]$ and $j \in [s_i - 1]$. Furthermore, let $f(u_{i,s_i}v_j) = s_i$ if $s_i \in S_j$ and $f(u_{i,s_i}v_j) = 0$ for all $i \in [3m]$ and $j \in [m]$. It is straightforward to check that Equations (4.1) to (4.5) hold, hence f is a flow. There is exactly one edge with outgoing flow for each turbine, hence the flow has a cost of exactly $\sum_{i\in[3m]} s_i = mT$.

Let f be a flow on the wind farm graph of cost at most mT. By induction, it holds that $f(u_{i,j}u_{i,j+1}) = j$ for all $i \in [3m]$, $j \in [s_i - 1]$ using Equation (4.1). These are $\sum_{i\in[3m]} s_i - 1 = mT - 3m$ edges with non-zero flow. Thus, each vertex u_{i,s_1} has exactly one outgoing edge and the flow on this edge equals s_i. For every $j \in [m]$, let S_j be the set of the numbers such that the turbine u_{i,s_i} is connected to substation v_j. By Equation (4.2) it holds that $\sum_{s\in S_j} s \leq T$ for all $j \in [m]$ and equality holds since $\sum_{s\in S} s = mT$. Thus, the assignment of turbines to substations yields a 3-partition. □

Corollary 1. *WCP is strongly NP-hard, even if edge lengths are uniform, there is only one non-trivial cable type, and there is only one substation, which has infinite capacity.*

Proof. We only state the changes in the construction from Theorem 3: Each substation is replaced by a turbine and all of these turbines are connected to all turbines from the previous construction and to the substation. The only cable type has capacity $T + 1$ and cost 1. The instance of 3-Partition is a yes-instance if and only if there is a flow on the wind farm graph of cost at most $m(T + 1)$. In this case, the way how inflow arrives at the newly added turbines is a direct representation of the 3-Partition instance. □

In the classical case with linear costs, a central structural result is the Integer Flow Theorem [AMO93, Thm. 9.10]. It states that if a flow exists and if all edge capacities and vertex balances (similar to the right-hand sides of Equations (4.1) and (4.2)) are integers, then there is a cost-minimal integer flow, i. e., a cost-minimal flow that has integer values on all edges. The values in WCP that represent vertex balances, namely turbine generations and substation capacities, are natural numbers by definition. It is also needed that the cable capacities are defined as natural numbers to obtain the following analogon to the Integer Flow Theorem:

Theorem 4. *If a wind farm graph and a given set of cable types admit a flow, then there is a cost-minimal integral flow.*

Proof. Suppose f is a (possibly non-integral) flow of minimum costs. We define another flow network on the same graph by setting the capacity $\text{cap}_E(e)$ of every edge e to $\lceil |f(e)| \rceil$. Each turbine requires a net flow of -1. We model the substation capacities by adding a new vertex s and edges from all substations to s with capacities equal to the substation capacities. The net flow shall be 0 at all substations and $|V_T|$ at s. We further define zero costs for flows on all edges. By the integrality property of Minimum-Cost Flow Problems with linear cost functions (e. g. [AMO93, Thm. 9.10]) there is a feasible integral flow f' in this network. Due to the construction of the flow network, f' satisfies the constraints in Equations (4.1) to (4.5).

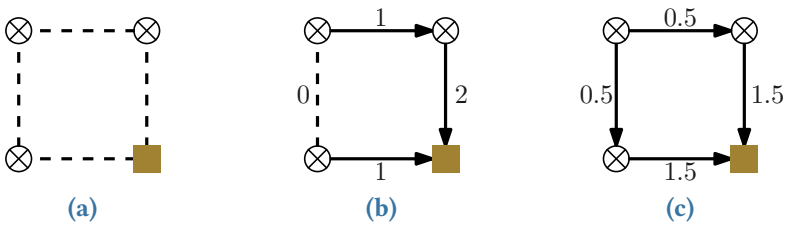


Figure 4.2: An example instance in which all integer flows are not cost-optimal. *(a)* The instance consists of three turbines and a substation with edges as depicted. All edge lengths are set to 1. The available non-trivial cable types are $(1.5, 1)$ and $(2, 3)$. *(b)* Since the edge capacity is 2 there is only one integer flow (up to symmetry). The labels show the values of the flow on the edges and the arrows resemble the direction of the flow. This flow has a total cost of 5. *(c)* This non-integer flow has a total cost of 4.

Since the cost function c is non-decreasing, it holds for all $e \in E$ that $c(x) \leq c(|f(e)|)$ for all $x \in [0, |f(e)|]$. By the other properties of c stated in Proposition 1, we also have $c(x) = c(|f(e)|)$ for all $x \in [|f(e)|, \text{cap}_E(e)]$. It holds in particular that $c(|f'(e)|) \leq c(|f(e)|)$. Thus, $\text{Cost}(f') \leq \text{Cost}(f)$ and f' is optimal in the original network. □

Figure 4.2 gives an example to show that Theorem 4 would not remain true if there were non-natural cable capacities even though the edge capacity, defined by the biggest cable type of finite cost, is integer. This justifies the model assumption that cable types must have integer capacities.

4.3 MILP Formulation

WCP can be formulated as a Mixed-Integer Linear Program (MILP) that can be tackled using commercial or non-commercial solvers. We use this MILP formulation as a competitor to which we can compare our algorithmic solutions to WCP such as our heuristic based on Negative Cycle Canceling (Section 4.4). While standard Minimum-Cost Flow Problems can be modelled using only continuous variables, the incorporation of cable types calls for the use of binary variables. We state the MILP formulation next as given in joint work with Dorothea Wagner and Matthias Wolf [GWW20] and give a formal proof of its correctness afterwards.

Let G be a wind farm graph and K a set of cable types as defined in Section 4.1. Each cable type $\kappa \in \text{K}$ has a capacity $\text{cap}_\text{K}(\kappa)$ on the amount of turbine generation that can be transmitted through it, as well as a cost per unit length $\text{c}_\text{K}(\kappa)$ for laying a cable of type κ. Here, we omit the trivial cable types $(0, 0)$ and (∞, ∞) to decrease the size of the MILP. We use binary variables $x \in \{0, 1\}^{E \times \text{K}}$ where E are the edges of the underlying wind farm graph and interpret $x(e, \kappa) = 1$ as a cable of type κ being installed on edge e. We further use variables $f \in \mathbb{R}^E$ that (by abuse of notation) represent the flow in the wind farm graph.[3]

[3] With our Integer Flow Theorem (Theorem 4), we could have enforced in the MILP that flow variables be integers as well. However, by rule of thumb, it is advisable to prefer real over integer variables in MILPs.

The MILP formulation for WCP we used in our experiments is as follows:

$$\min \sum_{e \in E} \sum_{\kappa \in \mathrm{K}} \mathrm{c}_{\mathrm{K}}(\kappa) \cdot x(e,\kappa) \cdot \mathrm{len}(e) \tag{4.9}$$

$$\text{s. t. } f_{\mathrm{net}}(u) = -1 \qquad \forall u \in V_T, \tag{4.10}$$

$$f_{\mathrm{net}}(v) \leq \mathrm{cap}_{\mathrm{Sub}}(v) \qquad \forall v \in V_S, \tag{4.11}$$

$$f(e) \leq \sum_{\kappa \in \mathrm{K}} x(e,\kappa) \cdot \mathrm{cap}_{\mathrm{K}}(\kappa) \qquad \forall e \in E, \tag{4.12}$$

$$-f(e) \leq \sum_{\kappa \in \mathrm{K}} x(e,\kappa) \cdot \mathrm{cap}_{\mathrm{K}}(\kappa) \qquad \forall e \in E, \tag{4.13}$$

$$\sum_{\kappa \in \mathrm{K}} x(e,\kappa) \leq 1 \qquad \forall e \in E, \tag{4.14}$$

$$f(u,v) \leq 0 \qquad \forall (u,v) \in E \colon u \in V_S, \tag{4.15}$$

$$f(u,v) \geq 0 \qquad \forall (u,v) \in E \colon v \in V_S, \tag{4.16}$$

where f_{net} denotes the net flow defined in Equation (3.1) using the flow variables f of the MILP. Equations (4.10) and (4.11) are the same as the constraints given in Equations (4.1) and (4.2). Equations (4.12) and (4.13) are equivalent to

$$|f(e)| \leq \sum_{\kappa \in \mathrm{K}} x(e,\kappa) \cdot \mathrm{cap}_{\mathrm{K}}(\kappa)$$

for all $e \in E$ and ensure that there is sufficient cable capacity installed on every edge for the respective flow, while there is only one cable type on that edge due to Equation (4.14). Equations (4.15) and (4.16) correspond to Equations (4.4) and (4.5) and ensure that no flow leaves any substation.

Lemma 1. *The following assertions on the connection between WCP and the MILP formulation hold:*

- *For every flow in the wind farm graph there is a feasible solution to the MILP with equal cost which extends the flow.*
- *Every feasible solution to the MILP gives rise to a flow in the wind farm and the value of the MILP solution is at least the cost of the flow.*
- *The optimal values of WCP and the corresponding MILP formulation are equal.*

Proof. Starting with a flow in the wind farm graph, we observe again that Equations (4.10), (4.11), (4.15) and (4.16) trivially hold. For every edge e with non-zero flow the edge capacity constraint Equation (4.3) in conjunction with the maximum cable capacity Equation (4.6) implies that there is a cable type of sufficient capacity to support the flow. We set $x(e,\kappa) = 1$ if and only if κ realizes the minimum in the cost function Equation (4.7), i. e., κ is the cheapest cable type

with sufficient capacity. For edges with zero flow, we set $x(e, \kappa) = 0$ for all cable types κ. Thus, at most one cable type is used, so Equation (4.14) holds. Consequently, the right-hand sides of Equations (4.12) and (4.13) collapse to 0 for edges with zero flow and, elsewise, to $\text{cap}_K(\kappa)$ for the cable type realizing Equation (4.7). As for the costs, $\sum_{\kappa \in K} c_K(\kappa) \cdot x(e, \kappa) \cdot \text{len}(e)$ collapses analogously and equals $c(f(e)) \cdot \text{len}(e)$ from Equation (4.8).

Starting with a feasible solution to the MILP, we see that constraints in the definition of a flow (Equations (4.1) to (4.5)) follow directly from the constraints of the MILP. Thus, the flow variables define a flow. On each edge, the chosen cable type from the binary variables (which may be the $(0, 0)$ cable type if all decision variables are zero) is one of the cables types from which the minimum over all unit costs is taken in Equation (4.7). Thus, the MILP may overestimate the costs of the flow on each edge and, consequently, the overall costs as well.

The third assertion follows immediately from the previous ones. In an optimal solution, the cable types on each edge will be chosen according to Equation (4.7). □

A fundamental notion to determine the progress of optimization in Mixed-Integer Linear Programming are *MI(L)P gaps* (or *relative gaps*). In the case of minimization problems, a solver (i. e., a program that solves MILPs) has two goals: Find a feasible solution and determine that no better solutions exist. On the one hand, any feasible solution a solver finds gives an upper bound (ub) on the optimal value. On the other hand, a solver tries to prove lower bounds on the optimal value (lb). Those bounds can be combined to the MIP gap $^{\text{ub}-\text{lb}}/_{\text{ub}}$.

By definition, MIP gaps take values in the unit interval and give information on how far the solution value might be off from the (unknown) optimal value. A MIP gap of 0 (or in practice a gap below a certain threshold close to zero) implies that the best solution that the solver has found (also called *incumbent* solution) is indeed optimal. A positive gap, however can mean two things: that the incumbent solution is not optimal or that it is optimal but not yet proven to be.

In our evaluations we will also adapt the MIP gap by using the lower bound from a solver and the upper bound as the best solution value as computed by another algorithm.

4.4 Negative Cycle Canceling for WCP

With the presentation of the MILP in the previous section, we have seen one competitor of our heuristic based on Negative Cycle Canceling. We proceed to present our algorithm. The NCC algorithm for linear cost flows is described in Section 3.3 including the residual graph and the Bellman-Ford algorithm to detect negative cycles. In Section 4.4.1 we define our adaptation of the residual graph. Section 4.4.2 states our algorithm on a high level and explains the difficulties arising from the step cost function. More details on the changes to the Bellman-Ford algorithm are provided in Section 4.4.3 and how it is employed in our NCC algorithm is explained in Section 4.4.4. The elaborations in this section are based on joint work with Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf [Gri+18, Gri+19].

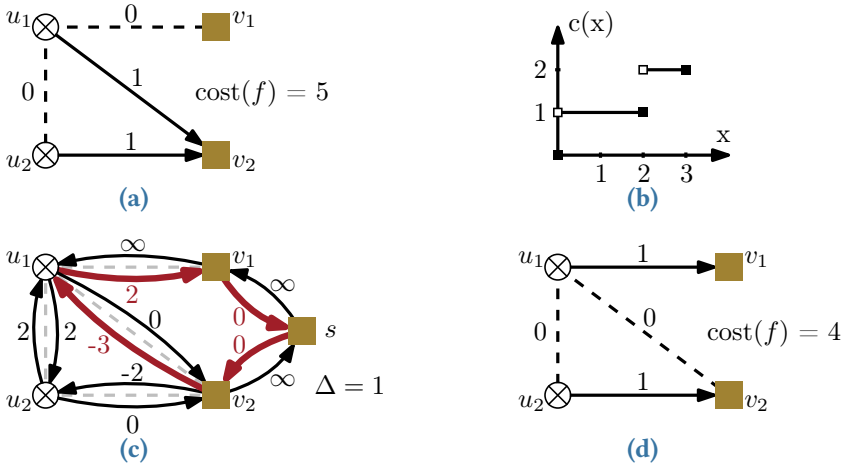


Figure 4.3: Residual Costs and Canceling a Negative Cycle. *(a)* A wind farm graph G with two turbines u_1, u_2 and two substations v_1, v_2. Edge u_1v_2 has a length of 3, all other edges have a length of 2. Substation capacities are 2. The values on the edges show a flow, which has a cost of 5 using the cable types depicted in *(b)*. *(c)* The residual graph R of G with residual costs obtained from the flow in (a) and $\Delta = 1$. Note how infinite residual costs are used to ensure that Equations (4.1), (4.2), (4.4) and (4.5) hold. There is no visualization for the edge capacity constraint Equation (4.3). The bold red edges show a negative cycle. *(d)* The flow obtained from canceling the negative cycle. The cost of the flow changed by exactly the value of the cycle.

4.4.1 The Residual Graph

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 generation of one turbine to be reassigned to another substation. Note that in line with the classical definition in Section 3.3 we have replaced each edge with two directed edges in opposite directions.

For $\Delta \in \mathbb{N}$ and a given flow f in G, which by definition has finite cost, 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 4.3 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 $\gamma : E(R) \to \mathbb{R}$ 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 start at 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 edge capacity of e is exceeded. For $u \in V_S$ and $v \in V_T$ that are adjacent in G, 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 $f(u,s) + \Delta \leq \text{cap}_{\text{Sub}}(u)$ and $\gamma(u,s) = \infty$ otherwise. On edges leaving the super substation, we set $\gamma(s,u) = 0$ if $f(u,s) \geq \Delta$ and $\gamma(s,u) = \infty$ otherwise to prevent flow from leaving the substation.

In the classical setting, the cost changes are only computed for $\Delta = 1$ and the residual capacities determine by how much the current flow is augmented. We address the problems with a straightforward adoption of this approach in Section 4.4.2.

On the residual graph with associated residual costs obtained from a flow f, we search for negative cycles to cancel. For the moment, let us assume that a cycle does not include an edge and its reverse edge. Given a negative cycle C with edges $E(C)$, *canceling* C means constructing a new flow f' from f by sending Δ units of flow along the edges of C. More formally, we define f' by

$$f'(u,v) = \begin{cases} f(u,v) + \Delta, & \text{if } (u,v) \in E(C), \\ f(u,v) - \Delta, & \text{if } (v,u) \in E(C), \\ f(u,v), & \text{if } (u,v), (v,u) \notin E(C). \end{cases}$$

for any $(u,v) \in E(G)$. By definition of the residual costs, it holds that

$$\text{Cost}(f') = \text{Cost}(f) + \sum_{e \in E(C)} \gamma(e). \tag{4.17}$$

The reason for our assumption that we only consider cycles without both an edge and its reverse can be seen in Figure 4.3 (c): The cycle $u_2 v_2 u_2$ has total residual costs of -2, but sending one unit of flow from u_2 to v_2 and one unit of flow from v_2 to u_2 does not change the flow, so the cost should not change either. In this case, one of the residual costs does not accurately represent the change in costs.

4.4.2 NCC in a Nutshell and Challenges on Wind Farms

In the language of Section 4.4.1, the cycle-canceling algorithm in the original setting of Minimum-Cost Flow Problems with linear costs (cf. Section 3.3) works as follows: Given a flow, the residual costs are computed for $\Delta = 1$ and if a negative cycle exists in the residual graph, it is found for example by means of the Bellman-Ford algorithm. Then, this cycle is canceled not only with $\Delta = 1$, but with an amount equal to the least residual capacity over any edge of the cycle, i. e., the smallest amount of additional units of flow, such that the flow hits zero or the edge capacity on any edge of the cycle. By linearity, the change in cost equals the value of the negative cycle multiplied by the least residual capacity. Iterating this approach exhaustively yields an optimal solution, as we have stated before:

Theorem 1 ([AMO93, Thm. 9.1] Negative Cycle Optimality Conditions). *A flow f is an optimal solution to the Minimum-Cost Flow Problem if and only if the corresponding residual graph R_f does not contain any negative cycles.*

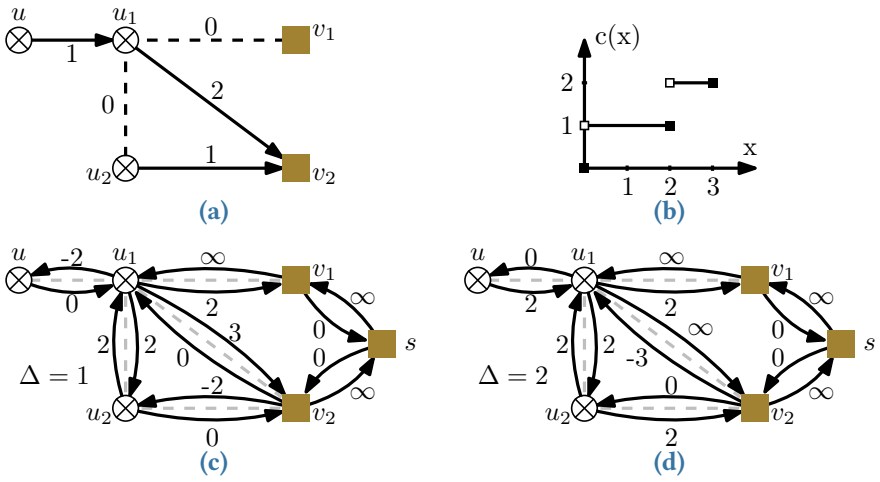


Figure 4.4: Example of a wind farm for which it is necessary to consider more than one value for Δ. *(a)* shows the adapted wind farm graph from Figure 4.3: Turbine u can only be connected to u_1. We assume that the production from u is routed via u_1 directly to v_2. The substation capacities are increased to 3. All edges have a length of 2 for all edges except (u_1, v_2) which has length 3. *(b)* The cable types remain unchanged. *(c)* The residual graph for the flow from (a) for $\Delta = 1$. *(d)* The residual graph for the flow from (a) for $\Delta = 2$. The cycle (u_1, v_1, s, v_2, u_1) is not negative for $\Delta = 1$ but for $\Delta = 2$.

For wind farms, Negative Cycle Canceling is less straightforward, since key properties used for the original algorithm do not hold: Canceling the same negative cycle up to the residual capacity is not consistent with the residual costs, higher values for Δ may be necessary, and the optimality condition does not hold any more. We explain all of these three problems in more detail in the following.

Assume a negative cycle (that does not contain an edge and its reverse) has been found for $\Delta = 1$. Then, in general, Equation (4.17) cannot be applied multiple times if more than one unit of flow is sent along the cycle, because the residual cost may also change after the first (and any subsequent) cancelation. Thus, it is sensible to recompute the residual costs after a cancelation.

Furthermore, only considering $\Delta = 1$ may not suffice: Let us modify the instance shown in Figure 4.3 by adding a turbine u that is only connected to u_1 and by increasing the substation capacity to 3 as shown in Figure 4.4 (a) with the same cable types as before. Assume that the generation of u is sent along (u, u_1, v_2). Then, the residual costs on the cycle (u_1, v_1, s, v_2, u_1) sum up to 2 for $\Delta = 1$ (the cycle should not be canceled, Figure 4.4 (c)) and to -1 for $\Delta = 2$ (the cycle can sensibly be canceled, Figure 4.4 (d)). Thus, not only one value of Δ should be considered for the computation of residual costs. Obviously, this observation can be extended to other values for Δ by adding a path of turbines and adjusting the cable types and capacities.

While the maximum value for Δ can be bounded by twice the maximum cable capacity, considering all possible values for Δ does not yield an optimal solution.

In fact, Theorem 1 does not hold in our setting: On the one hand, there are optimal solutions with negative cycles. While these cycles, which we call *short* cycles, necessarily consist of only two edges, namely an edge and its reverse, a standard Bellman-Ford algorithm would still find them. An example for an optimal solution with negative cycles can be seen in Figure 4.3 (d): In the residual graph for $\Delta = 1$, the cycle (u_2, v_2, u_2) is short and its residual costs sum up to -2. This negative cycle is visualized in Figure 4.3 (c). Note that the residual costs on this cycle have not changed, since the flow on the edge (u_2, v_2) has not changed.

On the other hand, the absence of *long* negative cycles (cycles with at least three edges, in particular without an edge and its reverse) for all possible values of Δ does not imply optimality. Consider Figure 4.5: The flow in (e) has a cost of 10, the flow in (g) costs 9. Thus, the flow in (e) is suboptimal. Yet, the residual graph for the flow in (e) (shown in (f) for $\Delta = 1$; for other Δ all residual costs are positive) does not have a negative cycles for any value of Δ. In particular due to the lack of an optimality criterion, we propose a heuristic approach to WCP.

This heuristic, which by abuse of name we also call the (or our) Negative Cycle Canceling (NCC) algorithm, is shown in Algorithm 4.1. It is designed to overcome the challenges outlined above. In a nutshell, it works as follows: Starting with an initial feasible flow and some value of Δ (Line 1), it computes the residual costs (Line 3), and runs the shortest path computations in our adaptation of the Bellman-Ford algorithm (Line 4, corresponding to Algorithm 3.1 Lines 2 to 6). The crucial difference is that the extraction step in Line 7 does not yield a desired long negative cycles but a set of cycles which may contain what we are looking for. The details of this step are explained in Section 4.4.3.

If the algorithm finds a long negative cycle in this set of cycles, it cancels the cycle, i. e., it changes the flow by adding Δ units of flow on all (residual) edges of the cycle (Lines 10 and 11). Then this procedure is repeated with the new flow and some value of Δ which may but not need to differ from the previous one. If no long negative cycle is found, a new value of Δ is chosen and new residual costs are computed (Line 14). This loop is repeated until all sensible values of Δ have been considered for a single flow (Line 2), which is then returned by the algorithm. This flow is of integer value, since the initial flow will be 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 Theorem 4, even though our algorithm does not necessarily find an optimal solution to WCP. In the following sections, we provide further details on different steps in Algorithm 4.1: Section 4.4.3 explains our adaptation of the Bellman-Ford algorithm and Section 4.4.4 details how this adaptation is incorporated into the NCC algorithm. How an initial flow is computed and how different values for Δ are chosen is explained in Sections 4.4.5 and 4.4.6.

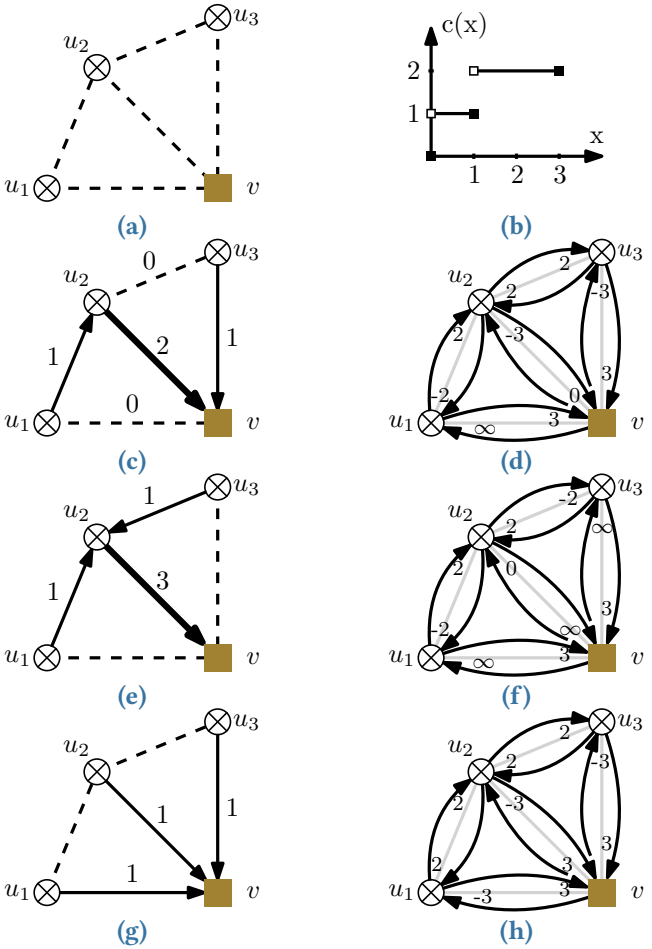


Figure 4.5: Examples of flows and corresponding residual graphs to underline the absence of the optimality condition from Theorem 1. *(a)* shows a wind farm graph. Edges between turbines are of length 2, edges between the substation v and any turbine are of length 3. *(b)* depicts a cost function induced by two cable types. *(c)* displays a flow. Dashed lines do not carry any flow, on other edges the label states the flow value and the arrow the (effective) direction of the flow. The thickness of solid lines represent 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: (v, u_2, v), (v, u_2, u_1, v), and (v, u_3, u_2, v). *(e)* shows the flow obtained by sending one unit of flow along (v, u_3, u_2, v) in (c). *(f)* is the residual graph for (e) and $\Delta = 1$. *(g)* depicts the flow obtained by sending one unit of flow along (v, u_2, u_1, v) in (c). *(h)* displays the residual graph for (g) and $\Delta = 1$.

Algorithm 4.1: Negative Cycle Canceling

Input: Wind Farm Graph G, cable types K
Result: A flow f in G

```
1  f := InitializeFlow(G, len), Δ := InitialDelta
2  while Δ ≠ null do
3      (R, γ) := ComputeResidualGraph(G, c_K, f, Δ)
4      RunBellmanFord(R, γ)
5      found := false
6      for each e ∈ E(R) do
7          W := FindNegativeClosedWalk(R, e)
8          C := DecomposeWalkIntoCycles(W)
9          for each 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
```

4.4.3 Detecting Long Negative Cycles

Preliminary experiments have shown that the standard Bellman-Ford algorithm applied to wind farm graphs tends to report short negative cycles even if long negative cycles exist. The reason allegedly 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. An example can be seen in Figure 4.5 (d): The negative residual costs on the edge vu_2 can influence the label at v—roughly speaking—every two iterations with a total change of -3 if edge u_2v is relaxed or every three iterations with a total change of -2 if the edges u_2u_1 and u_1v are relaxed. It is evident that traversing the parent pointers will yield the short negative cycle vu_2v.

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 line graph of R which we obtain from the line graph $\mathcal{L}(R)$ by removing all edges representing "U-turns", i. e., edges of the form $(e, \bar{e})$ for $e \in E(R)$. We define the residual 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 $\text{dist}(e)$ initialized as $\gamma(e)$. Note that we change the initialization compared to Algorithm 3.1: There, labels represent the distance from a certain vertex s. Here, we use an implicit starting vertex connected to all other vertices. Thus, throughout the Bellman-Ford algorithm, $\text{dist}(e)$ represents the length of some walk in L starting at any vertex of L (namely the first vertex on the walk after the implicit vertex) and ending at e. By construction of L, the label $\text{dist}(e)$ also stands for some walk in R which ends at the endvertex

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 than one cycle each of which may be negative. Therefore, we decompose the closed walk W into cycles, which, in turn, can be canceled one after another. For more details, refer to Section 4.4.4.

As $|V(L)| = |E(R)|$, running the Bellman-Ford algorithm on the line graph without U-turns has the downside that more distance labels have to be stored and updated. When naïvely implemented, this increases the worst-case running time to $\mathcal{O}(|V(L)| \cdot |E(L)|)$.

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 same asymptotic running time as the original Bellman-Ford algorithm on R. To this goal, we use the special structure of L to analyze what the steps of the Bellman-Ford algorithm on L mean for R. When running the Bellman-Ford algorithm on L, there is one label per 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 1. *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.*

We call these labels *relevant*. Consequently, throughout our modified version of the Bellman-Ford algorithm, we maintain two distance labels $\text{dist}_1(v)$ and $\text{dist}_2(v)$, and two parent pointers $\text{parent}_1(v)$ and $\text{parent}_2(v)$ for every $v \in V(R)$, respectively. As above, $\text{dist}_i(v)$ with $i = 1, 2$ stand 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 $\text{dist}_1(v) \leq \text{dist}_2(v)$ for every $v \in V(R)$. In every iteration of the Bellman-Ford algorithm, each edge of R is considered for relaxation: For an edge $e = (u, v)$ take $\text{dist}(u) = \text{dist}_1(u)$ if $\text{parent}_1(u) \neq v$ and $\text{dist}(u) = \text{dist}_2(u)$ otherwise. Then, check if $\text{dist}(u) + \gamma(e)$ yields a new relevant label at v. 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. This is in accordance with the standard Bellman-Ford algorithm, in which labels are only updated, if the new value is strictly smaller than the incumbent. For each edge, checking if it yields a new relevant label at its endvertex can be done in constant time. With Observation 1 we show reduced bounds for the number of iterations and the overall running time compared to a straightforward implementation on L.

Theorem 5. *There is a negative cycle in L if and only if after $2 \cdot |V(R)|$ iterations there is an edge that allows reducing a label.*

Proof. Let there be a negative cycle in L. We show a stronger statement: In every iteration, there is a label that is updated by the algorithm. To this end, we observe that a standard Bellman-Ford algorithm on L updates at least one label per iteration, since there is a negative cycle. We claim that in every iteration, one of updated labels is *relevant*. Consider all labels at $V(L)$ at the end of some iteration $k+1$. By traversing parent pointers, we find a label on some $e = (u, v) \in V(L)$ that has been updated in iteration $k+1$ and this update has used a label on some $e_1 \in V(L)$ from iteration k. The label on e_1 has been updated in iteration k, since the label on e is from iteration $k+1$. Furthermore, the label at e_1 is the smallest among all vertices in L of the form (w, u) with $w \neq v$, otherwise it would not have resulted in the label on e in iteration $k+1$. Thus, the label at e_1 is relevant at vertex u at the end of iteration k. By Observation 1 this label on e_1 is correctly updated by the algorithm, which completes this part of the proof.

Let $n = |V(R)|$ and suppose e_{2n+1} is an edge that allows reducing a label after $2n$ iterations. We iteratively construct a walk backwards starting from e_{2n+1} by repeatedly applying the following procedure. At an edge $e_i = (v, w)$ we define e_{i-1} as the incoming edge of v other than (w, v) with the smallest label. If there are several possibilities, we pick the edge with the oldest label among them—in particular, if $\text{dist}_1(v) = \text{dist}_2(v)$, use dist_1. The label at e_{i-1} is relevant by definition. We stop when an edge would be repeated. At this point, the walk contains a closed subwalk $W = (e_k, \dots, e_l)$ for suitable $k, l \in \mathbb{Z}$ with $k < l \leq 2n+1$. By Observation 1 there are at most $2n$ edges with relevant labels. Hence $k \geq 1$.

Since the label at e_{2n+1} can be updated after $2n$ iterations, the label at e_{2n} must have been updated in iteration $2n$. Repeating this argument inductively shows that for $i \geq k$ the label at e_i was updated in or after iteration i. Therefore, all labels of edges in W were updated after the initialization. By the way the labels are computed, we therefore have

$$\text{dist}(e_{i-1}) + \gamma(e_i) \leq \text{dist}(e_i)$$

for all edges e_i on W where we alias $e_{k-1} = e_l$.

If one of these inequalities is strict, i. e., $\text{dist}(e_{j-1}) + \gamma(e_j) < \text{dist}(e_j)$ for some $j \in \{k, \dots, l\}$, then summing over the inequalities for all edges in W will give

$$\sum_{e \in W} \big(\text{dist}(e) + \gamma(e)\big) < \sum_{e \in W} \text{dist}(e),$$

which can be simplified to

$$\sum_{e \in W} \gamma(e) < 0.$$

Hence, the total costs of W will be negative, which will complete the proof.

It remains to show that there is indeed some edge $e_j = (v, w)$ for which the inequality is strict. To this aim let e_j be the edge with the oldest label among edges in W. The label $\text{dist}(e_j)$ was computed from the label $\text{dist}'(e)$ of an edge $e = (u, v)$ with $u \neq w$, which may or may not be e_{j-1}. That means

$$\text{dist}'(e) + \gamma(e_j) = \text{dist}(e_j) \tag{4.18}$$

where dist denotes the labels after the algorithm finishes and dist′ denotes the labels when $\text{dist}(e_j)$ is computed. Note that the label at e may have been updated afterwards, i. e., $\text{dist}(e) \leq \text{dist}'(e)$.

For the sake of contradiction assume $\text{dist}(e_{j-1}) \geq \text{dist}'(e)$. Then, $\text{dist}'(e) \geq \text{dist}(e) \geq \text{dist}(e_{j-1}) \geq \text{dist}'(e)$ where the first inequality holds since labels at the same edge do not increase during the algorithm and the second inequality follows from e being an incoming edge of v. Hence, all these labels are equal. The first equality implies that the label on e was not updated after the point in time when $\text{dist}(e_j)$ was computed and that $\text{dist}(e)$ is older than $\text{dist}(e_j)$. Using the second equality, we distinguish two cases: If $e = e_{j-1}$, then $\text{dist}(e_{j-1})$ is older than $\text{dist}(e_j)$, which contradicts the choice of e_j. If $e \neq e_{j-1}$, then e should have been included in W instead of e_{j-1}. Thus, the assumption of $\text{dist}(e_{j-1}) \geq \text{dist}'(e)$ is wrong and it holds that $\text{dist}(e_{j-1}) < \text{dist}'(e)$. Combining this inequality and Equation (4.18) completes the proof. □

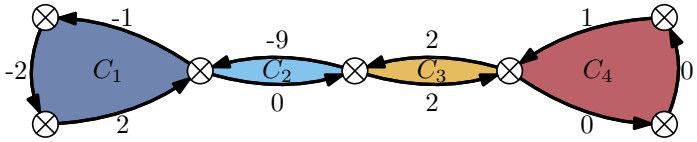
Both implications can be combined to efficiently extract negative cycles, if they exist, by the procedure in the second half of the proof. This yields the following corollary stating that our adapted version of the Bellman-Ford algorithm has the same asymptotic running time as the original.

Corollary 2. *A negative cycle in L can be computed in $\mathcal{O}(|V(R)| \cdot |E(R)|)$ time if one exists.*

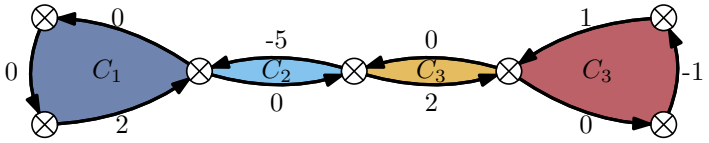
Note that Corollary 2 refers to cycles in L, not in R. In the upcoming Section 4.4.4 we describe how these cycles in L translate to cycles in R.

4.4.4 NCC Algorithm in Detail

The previously described Bellman-Ford algorithm on L is encapsulated in Algorithm 4.1, which we describe in more detail here. We first compute some initial flow (Line 1) using one of eight *initialization strategies* presented in Section 4.4.5. 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). After termination of the Bellman-Ford algorithm, we consider one edge after another and check in Line 7 if it can be relaxed (again). In that case, we extract a closed walk W in R with negative costs by traversing parent pointers backwords from that edge. However, canceling W as a whole may not improve the costs of the flow as W may still contain an edge and its reverse. By construction, every vertex in W has equal in- and out-degree with values of 1 or 2 due to traversing only parent pointers. We can therefore decompose W into a set of simple cycles $\mathcal{C}$ in Line 8 and cancel each cycle independently if it is long and has negative costs (Lines 9 to 12, cf. Figure 4.6 (a)). 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 (cf. Figure 4.6 (b)). In this case we search for another negative cycle in L (Line 13). If no negative cycle in the current graph R is canceled, a new value for Δ is determined according to the *delta strategy* (cf. Section 4.4.6) in Line 14 and new residual costs γ are computed. Line 14 also checks if every possible value for Δ has been used without improving the solution after the last update of f. If so, f is returned.



(a) Only C_1 is canceled since it is both long and negative.



(b) No cycle is canceled.

Figure 4.6: Two examples of a negative closed walk and the decomposition into cycles.

We apply the well-known speed-up techniques attributed to Edward Moore as outlined in Section 3.3: 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 startvertices, 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.

By Theorem 4, there is a cost-minimal integer flow if any flow exists in the wind farm graph. Thus, we design our algorithm to maintain an integer flow at all times. We define our initialization strategies in a way that they will only output integer flows. Then, in the course of our cycle-canceling procedure, we only consider natural values for Δ as returned by our delta strategies. It is the focus of the next to sections to explain how our strategies are defined.

4.4.5 Initialization Strategies

Before we can start searching for and canceling negative cycles, we need some initial flow. To obtain such a flow, we consider eight strategies, which all roughly work as follows. We pick a turbine u whose generation 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 [Dij59]. The search only considers edges on which the generation of the turbine can be routed, i. e., it ignores congested edges. We then route the generation of u along P to v.

The initialization strategies differ along three dimensions: the metric for shortest paths, how turbine generation is collected, and the target substation. As for the metrics, we either use the lengths defined by len (cf. Section 4.1) or we assume a length of 1 for every edge. Turbine generation 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 generation of u along P, i. e., the flow along P is increased by 1. The other variant greedily collects as much generation 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 flow 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 generation 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 generation is collected greedily.

4.4.6 Delta Strategies

The delta strategies determine, in which order the different values for Δ are used to compute the residual costs. 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 cancelations, 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).

4.5 Experimental Evaluation of Negative Cycle Canceling

In Section 4.4, we have introduced a heuristic with various strategies for WCP. We first use statistical tests to evaluate these strategies and identify the best ones (Section 4.5.1). We compare the best *variant* (i. e., best combination of initialization and delta strategy) of our NCC algorithm with different baseline approaches for WCP namely solving an exact MILP formulation (Section 4.5.3) and a Simulated Annealing (SA) algorithm from the literature [LRWW17] (Section 4.5.4). In preliminary experiments (Section 4.5.2) we determine which of the MILP solvers

Table 4.1: Cable types from [BVMO11] used in the evaluation given by their capacity in number of turbines and their cost per unit of length.

Cable Type	a	b	c	d
Capacity cap_K	5	8	12	15
Cost c_K	20	25	27	41

Gurobi and CPLEX works better for WCP to establish which solver we compare our NCC algorithm to. The evaluation can also be found in our joint work with Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf [Gri+19]; Section 4.5.2 is only contained in the arXiv version. The underlying data can be found in [Çak+23].

For our evaluation we use the five benchmark sets $\mathcal{N}_1$–$\mathcal{N}_5$ for wind farms from [LRWW17] 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 and 2–7 substations, $\mathcal{N}_3$: 80–180 turbines and 4–9 substations, $\mathcal{N}_4$: 200–499 turbines and 10–39 substations), and complete graphs ($\mathcal{N}_5$: 80–180 turbines and 4–9 substations). We use the same cable types used for the evaluation in [LRWW17], which are originally from [BVMO11]. The cable types are shown in Table 4.1. 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. All computations are run in single-threaded mode to ensure comparability of the different algorithms.

4.5.1 The Best Variant of our Algorithm

The first goal of the evaluation is to establish which of all variants of our NCC algorithm works best. We use a two-stage process: Determine the best delta strategy across all initialization strategies. With the best delta strategy fixed, find the best initialization strategy for this delta strategy.

For the first stage, we randomly select 200 instances per benchmark set and run our algorithm on each instance with every pair of delta and initialization strategies. The first eight rows of Table 4.2 show the minimum, average, and maximum running times for each delta strategy across all initialization strategies, separated by benchmark set. 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. We see that `Dec` is always the slowest strategy on average, which can be explained by the fact that `Dec` often tries large values for Δ, for which negative cycles are rarely found. The other strategies all roughly terminate in the same time on average. It seems to be slightly faster to repeat the same Δ. However, for our purpose all variants have small enough running times. We therefore base our decision which variant to choose solely on their solution qualities.

Table 4.2: Minimum, average and maximum of running times in milliseconds of different variants. Running time measurement starts before the initial flow is computed and ends with the termination of the algorithm prior to outputting the solution. The first eight rows represent running times across all initialization strategies per delta strategies and benchmark sets. The best delta strategy in terms of solution quality is marked in green; minimal values per column are marked in yellow. The last row represents the algorithm variant `IncDec`, `CollectingDijkstraAny`.

Delta Strategy	$\mathcal{N}_1$			$\mathcal{N}_2$			$\mathcal{N}_3$			$\mathcal{N}_4$			$\mathcal{N}_5$		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max	min	avg	max
`Dec`	1.10	81.1	535	5.25	142.1	857	282	1.8k	11.3k	4.4k	59.7k	272k	2.4k	30.8k	216k
`Inc`	0.45	46.2	361	2.69	78.0	531	174	1.2k	8.4k	3.0k	49.1k	213k	1.8k	16.2k	131k
`IncDec`	0.45	45.9	433	2.67	77.7	539	174	1.2k	8.2k	3.0k	48.7k	212k	1.9k	16.2k	117k
`Random`	0.62	43.9	288	3.50	77.3	443	176	1.0k	5.9k	3.2k	32.6k	137k	1.9k	16.6k	143k
`S-Dec`	0.76	62.2	461	3.76	111.4	725	210	1.4k	9.1k	3.5k	47.4k	206k	1.9k	14.7k	133k
`S-Inc`	0.46	42.2	295	2.70	72.8	438	171	1.0k	6.6k	2.8k	36.2k	147k	1.8k	14.4k	97k
`S-IncDec`	0.45	42.2	310	2.68	72.7	437	171	1.0k	6.3k	2.8k	36.0k	154k	1.8k	14.4k	120k
`S-Random`	0.57	44.1	333	3.25	79.1	486	193	1.1k	6.0k	3.0k	35.1k	147k	1.7k	14.1k	106k
`BestVar`	0.48	36.2	217	3.51	52.6	257	174	706	3.1k	3.0k	27.0k	92.6k	1.9k	13.4k	82.6k

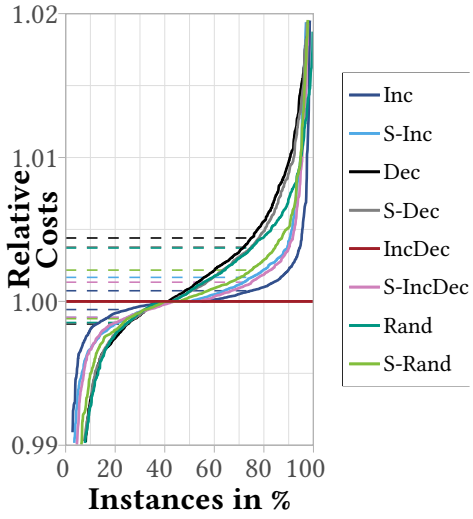
Table 4.3: Comparison of delta strategies over all initialization strategies. An entry in row i and column j shows on how many instances strategy i produces better solutions than strategy j. Values are marked by a star if they are significant with $p < 10^{-2}$ and by two stars if $p < 10^{-4}$. The best strategy is marked in green.

	Inc	Dec	IncDec	Random	S-Inc	S-Dec	S-IncDec	S-Random
Inc	—	60.6 %**	48.4 %	60.2 %**	54.2 %	59.4 %**	50.7 %	56.5 %*
Dec	39.4 %	—	38.9 %	46.7 %	40.8 %	48.4 %	40.6 %	41.2 %
IncDec	51.6 %	61.1 %**	—	59.9 %**	54.0 %	60.1 %**	50.8 %	57.3 %*
Random	39.8 %	53.3 %	40.1 %	—	42.4 %	52 %	42.7 %	43.4 %
S-Inc	45.8 %	59.2 %**	46.0 %	57.6 %*	—	58.1 %**	46.9 %	54.7 %
S-Dec	40.6 %	51.6 %	39.9 %	48.0 %	41.9 %	—	41.7 %	41.9 %
S-IncDec	49.3 %	59.4 %**	49.2 %	57.3 %*	53.1 %	58.3 %**	—	55.4 %
S-Random	43.5 %	58.8 %**	42.7 %	56.6 %*	45.3 %	58.1 %**	44.6 %	—

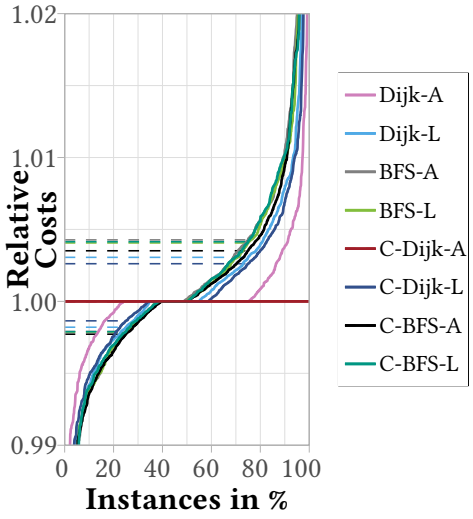
To compare the variants in terms of solution quality, we compute for each delta strategy i and instance m the mean $X_m^{(i)}$ of the solution values over all eight initialization strategies. This gives us 1000 data points per delta strategy. For delta strategies i, j we perform a Binomial Sign Test as described in Section 3.4 counting instances with $X_m^{(i)} < X_m^{(j)}$ and $X_m^{(j)} < X_m^{(i)}$ (Section 3.4). Table 4.3 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 %, three of which are significant at the 10^{-4}- and another one at the 10^{-2}-level. The smallest value (50.8 % in column `StayIncDec`) stands for 460 instances on which `IncDec` performs better than `StayIncDec`. To the contrary, there are 446 instances on which `StayIncDec` yields better solutions (cf. entry 49.2 % 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 4.7 (a) we show the average costs by delta strategy in relation to the `IncDec` strategy: For example, 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 accordingly. The minimum ratios range between 0.870 (`Random`) and 0.947 (`Inc`) and the maximum ratios are between 1.027 (`Random`) and 1.104 (`StayIncDec`). We see, for example, that `IncDec` works strictly better than `StayInc` on 49.6 % and equally on 8.1 % of all instances and on 4.5 % of all instances `IncDec` outperforms `Inc` by at least 0.5 % in cost ratio.



(a) The delta strategies are presented relative to the `IncDec` strategy. Solution values represent the average over all initialization strategies.



(b) The initialization strategies are presented relative to the `CollectingDijkstraAny` strategy with fixed delta strategy `IncDec`.[4]

Figure 4.7: Comparison of our 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.

Table 4.4: 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	—	55.8 %	49.5 %	54.9 %	55.6 %	53.7 %	53.9 %	56.5 %*
BFS-A	44.2 %	—	42.7 %	46.5 %	47.6 %	51.1 %	46.7 %	49.3 %
C-Dijk-A	50.5 %	57.3 %*	—	55.3 %	56.5 %	56.5 %*	54.4 %	56.3 %
C-BFS-A	45.1 %	53.5 %	44.7 %	—	51.2 %	54.5 %	49.3 %	55.4 %
Dijk-L	44.4 %	52.4 %	43.5 %	48.8 %	—	50.4 %	48.1 %	51.7 %
BFS-L	46.3 %	48.9 %	43.5 %	45.5 %	49.6 %	—	47.7 %	53.7 %
C-Dijk-L	46.1 %	53.3 %	45.6 %	50.7 %	51.9 %	52.3 %	—	53.1 %
C-BFS-L	43.5 %	50.7 %	43.7 %	44.6 %	48.3 %	46.3 %	46.9 %	—

[4] Note that in [Gri+19], this plot erroneously included the mean values across all delta strategies.

Next, we want to find the best initialization strategy after fixing `IncDec` as the delta strategy. We pair each initialization strategy with `IncDec` on the same 1000 instances and summarize the results of all pairwise tests after Bonferroni-correction with factor 112 in Table 4.4. We see that both initialization strategies using Euclidean distances and routing turbine generation to the nearest free substation, i. e., `DijkstraAny` and `CollectingDijkstraAny`, seem to work best. In particular, these two initialization strategies stand out by providing better solutions more often than not compared to any of the other initialization strategies. Furthermore, these are the only initialization strategies that show some significant advantage over other strategies. In Figure 4.7 (b) we depict ratios of solution values compared to `CollectingDijkstraAny`. The minimum ratios are between 0.886 and 0.923 for all strategies other than `DijkstraAny` (0.974). The maximum ratios range between 1.054 and 1.085. For the main part there is hardly any difference between collecting strategies and their non-collecting counterparts. The figure shows, e. g., that on roughly 22 % of all instances `CollectingDijkstraAny` is better than `BFSLast` and `CollectingBFSLast` by 0.5 %. `CollectingDijkstraAny` has a slight but not significant advantage over `DijkstraAny`, the other initialization strategies fall behind quite remarkably. We therefore declare `CollectingDijkstraAny` paired with `IncDec` as our best variant.

The last row in Table 4.2 shows the running time characteristics of `CollectingDijkstraAny` paired with `IncDec`. Running times range between tenths of milliseconds and 93 seconds.

4.5.2 Comparing MILP Solvers to Establish Baseline Solver

We conduct preliminary experiments to determine which MILP solver we use as a baseline for our algorithm. To this goal, we randomly choose 35 instances each from benchmark sets $\mathcal{N}_1$, and $\mathcal{N}_2$ and 70 instances each from benchmark sets $\mathcal{N}_3$, $\mathcal{N}_4$, and $\mathcal{N}_5$. We compare Gurobi 8.0.0 [Gur18] and IBM ILOG CPLEX Optimization Studio v12.8 [IBM17] with a running time of one day per instance and solver using the MILP formulation from Section 4.3. Since computing an optimal solution to the MILP takes too long in almost all instances, we restrict the solvers to different maximum running times. Each solver uses one thread per instance and node files are written to disk after the solver uses more than 0.5 GB of memory to store node files. Other than that, default values are used.

During the experiments, we consider three time stamps: one hour, twelve hours, and one day. For each solver, instance, and time stamp we record the value of the best incumbent solution and the MIP gap (as explained in Section 4.3). If a solver terminates with a proven optimal solution before time stamp t, then the respective values during termination are assigned to time stamp t and all subsequent time stamps.

The results of the experiment are depicted in Figure 4.8 for the quality of the best solution found by the respective solver and in Figure 4.9 for a comparison of MIP gaps. In Figure 4.8 the boxplots represent one data point for each instance and time stamp. The value on the abscissa stands for a normalized difference in solution values, i. e., $(\text{sol}_{\text{Gurobi}} - \text{sol}_{\text{CPLEX}}) / \max(\text{sol}_{\text{Gurobi}}, \text{sol}_{\text{CPLEX}})$. This yields a value in $[-1, 1]$, which is negative if and only if Gurobi finds a better solution than CPLEX.

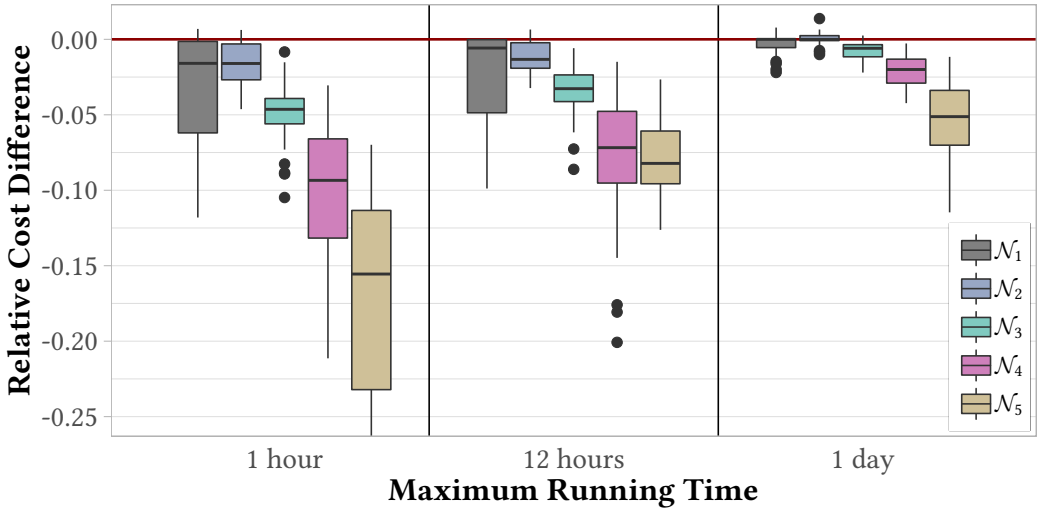


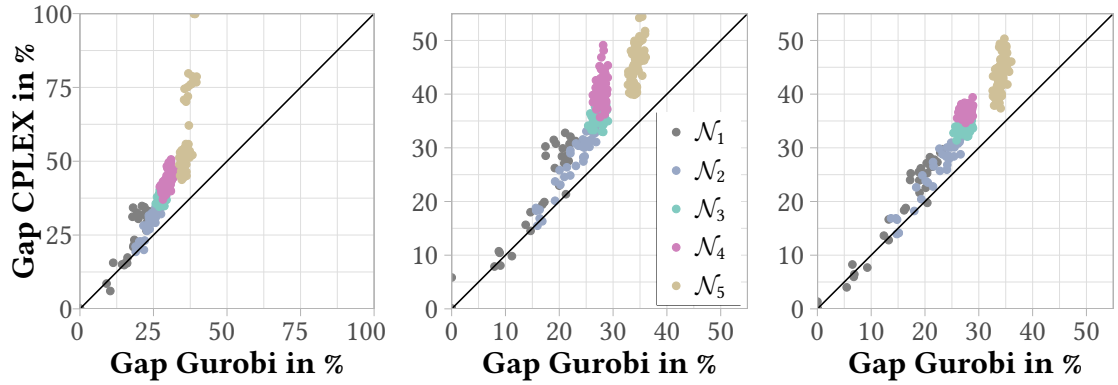
Figure 4.8: Comparison of solution values found by MILP solvers CPLEX and Gurobi after different running times. For each instance and running time the abscissa shows a normalized difference in solution values, i. e., $(\text{sol}_{\text{Gurobi}} - \text{sol}_{\text{CPLEX}}) / \max(\text{sol}_{\text{Gurobi}}, \text{sol}_{\text{CPLEX}})$. After a running time of one hour, there are twelve instances from $\mathcal{N}_5$ with a value between -0.29 and -0.49 and another three instances from $\mathcal{N}_5$ with a value less than -0.99. Four instances from $\mathcal{N}_5$ are infeasible and not included in this figure.

Figure 4.9 shows the *MIP gaps* computed by CPLEX and Gurobi for each instance and time stamps. This value is in the unit interval and gives information on how "bad" the solution value can be compared to the (unknown) optimal value.

Evidently, Gurobi performs better across all benchmark sets and time stamps, except for $\mathcal{N}_2$ after a running time of one day where CPLEX has an ever so slight advantage. While there is evidence that the best incumbent solutions computed by Gurobi and CPLEX become more similar the longer the experiments run, we also see that Gurobi seems to work better than CPLEX the bigger the instances become. We therefore use Gurobi as the MILP solver to compute the baseline to which we compare our Negative-Cycle-Canceling-based algorithm.

4.5.3 Comparing our Best Variant with Gurobi

We compare our NCC algorithm in its best variant, i. e., `CollectingDijkstraAny` with `IncDec`, with Gurobi on the MILP formulation in Section 4.3. We randomly select 200 instances per benchmark set from the benchmark sets in [LRWW17], independently chosen from the previous experiments. To increase readability, we identify the MILP formulation and Gurobi solving this formulation.



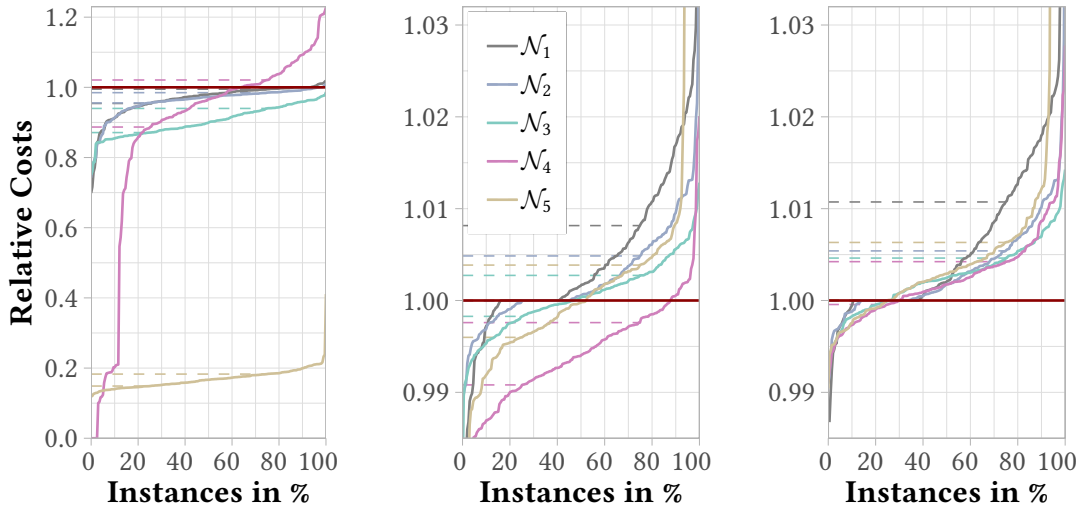
(a) Running time: one hour **(b)** Running time: twelve hours **(c)** Running time: one day

Figure 4.9: Comparison of MIP gaps for solutions computed by CPLEX and Gurobi separated by benchmark sets after different maximum running times. Each subfigure stands for a different maximum running time, after which the respective best upper and lower bounds are used to compute the MIP gaps. Each data point represents one instance; the ordinate states the MIP gap computed by Gurobi, the abscissa the gap computed by CPLEX. The four infeasible instances from $\mathcal{N}_5$ are omitted.

Before we talk about the solution quality, let us quickly revisit the running times of our NCC algorithm. The running times quite closely match those shown in Table 4.2. Of biggest interest is probably the maximum running time across all instances: There is one instance from $\mathcal{N}_5$ with a running time of 97.4 s, all other instances, in particular all from $\mathcal{N}_4$ are smaller than 90 s.

In Figure 4.10 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. These running times represent both interactive and non-time-critical planning. Since our algorithm terminates in under 100 seconds, the comparisons in Figures 4.10 (b) and 4.10 (c) use the solution our algorithm provides at termination. While discussing the plots, we also discuss an adaptation of the MIP gaps $^{\text{ub}-\text{lb}}/_{\text{ub}}$ we introduced in Section 4.3. For each instance, we use the lower bounds from the one-day MILP experiments. For each instance, each maximum running time and for both the MILP and the NCC algorithm take the best solution value (ub) found at the maximum running time. We refer to the relative gaps as *MIP gap* and *NCC gap*, respectively, and show them in Figure 4.11.

After two seconds (Figure 4.10 (a)) our algorithm outperforms Gurobi on all benchmark sets as it finds better solutions on 89 % of all instances with the lowest percentage on benchmark set $\mathcal{N}_4$. On $\mathcal{N}_1$ the NCC gaps are on average 14.1 % with a maximum of 24.8 % compared to MIP gaps of 16.9 % on average and at most 43.1 %. For $\mathcal{N}_3$, the NCC gaps are on average 27.6 % with a spread of only seven percentage points, compared to a mean of 34.6 % and a maximum of 45.4 % for the MIP gap. The values for $\mathcal{N}_2$ range between those for $\mathcal{N}_1$ and $\mathcal{N}_3$. The ratios of



(a) Running time: two seconds **(b)** Running time: one hour **(c)** Running time: one day

Figure 4.10: Comparison of our NCC algorithm to Gurobi on 200 instances per benchmark set. For each instance and three different maximum running times, the ratio of the best solution value found by NCC to the best solution value found by Gurobi are computed and shown in increasing order along the ordinate. The dashed lines represent the 25 % and 75 % quantiles of the instances.

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. 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 18 instances on which the ratio of solution values exceeds 1.1 with a maximum of 1.228. On those very large instances, detecting negative cycles takes longer and fewer iterations are performed in two seconds. The NCC gaps spread between 31.6 % and 57.4 % with an average of 42.6 %. The MIP gaps are even worse with a mean value of 48.3 % 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 0.411). The gaps are on average at 53.6 % for the NCC algorithm and at 92.3 % for Gurobi.

Within one hour (Figure 4.10 (b)) Gurobi finds better or equivalent solutions compared to our algorithm on a majority of the instances in benchmark sets $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$. On 25 % of the instances from $\mathcal{N}_1$, on 18.5 % of the instances from $\mathcal{N}_2$, and on one instance from $\mathcal{N}_3$ the solution values are equal. On $\mathcal{N}_4$ and $\mathcal{N}_5$, our algorithm still yields better solutions on 87.5 % and 52 % of the instances, respectively. Our algorithm is within 0.5 % of Gurobi's best solution on 81.4 % and within 1 % on 91.3 % of all instances. Only on six of 1000 instances (all in $\mathcal{N}_5$), the ratio

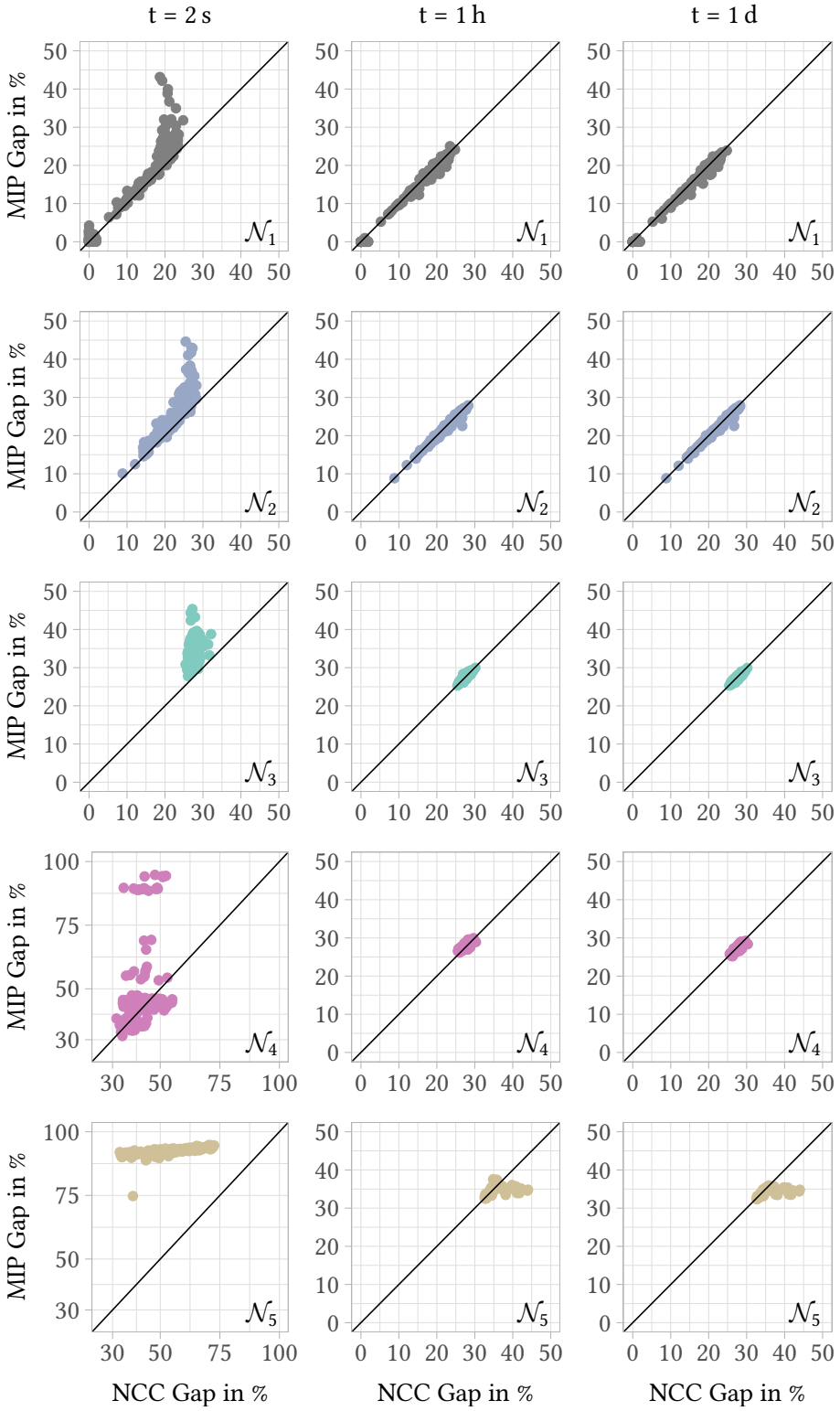


Figure 4.11: Comparison of gaps between solution values and lower bounds on the optimal value for solutions computed by Gurobi (MILP) and the NCC algorithm separated by benchmark sets ($\mathcal{N}_1$ in row 1 through $\mathcal{N}_5$ in row 5) after maximum running times of two seconds, one hour, and one day. Lower bounds are taken from MILP experiments with running times of one day.

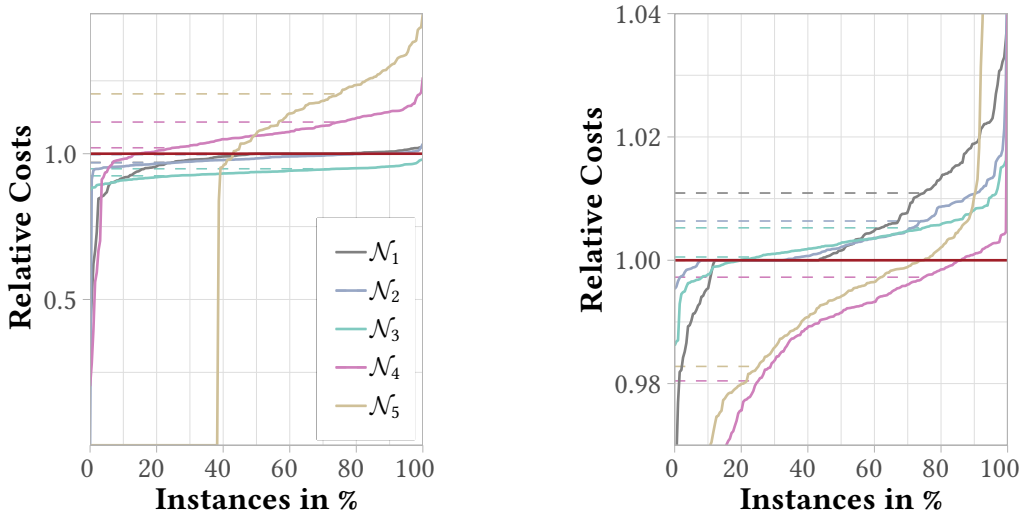
Table 4.5: Running time characteristics for those instances, on which Gurobi finds equal or better solutions than the NCC algorithm. Columns 2 and 3 count instances, the remaining columns give an overview of the running times.

Set	Instances: Gurobi Better		Running Time Characteristics in Seconds				
	total	Gurobi Also Faster	min	1st Quartile	median	3rd Quartile	max
$\mathcal{N}_1$	179	0	0.10	5.61	31.7	288	51 516
$\mathcal{N}_2$	172	0	0.11	20.3	107	872	70 293
$\mathcal{N}_3$	147	0	12.6	585	1299	3751	46 237
$\mathcal{N}_4$	140	0	44.9	5274	14 614	31 757	81 968
$\mathcal{N}_5$	150	0	10.9	612	1975	7357	63 904

exceeds 1.10 with a maximum of 1.165. That means, while the NCC algorithm is comparable to Gurobi in solution quality on small instances, it proves better on larger wind farms. Furthermore, our algorithm is much faster since it terminates in under 100 seconds—compared to one hour of maximum running time for Gurobi.

After running times of one day (right plot in Figure 4.10), while our algorithm is at least as good as Gurobi only on between 25 % ($\mathcal{N}_5$) and 38.5 % ($\mathcal{N}_1$) of the instances, it is within 1 % of Gurobi's solution on 87.7 % of all instances. Again, there are only six instances with a ratio worse than 1.10 with a maximum of 1.169. 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 one day, the gaps look vastly the same and there is hardly any difference between NCC gaps and MIP 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.

For those instances on which Gurobi computes worse solutions than our NCC we can conclude that our algorithm yields better solutions in shorter amount of running times. For the other instances, it is interesting to compare the running times Gurobi needs to find at least equivalent solutions compared to our algorithm. Gurobi might find those solutions faster or it might need much more running time to do so. In the latter case, our algorithm trades a bit of solution quality (cf. Figure 4.10) for substantial savings in running time. Table 4.5 gives an overview of the running time Gurobi needs to outperform our NCC algorithm. For each benchmark set, we show the number of instances on which Gurobi finds solutions of equal or better value and the number of instances for which it does so faster (left part of the table). For all instances with at least equivalent Gurobi solutions, we show the sample characteristics of the running times at which Gurobi reaches or overtakes the NCC algorithm in solution quality for the first time (right part of the table). We see that even when Gurobi is better, it is never faster. Thus, our NCC algorithm computes slightly worse solutions in shorter running times. In particular on the two smallest benchmark sets, the running times as a matter of milliseconds (cf. last row



(a) Running time: two seconds **(b)** Running time: one hour

Figure 4.12: Comparison of NCC algorithm to the Simulated Annealing algorithm on 200 instances per benchmark set. For each instance and two different maximum running times, the ratio of the best solution value found by NCC to the best solution value found by SA are computed and shown in increasing order along the ordinate. The dashed lines represent the 25 % and 75 % quantiles of the instances.

of Table 4.2) are a lot faster than Gurobi is. On the other benchmark sets, already at the first quartile of instances, does Gurobi need roughly ten minutes to find equivalent solution. On the biggest instances, i. e., set $\mathcal{N}_4$, this time rises to approximately one and a half hours. In the median, Gurobi needs more than 20 minutes, while our algorithm terminates after one and a half minutes (and often a lot faster).

In summary, these experiments show that our NCC algorithm is a viable option compared to Gurobi. If only a short amount of time is given, our algorithm outperforms Gurobi. After longer running times, the solution quality of our algorithm remains competitive, while it maintains substantial savings in running times.

4.5.4 Comparison to Metaheuristic Simulated Annealing

We compare our best algorithm variant with the best variant of a Simulated Annealing (SA) algorithm [LRWW17]. We run the SA algorithm and the NCC 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 4.12). Considering a running time of one day is not sensible since our algorithm terminates within tens of seconds and the SA algorithm has only been considered for short running times of at most half an hour [LRWW17, p. 206].

After two seconds, the NCC algorithm performs at least as good as the SA algorithm on all instances from $\mathcal{N}_3$ and on 74.5 % and 90.5 % on $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. The minimum ratios of solution values are 0.381 for $\mathcal{N}_1$, 0.911 for $\mathcal{N}_2$, and 0.875 for $\mathcal{N}_3$ 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 1.034. 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 71 % 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 1.261 for $\mathcal{N}_4$ and from 0.838 to 1.480 for $\mathcal{N}_5$ (not taking into consideration the instances without a solution from the SA algorithm).

After one hour, the SA algorithm provides better solutions than our algorithm on 67.5 % and 80 % of instances from $\mathcal{N}_2$ and $\mathcal{N}_3$, respectively. Our algorithm, however, stays within 1 % in solution quality on 84.2 % 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 86 % and 74.5 %, 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 0.905 and 0.995 for the other benchmark sets. The maximum ratios are at most 1.057 for all benchmark sets except $\mathcal{N}_5$ (1.159).

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.

4.5.5 Lessons Learned

The evaluation has shown that our NCC algorithm is able to compute solutions competitive to an MILP formulation solved by Gurobi to a solution approach using Simulated Annealing. It achieves these solutions in shorter running times than its competitors: On the smallest instances our algorithm terminates within milliseconds; on the biggest instances it needs on average half a minute with worst-case running times of around one-and-a-half minutes. The MILP, to the contrary, needs more than four hours to compute equivalent solutions in the median case. Thus, our algorithm is well-suited to compute good cable layouts in short running times. This can be used, for example, in preliminary planning steps when an estimate for cable layouts for various placements of turbines and substations is needed.

We have also seen from the combination of short running times and slightly worse solution quality compared to the MILP that our algorithm runs into local minima from which it cannot escape. To adress this, we develop strategies to perturb solutions in a way that further cycle canceling is possible. This yields an Iterated Local Search built upon our Negative Cycle Canceling algorithm which we discuss in the next section.

4.6 Negative Cycle Canceling in an Iterated Local Search

The evaluation of our Negative Cycle Canceling adaptation has shown that the algorithm yields competitive solutions within tens of seconds compared to, for example, an MILP with a maximum running time of one day. Even in those cases when Gurobi finds better solutions than our algorithm, it needs substantially more time to do so, as we have seen in Table 4.5. While NCC terminates quickly with competitive solutions, it also gets stuck in local minima from which it cannot escape. We can spend some of the running time advantage to find more (and hopefully better) solutions. While we could simply start with different initializations (and we have a bunch of these!), it is more sensible to change a locally optimal solution ever so slightly that NCC can proceed in order to find a different (and hopefully better) locally optimal solution and thereby escape the previous minimum. One way to combine a heuristic algorithm with strategies to escape local minima is known as *Iterated Local Search (ILS)* and it is said to be "much better than random restart" [LMS19, p. 136]. ILS has successfully been applied to a Fixed-Charge Transportation Problem [BRT14] and to a Water Distribution Network Design Problem [DS16].

In this section, we give an introduction into ILS presenting its main building blocks. In Section 4.6.1 we describe how we populate these building blocks and what our strategies for escaping local minima are. The experimental evaluation can be found in Section 4.7. The elaborations from Sections 4.6.1 and 4.7 are based on joint work with Dorothea Wagner and Matthias Wolf [GWW20].

We start by giving an overview of Iterated Local Search, following the description in [LMS19]. The general layout of ILS is depicted in Algorithm 4.2.

Algorithm 4.2: Iterated Local Search [LMS19, Algorithm 1]

```
1 s_0  = GenerateInitialSolution()
2 s*   = LocalSearch(s_0)
3 repeat
4 |   s'  = Perturbation(s*, history)
5 |   s'* = LocalSearch(s')
6 |   s*  = AcceptanceCriterion(s*, s'*, history)
7 until termination criterion met
```

At the core of ILS there is a heuristic algorithm (also referred to as a Local Search algorithm, even though it need not be one). After one invocation of the Local Search algorithm (Line 2), a first local optimum $s^\star$ is found. Then, the solution $s^\star$ is changed in a certain way ("perturbed", Line 4) yielding "an intermediate [solution] s'" [LMS19, p. 134], which is used as a starting point for another invocation of the Local Search (Line 5). The resulting solution $s'^\star$ may or may not replace the previous incumbent solution $s^\star$, depending on an acceptance criterion (Line 6). This sequence of perturbation and Local Search is iterated until a certain termination critierion is met (Line 7).

Iterated Local Search yields a sequence of locally optimal solutions. Both the perturbation step and the acceptance criterion may depend on previous solutions in this sequence. The authors of [LMS19] identify two advantages of an approach using ILS over random restarts of the Local Search algorithm: Not only do local searches "usually execute much faster on a solution obtained [from] a small perturbation" of a locally optimal solution than on a random solution [LMS19, p. 142] but also the sampling of solutions by Local Search will return solutions whose values "arbitrarily peak" around "a fixed percentage above the optimum" [LMS19, Section 5.2.2], which can be overcome by ILS. In particular with the perturbation step, algorithm engineers have to thread the needle: If a solution is perturbed too much, ILS looses the advantage in solution values over restarts of the Local Search; if the perturbation is not strong enough, the subsequent Local Search will not be able to obtain a new solution [LMS19, p. 139]. In the upcoming Section 4.6.1, we explain how we populate the ILS framework with our NCC algorithm and suitable perturbations.

4.6.1 ILS: Negative Cycle Canceling with Neighborhood Heuristics

In the following we call our Negative-Cycle-Canceling-based algorithm presented in Section 4.4 the/our *(standard) NCC algorithm* or *NCC algorithm without escaping*. This algorithm is the Local Search algorithm for the purpose of the ILS which we present here. As mentioned before, the goal of perturbations is to escape a local minimum computed by the standard NCC algorithm. We therefore call the different perturbations we propose the *escaping strategies* or *neighborhood heuristics* since the strategies enforce rather local changes. The resulting ILS algorithm is referred to as the *NCC algorithm with escaping* or simply as *our ILS*. We assign a weight to each escaping strategy, so that, whenever a solution needs to be perturbed, our ILS picks one strategy out of all available ones randomly according to the weights.

Drawing again connections to the general framework presented in Algorithm 4.2, our ILS works as follows: Our ILS starts by initializing a flow on the wind farm as specified by the standard NCC algorithm. This algorithm then performs its "local search" until no further negative cycle can be canceled (Lines 1 and 2). While the standard NCC algorithm terminates at this point, our ILS picks one of its available escaping strategies randomly according to the specified weights and applies it to the current flow (Line 4). If this strategy changes the flow (no matter if to the better or worse), another round of Negative Cycle Canceling is run until no more negative cycles are found (Line 5). This round of Negative Cycle Canceling is subject to adapted residual costs as explained below in detail for each strategy. This adaptation helps to prevent the fall-back to the previous solution during the ensuing NCC run. and is replaced by another adaptation once an escaping strategy successfully changes the flow.[5] If an escaping strategy is not able to change the flow, another escaping strategy is picked and the unsuccessful strategy will not be picked again until the current flow has been changed. We refer to one pick of an escaping strategy with the ensuing NCC run (in case of changes) as one *iteration*

[5] There is a certain resemblance to Tabu Search [GP19] here since the adaptations try to lead the search away from the previous solution, even though it is not strictly forbidden in our case.

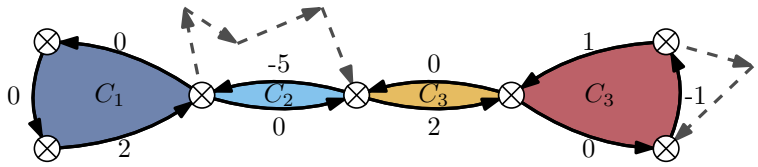


Figure 4.13: Visualization of the *Deal with Bonbon* strategy. A negative closed walk without a long negative cycle has been found. For each edge with negative residual costs, a shortest path (marked by dashed gray edges) is computed. If the shortest path and the walk form a negative cycle, that cycle is canceled.

of the ILS. As for the acceptance criterion (Line 6), the search always continues on the latest output from the standard NCC algorithm, $s'^{\star}$, but the overall best solution is recorded and given as the output solution upon termination. The ILS terminates if no escaping strategy is able to change the current flow anymore or if a given time limit is exceeded.

The three escaping strategies we propose are:

Free Upgrade (U). This strategy identifies all edges using a saturated cable type, i. e., edges with a non-zero flow such that adding one additional unit of flow incurs the need for a bigger cable type. The strategy then performs a single run of the NCC algorithm with $\Delta = 1$ in which all residual costs are computed normally except for the aforementioned edges. Their residual costs are set to zero instead of a finite positive residual cost, meaning that the bigger cable type can be used without incurring additional costs. If a cycle is canceled, all residual edges with a previously saturated cable type on that cycle are identified. For these edges, the residual costs for the remainder of the iteration are adjusted to reflect the free upgrade, i. e., the costs for the new cable type and all cable types with more capacity are reduced by the cost for the upgrade.

Move Leaf (L). This strategy identifies all leaves, i. e., turbines without incoming flow. It iterates over all leaves one-by-one and checks if they have a shorter outgoing edge than the one that transmits the current turbine's generation. In that case, one unit of flow is rerouted via the shorter edge to a substation with free capacity using only those edges whose flow is non-zero and less than the maximum cable capacity—if such a path exists. Then, for the remainder of this iteration, the residual costs of the leaves' new outgoing edges are adjusted in the sense that the cost for the cheapest cable type is given for free.

Deal with Bonbon (B). The adaptation of the Bellman-Ford algorithm used in the standard NCC algorithm may also identify negative sets of cycles in the residual graph which do not improve the flow when being canceled (cf. Section 4.4.4). The reason for those sets is that negative cycles consisting of two edges only may exist. Even if the negativity suggest a decrease of the total costs, canceling such a two-edge cycle does not change the flow at all. The drawing of such a set of cycles in Figure 4.13 inspired the name *bonbon* for such a set of cycles.

The *Deal with Bonbon* strategy is specifically designed to overcome the aforementioned downside of finding "unhelpful" sets of cycles by capitalizing on such sets. This escaping strategy makes use of such an unhelpful set of cycles as follows: It first identifies all negative residual edges on the bonbon. Then it tries to find paths in the residual graph that close a negative cycle when merged with a negative edge (and possibly other edges) from that bonbon. These are indicated by dashed, gray edges in Figure 4.13. The strategy therefore runs again the Bellman-Ford algorithm with the same setting (including the adapted residual costs from the previous iteration if applicable) for which the unhelpful bonbon was initially found. For each negative edge on the bonbon it considers each incoming edge in the graph. From there it traverses the parent pointers computed by the Bellman-Ford algorithm until a cycle is closed. If the cycle is negative, then it is canceled and the escaping strategy terminates. Otherwise, the search continues. This strategy does not change the standard residual cost computation for the following NCC run.

4.7 Experimental Evaluation of the Iterated Local Search

We provide an experimental evaluation of the ILS presented in Section 4.6.1: The goal of Section 4.7.1 is to determine which set of escaping strategies is the most fruitful. In Section 4.7.2 we investigate the improvement ILS provides over the standard NCC algorithm and in Section 4.7.3 we compare the solutions computed by the ILS to solutions from the MILP. As with our standard NCC algorithm, we also provide a comparison of ILS and SA (Section 4.7.4). The evaluation can also be found in joint work with Dorothea Wagner and Matthias Wolf [GWW20]; for the underlying data refer to [Çak+23].

The ILS is implemented in C++14 and compiled with GCC 8.2.1 using the `-O3 -march=native` flags. All simulations run in single-thread mode (to ensure comparability) on a 64-bit architecture with four 12-core AMD-CPUs clocked at 2.1 GHz with 256 GB RAM running OpenSUSE Leap 15.1. As with the evaluation of standard NCC algorithm in Section 4.5, we use the cable types and benchmark instances from [LRWW17].

4.7.1 Comparing Sets of Escaping Strategies

In a first step, we want to compare different sets of escaping strategies to see how they perform in helping the NCC algorithm to move away from local minima. We consider seven different sets. The sets are given by the three escaping strategies from Section 4.6.1 with each escaping strategy having either weight 0 or 1. The all-zero set is not considered. We refer to the sets by the letters from each strategy in the set with weight 1. Hence, the set referred to as UB uses the strategies *Free Upgrade* and *Deal with Bonbon* with a weight of 1 each but does not use the strategy *Move Leaf*. For the simulations, we randomly select 200 instances from each of the five benchmark sets in [LRWW17].

Table 4.6: Number of instances where escaping strategies yield better solutions than the NCC algorithm without escaping.

B	U	UB	L	LB	UL	ULB
87	297	327	434	460	580	603

Table 4.7: Comparison of sets of escaping strategies. An entry in row i and column j shows on how many instances setting i produces better solutions than setting j. Values are marked by a star if they are significant with $p < 10^{-2}$ and by two stars if $p < 10^{-4}$ according to the Binomial Sign Test. The best set of strategies is marked green.

	B	L	LB	U	UB	UL	ULB
B	—	8.9 %	0 %	17.8 %	8.5 %	4.3 %	0.3 %
L	91.9 %**	—	23.3 %	70.6 %**	65.7 %**	14.9 %	14.2 %
LB	100 %**	76.7 %**	—	74.8 %**	71.9 %**	23.0 %	16.9 %
U	82.2 %**	29.4 %	25.2 %	—	1.8 %	1.0 %	2.7 %
UB	91.5 %**	34.3 %	28.1 %	98.2 %**	—	6.5 %	4.0 %
UL	95.7 %**	85.1 %**	77.0 %**	99.0 %**	93.5 %**	—	36.1 %
ULB	99.7 %**	85.8 %**	83.1 %**	97.3 %**	96.0 %**	63.1 %*	—

Each set of strategies is run on each instance with a time limit of 15 minutes and the best solution value for the combination of strategy set and instance is recorded. In the terminology of Sections 4.4.5 and 4.4.6 we use `CollectingDijkstraAny` as the initialization and `IncDec` as the delta strategy. Those have been identified as the best strategies for the standard NCC algorithm (see Section 4.5.1).

Table 4.6 shows the numbers of instances out of the selected 1000 on which ILS using one of the aforementioned sets of escaping strategies yields a better solution to WCP than the standard NCC algorithm. For the *Deal with Bonbons* strategy this count equals the number of instances where there was a change to the flow at all. The other two strategies rely on the respective ensuing run of the NCC algorithm to find better solutions. In particular, for those strategies, a change to the flow does not necessarily yield an improvement to the best solution. The numbers in Table 4.6 show that *Deal with Bonbons* and *Free Upgrade* seem to have the most problems to allow finding better solutions with improvements on 9 % and 30 % of all instances, respectively. *Move Leaf* on its own allows finding better solutions on more than 43 % of the instances. But only in conjunction with the other escaping strategies, better solutions can be found on approximately 60 % of all instances.

These numbers only show whether a given set of escaping strategies improves the best solution in comparison to the NCC algorithm without escaping. They do not state how the best solutions compare between different set of escaping strategies. Therefore, we go back to the Binomial Sign Test as stated in Section 3.4: We count for each ordered pair (i, j) of set of strategies the number of instances n_i and n_j where the best solution found by i (and j,

respectively) is better than the one found by j (and i, respectively). If i and j were equally good, then $n_i \sim \text{Bin}(n_i + n_j, \theta)$ with $\theta = 0.5$ for each set of randomly and independently chosen instances. For each ordered pair we perform the one-sided Binomial Sign Test: We test the null hypothesis $H_0 : \theta = 0.5$ against the alternative hypothesis $H_1 : \theta > 0.5$. We apply a Bonferroni-correction by the number of tests, i. e., 42. We interpret rejecting the null hypothesis as setting i performing better than setting j. In Table 4.7 we show the ratios n_i/n_i+n_j in % and the corresponding significance levels. Note that as before, symmetric entries need not represent all 1000 instances since instances are omitted if both strategies find best solutions of same value.

The set of strategies which looked most promising in quantity according to Table 4.6 also turns out to provide better solutions than all the other sets. The difference between ULB and UL, however, is quite small: Only on 295 out of 1000 instances the best solutions are of different value. It may be somewhat surprising that UL yields better solutions on roughly one third of these instances, even though it does not employ *Deal with Bonbons* (which cannot worsen the best solutions). This can be explained by different trajectories in the order how escaping strategies are picked. Clearly, the results from Tables 4.6 and 4.7 show that ULB is the best set of strategies among those we considered in our simulations. We therefore use this set of escaping strategies when we compare our ILS to other approaches to WCP.

4.7.2 Improvement over Standard NCC

From the construction of our ILS it is obvious that it does not perform worse than our standard NCC algorithm since the output solution of ILS is the best solution found at any time in the course of the search. We want to see to what extent our ILS improves the solutions from the algorithm without escaping. To this end, we randomly draw 200 instances from each of the five benchmark sets, independently from previous selections. We run our standard algorithm on each of those instances. As stated in Table 4.2, the standard NCC algorithm terminates in approximately one and a half minutes on the biggest instances. We also run our ILS with escaping strategies ULB on each instance five times with different random seeds and a time limit of 15 minutes each. Performing multiple runs with different random seeds allows us to account for different trajectories of picks of escaping strategies.

From each of the five runs, we record the best solution found. From these five data points, we can isolate the worst and the median values and use them for the comparison to our NCC algorithm. The worst solution value can be interpreted as a "worst-case" analysis (even though by the literal meaning, it is not) and the median value can be seen as a "usual" run of ILS. Table 4.8 shows the number of instances from each of the five benchmark sets on which the worst and median solution values from the five ILS runs is better than the best solution from the NCC algorithm by a certain margin in percent. The $> 0\,\%$ column therefore displays the number of instances on which the respective ILS run yields a better solution than NCC. On the smallest instances, ILS better solutions only on about one in three instances for both the median and worst run. This number increases the bigger the instances become to about seven out of eight for $\mathcal{N}_4$. The numbers do not differ much between worst and median ILS solutions.

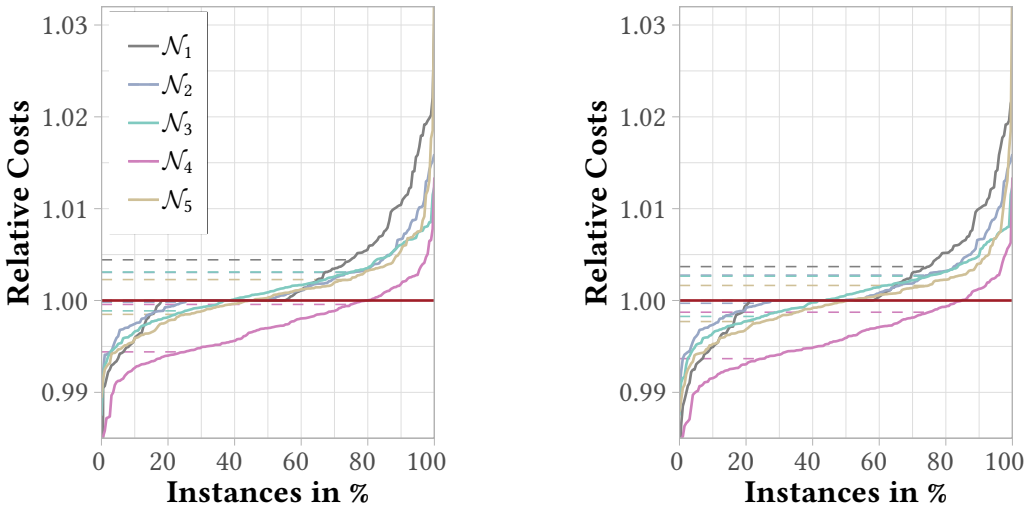
Table 4.8: Number of Instances for which the worst and the median of five ILS runs yield improvements of more than x % over the standard NCC algorithm

Set	Instances with Improvement					
	Worst ILS Run			Median ILS Run		
	$> 0\,\%$	$> 1\,\%$	$> 5\,\%$	$> 0\,\%$	$> 1\,\%$	$> 5\,\%$
$\mathcal{N}_1$	73	23	0	81	30	0
$\mathcal{N}_2$	65	3	0	74	4	0
$\mathcal{N}_3$	138	5	0	148	7	0
$\mathcal{N}_4$	170	14	1	179	20	1
$\mathcal{N}_5$	143	24	17	153	26	17
all	589	69	18	635	87	18

One might therefore think that running our ILS once suffices because the difference between a good and a bad ILS run is only marginal. We look into this more later when we include the MILP solutions as a reference solution.

It also stands out, that the number of instances with better solutions from ILS than from NCC rapidly decreases once we take certain margins of improvement into account, cf. the "$> 1\,\%$" and "$> 5\,\%$" columns from Table 4.8. While this may seem disappointing, it is not unexpected: We have seen in the comparison of our standard NCC algorithm and the MILP in Section 4.5.3, in particular Figure 4.10, that the margins between NCC and MILP are already quite small. Still, a notable improvement on some of the instances from $\mathcal{N}_5$ (the complete graphs) can be identified. This is also in line with Figure 4.10 where we have seen a tail of instances from $\mathcal{N}_5$ with surprisingly high solution values compared to the MILP solutions. We talk about one of those instances in the next part.

Another thing we see in the data obtained from this experiment (and all remaining ones as well) is that for most of the instances the sequence of locally optimal solutions is either very short or eventually periodic. This can be attributed to the implementation of the escaping strategies as deterministic algorithms. That means that each escaping strategy will always do the same thing on a given instance with a given locally optimal flow. While this is clearly a flaw in the implementation (and we address this in our elaborations of future research directions in Section 4.11), it cannot only be remedied by improving the implementation, but also does it strenghten our interpretation: Only a couple of applications of escaping strategies are needed to obtain the results we see and, with a bit more variety in the execution of the strategies, the performance of the ILS is likely to improve further.



(a) Worst of five ILS runs

(b) Median of five ILS runs

Figure 4.14: Comparison of ILS to Gurobi on 200 instances per benchmark set. For each instance, the ILS is invoked five times. The median and worst solution values out of these five runs is recorded for each instance and set in relation to the best solution value found by Gurobi. These ratios are shown in increasing order along the ordinate. The dashed lines represent the 25 % and 75 % quantiles of the instances.

4.7.3 Comparing the ILS to the MILP

Next, we want to see how our ILS compares to Mixed-Integer Linear Programming. To this end, we use Gurobi 9.0.0 [Gur18] on the formulation from Section 4.3 with a maximum running time of one day for each of the 1000 instances from Section 4.7.2.

We compute the ratios of the worst and the median ILS solution values to the best solution from the MILP for each instance, sort the ratios in increasing order and depict them by benchmark set in Figure 4.14. A value less than 1 in Figure 4.14 (a) means that all five ILS runs provided a better solution than Gurobi. In Figure 4.14 (b), accordingly, a value less than 1 means that at least three out of five runs yield a better solution than the MILP. For each instance, we identify the worst of the five separate best solutions after 15 minutes and use this value for the comparison to Gurobi. Picking the worst of the five solutions relates to a worst-case analysis of our ILS. We compute the ratio of this "worst-case value" to the best solution found by Gurobi at the conclusion of its one-day running time. A value less than 1 means that in all five runs of our ILS the best solution found is better than the solution given by Gurobi. Figure 4.14 shows these ratios in increasing order separated by benchmark set. The following outliers are not shown: in Figure 4.14 (a) 1.0362 from $\mathcal{N}_5$, 0.9846 from $\mathcal{N}_1$, and 0.9797 from $\mathcal{N}_4$, as well as 1.0362 from $\mathcal{N}_5$, 0.9846 from $\mathcal{N}_1$, and 0.9837 and 0.9758 from $\mathcal{N}_4$ in Figure 4.14 (b).

A side-by-side comparison of Figure 4.14 (a) and Figure 4.14 (b) shows hardly any changes in the course of the plotted lines. One can see, however, that for the median runs the lines are shifted down, so that some improvement is indeed obtained from considering the median over the worst run. The most notable advantage of our ILS can be seen on instances from $\mathcal{N}_4$ where we observe the best solution ratios across all benchmark sets. On more than 79 % of those instances, all five ILS runs outperform Gurobi. This number increases to over 84 % when we compare the median run to Gurobi. We can also see that the tail of instances from $\mathcal{N}_5$ we have seen in Figure 4.10 disappeared, so the ILS is able to help out the standard NCC algorithm.

To take another look at the improvement the ILS brings over our standard NCC algorithm, we compare the solution ratios in relation to the MILP in Figure 4.15. Separated by benchmark set, each data point corresponds to one instance: On the x-axis we display the solution ratios from Figure 4.14. On the y-axis the ratios of the standard NCC algorithm to the MILP are depicted. Values above the diagonal represent instances on which the escaping strategies yields better solutions than NCC without escaping. The annotations in the small tables on the bottom-right of each block count the instances according to the solution ratios: In the first three quadrants, we see the number of instances, on which our ILS finds better solutions than our NCC algorithm. The first quadrant stands for instances with NCC ratio and ILS ratio at least 1. The second quadrant counts instances where the ILS ratio is less than 1 but the NCC ratio is at least 1. The third quadrant represents instances for which both ratios are less than 1. Thus, quadrants 1 to 3 correspond to the quadrants in the plots marked by the red lines. The numbers in the fourth quadrant are the number of instances for which the ILS does not yield an improvement. These numbers correspond to the numbers in Table 4.8 and the data points corresponding to those instance lie on the diagonal.

Again we see that, in particular on the bigger instances corresponding to $\mathcal{N}_3$ through $\mathcal{N}_5$, our ILS is able to improve the NCC solutions on a vast majority of the instances. While on $\mathcal{N}_3$, it is not able to surpass the MILP on many of those instances, the results on $\mathcal{N}_5$ and (even more so) on $\mathcal{N}_4$ are compelling: On nearly one fourth ($\mathcal{N}_5$) and one third ($\mathcal{N}_4$) of all instances, the median run of our ILS outperforms the MILP when our NCC could not. Additionally, in those cases when our ILS cannot outperform the MILP except one, it is within 1 % of the MILP solution quality—despite the shortcoming in the implementation we mentioned in Section 4.7.2.

A total of 24 data points (5 from $\mathcal{N}_1$, 1 from $\mathcal{N}_4$, and 18 from $\mathcal{N}_5$) are not depicted in Figure 4.15, all of which are due to the fact that the standard NCC solution yields a ratio bigger than the imposed upper limit of the ordinate. In all those instances the ILS provides strictly better solutions in its worst run than the standard algorithm. For the instances from $\mathcal{N}_1$, the worst of five ILS solutions per instance brings the ratio down to values between 1 and 1.0304. In the single instance from $\mathcal{N}_4$ the ILS is even able to outperform the MILP in all five runs. All standard NCC ratios for the instances from $\mathcal{N}_5$ but one are between 1.0644 and 1.1707 with corresponding "worst-case" ILS ratios between 0.9944 and 1.0362 (six of which are smaller than 1). The single remaining point has an NCC ratio of 1.9937, which reduces to 1.0014 (in the worst run) after applying escaping strategies.

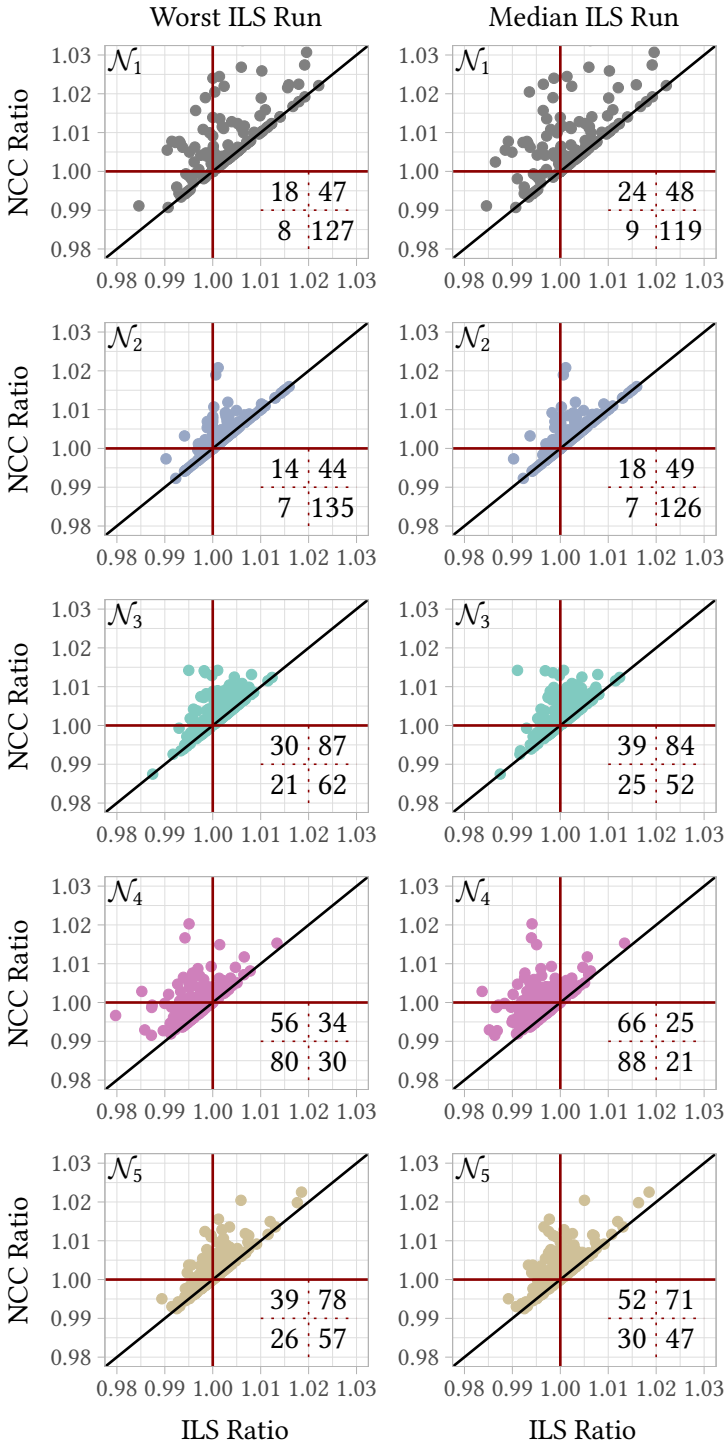


Figure 4.15: Ratios of best solutions computed by NCC algorithms to best solutions found by Gurobi separated by benchmark sets. Ratios on the x-axis show solutions from the ILS (worst of five runs in the left column, median run on the right) and ratios on the y-axis show solutions from our standard NCC algorithm. The numbers on the bottom-left of each figure count the instances according to the solution ratios; the exact definition can be found in the main text.

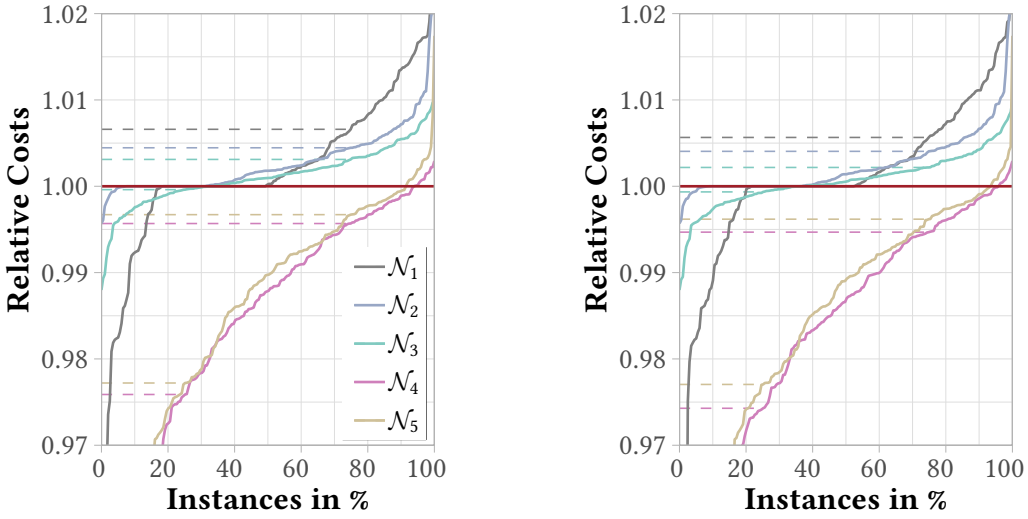
The last instance is an interesting outlier, since it reveals possible situations in which the standard NCC algorithm may have problems: In that instance the total substation capacity is very close to the number of turbines. Visual inspection of the cable layout computed by the standard NCC algorithm shows that it consists of stars centered at the substations. Due to the tight capacity, some turbines are connected to distant substations via very long edges. Our cycle detection delivers such a long edge and its reverse as well as two positive cycles at either end as a negative walk. Canceling these cycles does not help. What does help, however, is our escaping strategy *Deal with Bonbons*. It can reroute the flow from these long edges and enables the normal cycle canceling to resume its work. As such, it may be wise to use that escaping strategy by default: It helps in situations with tight capacities and it maintains the sense of a local search algorithm that only accepts better solutions.

Interestingly, we will see this instance again in Section 4.9.4 when we talk about reactive power injection.

4.7.4 Comparing the ILS to Simulated Annealing

As with the standard NCC algorithm, we also want to compare our ILS to the Simulated Annealing approach from [LRWW17]. To this end, we again select 200 instances from each of the five benchmark sets. As before, we perform five runs of our ILS per instance for 15 minutes each with different random seeds. We also run the Simulated Annealing approach on each instance with a maximum running of one hour in its best setting according to [LRWW17].

In Figure 4.16 we show the results from those simulations. As before, we record the best solution from each of the five runs of our ILS and identify the worst and median of those five solutions. We divide these solution values by the best solution found by the Simulated Annealing approach. The resulting ratios are ordered increasingly and displayed on the ordinate. Again, values smaller than 1 stand for instances on which the NCC algorithm with escaping performed better than Simulated Annealing in all (respectively in three out of) five of its runs. It stands out that the benchmark sets separate in two categories: those with few edges ($\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$) and those with many edges ($\mathcal{N}_4$ and $\mathcal{N}_5$). On the smaller instances, Simulated Annealing performs better than our ILS: On $\mathcal{N}_1$, Simulated Annealing finds better solutions on 47.5 % of the instances when considering the worst ILS solution. On $\mathcal{N}_2$ and $\mathcal{N}_3$ this holds on 63 % and 62 %, respectively. There are a total of 122 out of 600 instances in those benchmark sets on which both approaches yield equal solutions. Even though Simulated Annealing performs better on instances from the three benchmark sets with smaller instances, in all but four instances our worst ILS solutions are at most 2 % worse than the solutions provided by Simulated Annealing. These numbers do not change when we consider the median ILS runs. We suspect that this can be mostly attributed to the fact that escaping often does not improve the NCC solutions on these small benchmark instances. At the other end of the solution ratios for the worst ILS runs, there are also five instances from $\mathcal{N}_1$ with ratios between 0.8984 and 0.9689. The smallest ratio here is a pretty remarkable improvement the ILS yields over Simulated Annealing. Looking at the benchmark sets whose instances have many edges, we see that the NCC algorithm with



(a) Worst of five ILS runs

(b) Median of five ILS runs

Figure 4.16: Comparison of ILS to Simulated Annealing on 200 instances per benchmark set. For each instance, the ILS is invoked five times. The median and worst solution values out of these five runs is recorded for each instance and set in relation to the best solution value found by Simulated Annealing. These ratios are shown in increasing order along the ordinate. The dashed lines represent the 25 % and 75 % quantiles of the instances.

escaping strategies outperforms Simulated Annealing. In both the worst and the median runs, it provides better solutions on 95.5 % ($\mathcal{N}_4$) and 93 % ($\mathcal{N}_5$) of the selected instances. On more than half of these instances, our algorithm is better than Simulated Annealing by a margin of 1 %. On between 15.5 % and 18.6 % this margin is even bigger than 3 % across both benchmark sets and ILS variants (not depicted in Figure 4.16).

We conclude that the answer to the question whether NCC with escaping or Simulated Annealing performs better heavily depends on the size of the input instance: On smaller instances the "worst-case" of our ILS cannot quite keep up with Simulated Annealing (and so did the standard NCC algorithm, cf. Figure 4.12 (b)), probably due to not being able to improve the NCC solutions. On larger instances, our ILS is able to extend the lead it already had over Simulated Annealing.

4.7.5 Lessons Learned

We have seen in the evaluation that on small instances our ILS is not able to improve the best NCC solution on roughly two out of three instances, possibly due to a lack of variety in the application of each escaping strategy. In those cases, when the ILS finds better solutions, it is more likely to remain behind its competitors in solution quality. Thus, on the small instances it may be more advisable to use our NCC algorithm instead of the ILS. On the bigger instances, we see that our

ILS is able to greatly improve the NCC solutions and that it needs not much more time, since the allotted 15 minutes are hardly needed. Here, it is recommendable to use our ILS over NCC and its competitors, in particular on the biggest instances from $\mathcal{N}_4$, unless short running times is essential, in which case our NCC algorithm should be preferred. Furthermore, after addressing the aforementioned shortcomings, our ILS should be able to extend its lead even further.

4.8 Case Study: Hornsea One

In the previous evaluation we have seen that the ILS using the NCC algorithm yields very good solutions compared to two other approaches from the literature. These evaluations, however, make use of synthetic benchmark instances. We want to see how our algorithms work on real-world data. To this goal we do a case study on the formerly largest offshore wind farm in the world: Hornsea One off the coast of Great Britain [Zia19]. This section is based on joint work with Dorothea Wagner and Matthias Wolf [GWW20].

We extract the geographical coordinates of the wind turbines and substations of Hornsea One from Open Street Map [OSM20]. From there, we create a complete graph (except for edges between pairs of substations as stated in Section 4.1). This results in a graph with 174 turbines, three substations and a total of 15573 edges. In reference to the benchmark sets from [LRWW17], such an instance would be one the largest instances from $\mathcal{N}_5$. For simplicity we use Euclidean distances for the edges instead of orthodromic distances. We continue to use cable types as specified in Table 4.1, which are based on a planned real-world wind farm [DO11, BVMO11].

The real-world cable layout can be seen in Figure 4.17 (a). It appears that only one cable type is used. By coincidence, any real-world cable connects at most five turbines, which fits perfectly to the smallest cable type from the evaluation.

As before, we give Gurobi one day of maximum running time. Our ILS including all three escaping strategies (with equal weight) runs for 15 minutes each in five runs with different random seeds. In Table 4.9 we show the ratios of solution values from each of the five ILS runs and from our standard NCC algorithm to the MILP solution. As before, ratios below 1 indicate that our framework finds a better solution than Gurobi.

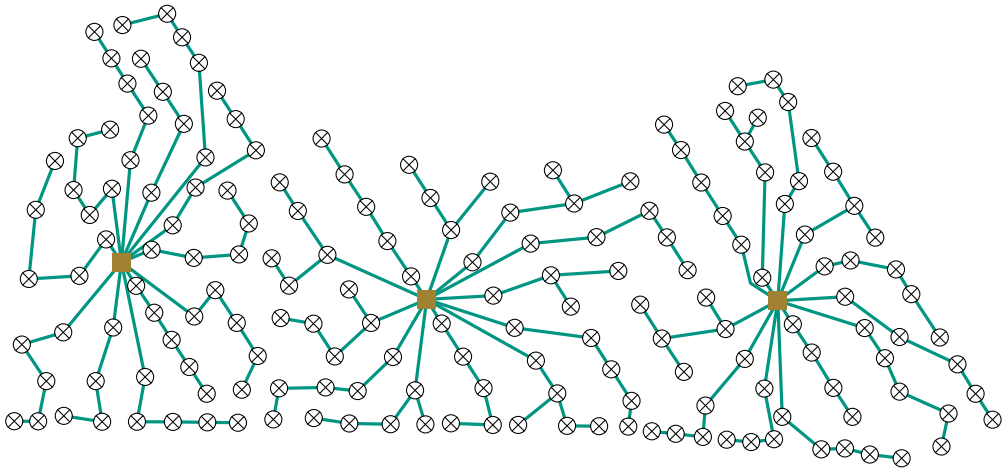
Table 4.9: Ratios of Best Solutions of ILS and NCC algorithm to Gurobi on Hornsea One

Run 1	Run 2	Run 3	Run 4	Run 5	NCC
0.9966	0.9909	1.0015	0.9967	1.0013	1.0314

Our NCC algorithm terminates in 88 seconds, which is in line with previous running times, but is approximately 3 % worse than the MILP. Our ILS is, as before, able to greatly improve upon the NCC solution: In three of five runs does it find a better solution than Gurobi. In the worst of the five runs, the solution given by the NCC algorithm with escaping is less than 0.2 % worse than the solution from the MILP. The best solution across five runs is nearly 1 % better than the solution computed by Gurobi.

For the sake of completeness, we mention an approximation of the real costs. In our setting, with the edge lengths as defined before and the cable types given in Table 4.1, the cost of the real-world cable layout is 79.061[6]. The MILP solution, in comparison, costs 64.3724 and the worst ILS run (run 3) costs 64.4693.[7] We do not go further stating a number on potential savings, since this would rather advocate for using more cable types and will most likely ignore certain design decisions taken by the wind farm planners.

Instead, we want to visually compare the different cable layouts to see if we can learn more by that: One by our standard NCC algorithm, one by our ILS, one by Gurobi and the real-world internal cable layout obtained from [4C20]. For our ILS we choose the cable layout from run 3, which is the worst of all runs—0.15 % worse than the MILP solution. From top to bottom in Figure 4.17 we show the real-world cable layout and the cable layouts computed by the NCC algorithm, by the ILS, and by Gurobi. We color-code the four cables types according to increasing capacity (green, orange, red, and black).



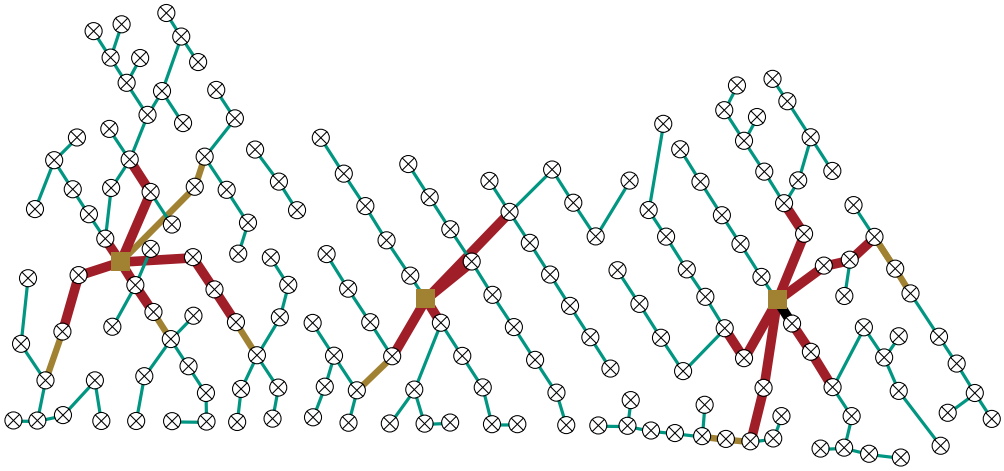
(a) Real-world cable layout adapted to the cable types from the evaluation

Figure 4.17: Visualization of four internal cable layouts for the Hornsea One wind farm[8]

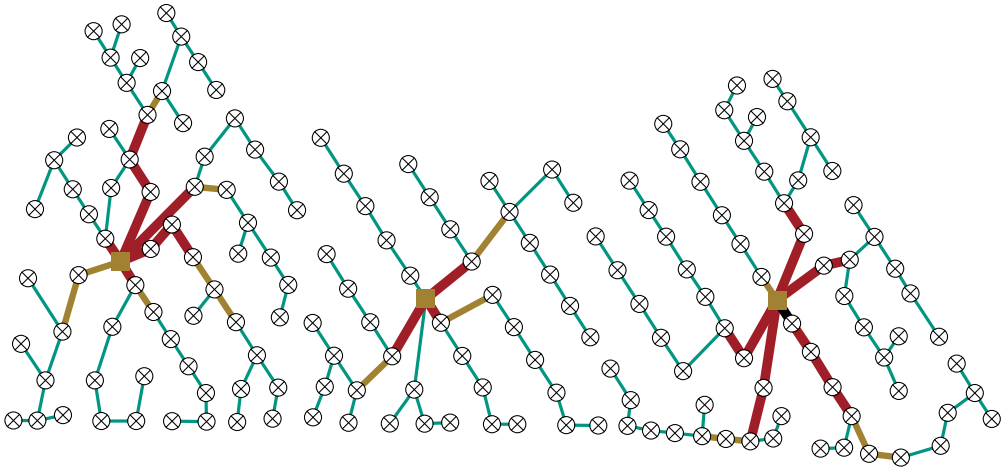
[6] With a certain unit which we shall not specify further—but which is also not important.

[7] For the sake of completeness, the full list of best solutions found by algorithm is: NCC: 66.3944, MILP: 64.3724, ILS runs 1–5: 64.1514, 63.7847, 64.4693, 64.1623, and 64.454.

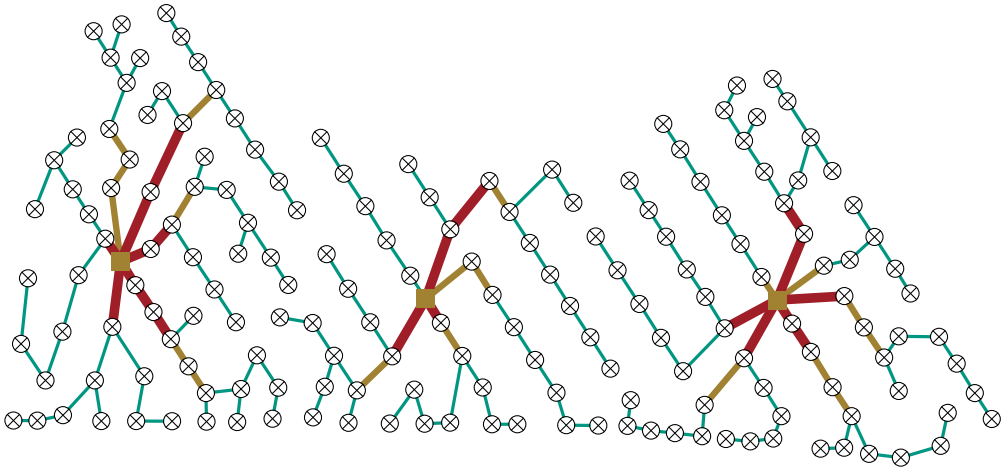
[8] We should point out that there is some degree of distortion in the edge lengths depicted here. Consider, for example, the turbines at 7 and 8 o'clock from the right-most substation in Figure 4.17 (d), each connected by a red cable to the substation: The turbine at 8 o'clock appears closer to the substation than the turbine at 7 o'clock. The opposite is true, however, when we look at the edge lengths in the graph used for the computations. The reader may, just from looking at the figure, understandibly but wrongly assume that in Figures 4.17 (b) and 4.17 (c) those two turbines and the substation form a negative cycle in the residual graph for $\Delta = 11$.



(b) Cable layout computed by our standard NCC algorithm



(c) Cable layout computed by our ILS in Run 3



(d) Cable layout computed by Gurobi

Figure 4.17: Visualization of four internal cable layouts for the Hornsea One wind farm

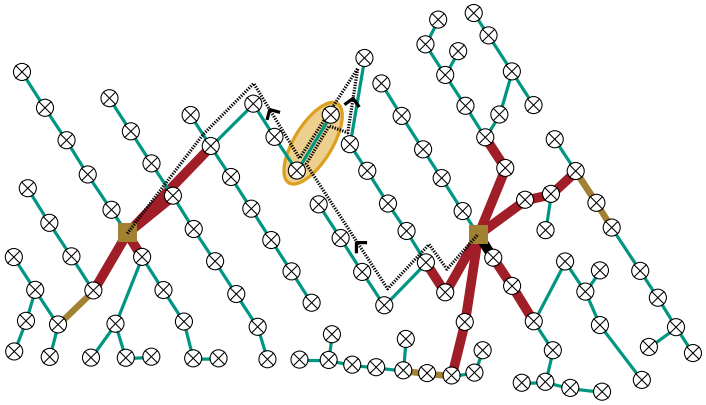


Figure 4.18: A bonbon in the cable layout for the Hornsea One wind farm computed by our standard NCC algorithm. It consists of three cycles: A positive cycle consisting of 13 edges, including edges to and from the super substation, a positive cycle consisting of three edges, and a negative cycle consisting of two edges, marked with an ellipse in the background. The bonbon corresponds to a negative closed walk in the graph L (constructed as a subgraph of $\mathcal{L}(R)$) for $\Delta = 1$. However, none of the three cycles can be canceled in order to improve the solution quality.

It stands out that all algorithmic approaches yield cablings with large proportions of radial layouts. Turbine generation tends to be collected from the radial layout bits to make use of cable types with higher capacity the closer the nearest substation gets. From looking at different solutions it seems hard to identify parts of a solution which are particularly good compared to other solutions. There seem to be, however, some parts in particular near the boundary of the wind farm, where multiple solutions coincide. It could prove helpful to the algorithmic approaches to fix those parts in the solutions to some extent to facilitate finding the best solutions possible in the given running time.

As a side-note, let us look again at the standard NCC layout: This layout includes a *bonbon* for $\Delta = 1$ as visualized in Figure 4.18. This bonbon is identified by traversing the parent pointers from the Bellman-Ford algorithm as explained in Sections 4.4.3 and 4.4.4. The two turbines marked by the ellipse constitute a short negative cycle. The negative residual cost on the edge between those two turbines directed upwards, which we denote by e, is propagated along the cycle of three edges, back along $\bar{e}$, and then along the cycle of 13 edges, on which it passes the super substation. Overall, the negative residual costs from e and from two more edges, one on either long cycle, outweigh the additional costs. However, if $\Delta = 1$ units of flow were sent along the bonbon, the actual cost of the flow would increase, since the residual costs on e do not count.

In summary, our case study supports the findings of the evaluation on the synthetic instances that our ILS computes solutions of very similar values compared to other approaches. An essential part of the applicability of our cable layouts to real-world wind farms, however, lies not only in the solution values but also in its behavior under electrical aspects.

4.9 Analysis of Layouts under Electrical Aspects

In the experimental evaluations in Sections 4.5 and 4.7 we have seen that our NCC algorithm with or without escaping strategies finds competitive solutions compared to an MILP formulation solved by Gurobi and a Simulated-Annealing-based algorithm. A wind farm planner may therefore be tempted to realise cable layouts computed by our algorithms in their wind farms, in particular since the speed of our standard NCC algorithm allows the computation of good layouts for various slightly different turbine placements in a reasonable amount of time. The underlying optimization problem as given in Section 4.1 and our NCC algorithm, however, do not make any promises about possible desired properties (like absence of cycles) or about the real-world performance in transmitting the turbine generation to the substations. Possible questions a wind farm planner may be interested in are "Does a cable layout include cable crossings?" or "Is the amount of power losses during transmission within reasonable boundaries?".

To shed light upon such questions, we present a workflow to evaluate cable layouts under electrical aspects. This results in a new coupling for graph algorithms for WCP and power system analysis. In short, cable layouts from cable layout optimization algorithms such as our NCC algorithm are converted to power flow models, which are simulated in the Energy Systems Analysis, Simulation, Modeling, Optimization and Visualization (eASiMOV) software framework [KÇKH17, ÇKKH18]. The evaluation of the power flow simulations under electrical metrics show that the cable layouts perform very well under electrical aspects on a vast majority of input instances. For the remaining minority we are able to identify structures in the solutions that result in a worse performance. These observations can be used as possible directions for future improvements to the algorithms. This section is based on joint work with Hüseyin Çakmak, Pascal Mehnert, Torsten Ueckerdt and Veit Hagenmeyer [Gri+21].

The workflow is presented on a high-level in Section 4.9.1. We explain the structure of our power flow models in Section 4.9.2 and explain how we obtain them from the abstract algorithm in- and output in Section 4.9.3. We analyse the cable layouts and the resulting power flow models with respect to various structural and electrical aspects in Section 4.9.4 and conclude with lessons to be learned from the analysis in Section 4.9.5.

4.9.1 Workflow in a Nutshell

An overview of the proposed workflow is shown in Figure 4.19 and explained in the following.

The workflow invokes the NCC algorithm or an MILP to compute an optimized cable layout (Step **1**) or it uses a precomputed cable layout in GraphML-format [Bra+02]. In Step **2**, the cable layout is converted into a power flow model that can be processed by the eASiMOV. This framework uses the open-source simulation package MATPOWER [ZMT11] to simulate the power flow models (Step **3**). In Step **4**, a range of metrics are obtained from the simulation results.

While there are more powerful models to simulate power systems, we believe the power flow model is a most suitable link between the complexity of power systems in real-world wind farms and the simplifying network flow model in WCP.

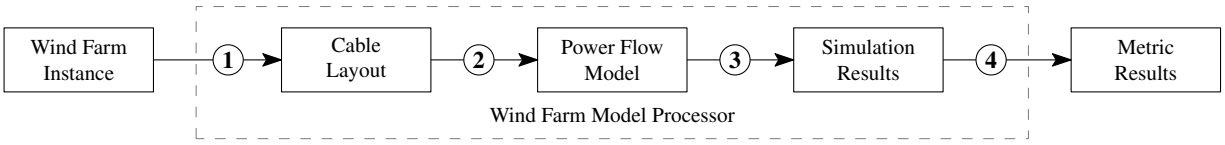


Figure 4.19: Overview of intermediate steps within the Wind Farm Model Processor

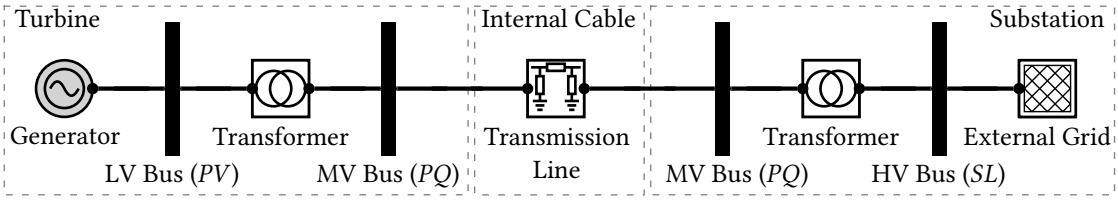


Figure 4.20: Schematic representation of the components of a power flow model for wind farms

The whole workflow is realized by a suite of Python3-scripts referred to as Wind Farm Model Processor. Multiple input instances can be processed in a single invocation. A command-line interface (CLI) provides fine-granulated control over all processing steps and facilitates extensions to all functionality that is provided in the Python-APIs of external software (or the CLI provided by the C++-code for the NCC algorithm).

4.9.2 General Structure of the Power Flow Models

In this section, we explain what a power flow model in our workflow looks like and what technical parameters need to be specified for all the components. To the contrary, Section 4.9.3 specifies how we populate these values—as far as possible from the algorithm output.

Figure 4.20 shows a wind farm as a power flow model with all electrical components. Turbines are modelled as a generator that is connected via a Low-Voltage (LV) generator *(PV)* bus and a step-up transformer to a Medium-Voltage (MV) load *(PQ)* bus. For the generator, the rated power and active power injection must be specified. The voltages at the buses correspond to the operating voltages of the generator and the internal cabling, respectively.

Substations are modelled as an MV load *(PQ)* bus connected to the internal cabling, a transformer and a High-Voltage (HV) slack bus. The voltage of the MV bus coincides with the voltage of the internal cabling and the voltage of the HV buses can be set independently. The transformers in turbines and substations are defined by rated power, short-circuit impedance, and X/R ratio. Note that the HV slack buses can also be replaced by load buses connected via external cables to a single common HV slack bus with an external grid.

Internal cables are modelled by a single π transmission line each that connect two MV buses. For the cables, resistance, reactance, and capacitance must be specified, all of which can be set as a per-unit-of-length value, as well as rated apparent power and the length of the line.

4.9.3 From Algorithm Output to Power Flow Models

For the electrical analysis of output from our NCC algorithms and the MILP, the computed solutions need to converted into power flow models as outlined in Figure 4.19 Step **2**. As seen in Section 4.1, not much of the necessary pieces of information are given by the underlying instance or a solution. Most notably, the turbine generation is given as a virtual unit and both substation and cable capacities follow this unit. In an ideal setting, for example if a wind farm planner decides to use our NCC algorithm, it is known what kind of turbines, substations and cables are available and the necessary parameters are specified by the manufacturer. In our non-ideal setting, we only know the cable capacities (in abstract units of turbine generation) and nonetheless need to populate all parameters.

As a rough outline, we achieve that goal as follows: Starting with a suite of real-world cables with different cross-sections, we compute for each cable the apparent power per turbine if this cable were used at full capacity as each of the four input cable types from Table 4.1. With this, we are able to choose a suitable power rating for our turbines and choose four cables that fit to our four cable types. The remaining electrical parameters are then given by the manufacturer's specification. Similarly, we find a real-world example of a turbine with the chosen power rating and can then look up the parameters. For the substation, the rated apparent power is computed from the rated power of the generator in a turbine. As a final step, we need to copy the cable layout itself. In this step, we infer the lengths of transmission lines from the positions of turbines and substations, which are given on an arbitrary scale.

Recall that the cables types used in the evaluation stem from a proposed wind farm operating with an internal cabling at a voltage of 33 kV [DO11, BVMO11]. We use this voltage for our wind farm as well. As a suite of real-world cables, we choose a selection of 3-core, XLPE-insulated submarine cables manufactured by Nexans [Nex08] with different cross-sectional areas and their rated currents as seen in Table 4.10.

By means of Equation (4.19) we compute the highest apparent power per turbine S_{Trb} that is allowable under the rated current of the conductor for any particular cable type κ if the cable type was used to full capacity $\mathrm{cap}_{\mathrm{K}}(\kappa)$ according to Table 4.1.

$$S_{\mathrm{Trb}} = \sqrt{3} \cdot 33\,\mathrm{kV} \cdot {}^{\mathrm{RatedCurrent}}/_{\mathrm{cap}_{\mathrm{K}}(\kappa)} \tag{4.19}$$

The results are shown in Table 4.10. Consider, for example the conductor with a cross-section of 95 mm^2 in the first row. If such a cable were used on a transmission line to which five turbines are connected, then the rated current of 291 A yields that a maximum of 3.33 MVA apparent power for the five turbines would be admissible for safe operation. In the last column, corresponding to the biggest of the four input cable types, we see that the highest admissible apparent power for a turbine is at 3.25 MVA if 15 turbines were connected. Thus, we choose 3 MW as the power rating of the generators, even though nowadays much bigger power ratings are being used (in 2018, the average rated capacity of an offshore wind turbine in Europe was 6.3 MW, up from 3.7 MW in 2015 [Tel+20]). With the rated apparent power per turbine computed from the biggest cable type, this choice yields sufficient capacity for a plausible power

Table 4.10: Apparent power per turbine for each cable type from Table 4.1 used at full capacity and conductor sizes of the Nexans cable suite [Nex08]

Cross-Section (mm^2)	Rated Current (A)	App. Power per Turbine for x Turbines (MVA)			
		5	8	12	15
95	291	3.33	2.08	1.39	1.11
120	330	3.77	2.36	1.57	1.26
185	411	4.70	2.94	1.96	1.57
240	470	5.37	3.36	2.24	1.79
400	627	7.17	4.48	2.99	2.39
500	699	7.99	4.99	3.33	2.66
630	777	8.88	5.55	3.70	2.96
800	852	9.74	6.09	4.06	3.25

factor of 0.92. For each of the four input cable type capacities, we choose the Nexans cable of smallest cross-sectional area that allow 3 MW generators. The respective cables are underlined in Table 4.10. We obtain the corresponding values for resistance, reactance, and capacitance from [Nex08].

As a wind turbine generator with 3 MW rated power we use an offshore version of the V112 by Vestas, which has a rotor diameter of 112 m. We use 690 V/33 kV step-up transformers with a rated apparent power of 3.45 MVA, i. e., 1.15 times the rated active power of the wind turbine generator. From an example offshore wind farm in DIgSILENT PowerFactory [DIg16] we obtain a short-circuit impedance of the transformer of 6 % with an X/R ratio of roughly 12 : 1.

For the offshore substations, a typical transformer with a primary voltage of 230 kV and a short-circuit impedance of 3 % is used. The impedance of the transformer is assumed to be purely reactive. Contrary to transformers at the turbines, the rated apparent power must be infered from the capacity of the substation cap_{Sub}, which is given by the input instances. The rated apparent power of a substation is calculated as $S_{\text{Sub}} = \text{cap}_{\text{Sub}} \cdot P_{\text{Trb}} \cdot (\cos\phi)^{-1}$ where P_{Trb} is the rated active power of the wind turbine generators and an assumed power factor of $\cos\phi = (1.15)^{-1} \approx 0.87$.

Thus, we have specified all electrical parameters needed in the power flow models. The power flow model itself is constructed from the computed cable layout by replacing all turbines and substations by the respective building block from Figure 4.20. Any edge with non-zero flow is replaced by a transmission line between the MV busses of the respective endvertices. Edges with zero flow are discarded. The length of all transmission lines is computed from the input instances by a scaling factor such that the minimum turbine distance is 700 m, i. e., 6.25 times the rotor diameter. With that, Step **2** in Figure 4.19 is complete and the power flow models can now be simulated to obtain the desired performance metrics.

4.9.4 Analysis of Algorithm Output

All analyses in this section use the cable layouts of the 1000 benchmark instances randomly selected for the comparison of the MILP and the NCC algorithm with and without neighborhood heuristics in Sections 4.7.2 and 4.7.3. In the following, we mainly focus our analysis on the cable layouts computed from the MILP and the NCC algorithm without neighborhood heuristics (refered to simply as the NCC algorithm).

The analysis is structured as follows: We start with structural observations from the layouts, looking in particular at additional constraints to WCP from the literature: the enforcement of tree-structures and avoidance of cable crossings. Using the simulation results, we continue by looking at average and maximum line loadings, which serves in part as a validation that the electrical parameters from Section 4.9.3 have been chosen sensibly. The remainder of the analysis is dedicated to active power losses, reactive power injection and voltage stability. Whenever possible, we draw comparisons to reference values from the literature. Outliers are inspected more thoroughly to investigate the reasons for the deviations, so that recommendations for future considerations by algorithm engineers working on WCP can be drawn.

The underlying data can be found in [Çak+23].

Structural Observations

The first set of analyses concerns the structural properties of the graphs induced by edges with non-zero flow in the solutions. That means we only look at the edges on which cables are indeed installed.

We start by comparing the usage of the different non-trivial cables types. Table 4.11 shows the average number of times each cable type is used across all 1000 instances.

Table 4.11: Number of cables using each cable type averaged over all 1000 cable layouts obtained from NCC and MILP

Cable Type	(5, 20)	(8, 25)	(12, 27)	(15, 41)	Total
NCC	114.1	5.10	5.42	0.077	124.697
MILP	114	4.93	5.68	0.033	124.643

For both approaches, a vast majority of cables is chosen from the smallest cable type. This is not surprising as all turbines need to be connected and a capacity of 5 is sufficient to collect large parts of the outer areas of the wind farms. Both algorithms also use the bigger cable types but to a much smaller extent and without any readily meaningful difference. It stands out, however, that the biggest cable type is used more than twice as often by the NCC algorithm. One could think that turbine generations are streamlined more by the NCC algorithm due to the cycle cancelations but this is highly speculative. Further analysis would be needed and the decreased usage of the second biggest cable type in the NCC algorithm might already be

evidence to the contrary. The total across all cable types suggests that the NCC algorithm uses more cables than the MILP. Reasons could be that the MILP uses more connected components in the graphs (which would be unproblematic) or that there are (more) cycles in the NCC cable layouts (which may be problematic for electrical reasons). We look into this next.

By means of a simple graph search, we can determine whether there are cycles in the cable layout. We find that 18 out of 1000 cable layouts computed by our NCC algorithm and six out of 1000 computed by the MILP contain cycles. The underlying instances are all from the benchmark sets $\mathcal{N}_1$ and $\mathcal{N}_4$ (NCC: 14 from $\mathcal{N}_1$, four from $\mathcal{N}_4$; MILP: four from $\mathcal{N}_1$, two from $\mathcal{N}_4$). All but one of the instances for which the MILP computes a cable layout with a cycle also have a cycle in the NCC cable layout. A tendency can be observed that the instances yielding cycles are among the biggest in their respective set of benchmark instances. These observations suggest that cycles may be a result of the dimensioning of different parameters: Generation from many different turbines accrues at a particular turbine, from which no cable type may have sufficient capacity or it is cheaper to split the flow along different paths. We have seen such a situation in the NCC and ILS cable layouts for the Hornsea One wind farm (Figure 4.17). It may be interesting to investigate whether cycles also occur with other cable types or with a higher number of substations such that turbine generation cannot "pile up". It should be noted that simply increasing the capacity of the largest cable type does not solve the problem: If the biggest cable type is comparatively expensive, flow will already split when the capacity of smaller cable types is reached.

We take a quick look at how the number of cycles changes after applying the escaping strategies of our ILS. To keep it quick and simple, we only look at the results of one of the five ILS runs. Here, the number of instances with cycles reduces to nine from $\mathcal{N}_1$ and one from $\mathcal{N}_4$, all of which also had cycles in the NCC solution. Our ILS has been able to improve the NCC solution in all but one of those instances. Thus, our escaping strategies seem to help to some extent to avoid cycles, even though there are not specifically designed to do so.

Cycles are indeed forbidden in various works in the literature on wind farm cable layout optimization, for example in [BVMO11, FP18]. Our numbers, however, point out that cycles occur indeed only rarely, which opens the possibility that this constraint may be relaxed to some extent for the sake of faster or simpler algorithms.

Another constraint is the explicit ban of cable crossings (e. g. [FP18]) due to higher maintenance cost [BHMP00]. Using a geometric argument, we can determine whether a cable layout includes crossings: For any pair of non-adjacent edges, check whether the straight line through the endvertices of one edge separates the endvertices of the other edge and vice versa. Out of the 1000 instances, NCC algorithm computes cable layouts with crossings on 78 instances ($\mathcal{N}_1$: 21, $\mathcal{N}_2$: 1, $\mathcal{N}_3$: 2, $\mathcal{N}_4$: 27, $\mathcal{N}_5$: 27). The MILP yields crossings on 20 instances ($\mathcal{N}_1$: 5, $\mathcal{N}_2$: 4, $\mathcal{N}_3$: 4, $\mathcal{N}_4$: 1, $\mathcal{N}_5$: 6). It is surprising that there is no clear cut on from which benchmark sets the instances with crossings come. NCC struggles with the biggest and the complete instances, the MILP does not seem to have problems with the biggest instances. It is clear though that there is a noticeable increase in numbers compared to counting the number of solutions with cycles.

Given the importance of avoiding cable crossings according to the literature, it might be worth to investigate if they can be incorporated into the flow model in WCP. Another remedy might lie in extending the Iterated Local Search. In our case study, the NCC cable layout (Figure 4.17 (b)) includes two crossings, both of which are not present any more in the depicted ILS layout (Figure 4.17 (c)). In fact, in each of the five ILS runs mentioned in Section 4.7.3, the number of solutions with crossings is at least halved—without any explicit avoidance of cable crossings. We look again at the same ILS run as in our investigation on cycles: A total of 39 cable layouts include crossings. Separated by benchmark sets, the distribution is $\mathcal{N}_1$: 13, $\mathcal{N}_2$: 1, $\mathcal{N}_3$: 2, $\mathcal{N}_4$: 16, $\mathcal{N}_5$: 7. This is a remarkable improvement over the numbers for the standard NCC cable layout, in particular on the instances from $\mathcal{N}_5$. A total of seven cable layouts have more than one crossing, with as much as five crossings in one layout. In all but one of those, there is one edge that is involved in all pairs of crossing edges. These edges could be considered as the point of attack for an escaping strategy with the sole purpose of resolving crossings in order to lazily[9] enforce a non-crossing constraint.

In conclusion of our analysis on structural properties of the cable layouts computed by the NCC algorithm and the MILP, we have seen that cycles in the cable layout occur only rarely. We have also seen that cable crossings occur more often, in particular in layouts from our NCC algorithm, but that incorporating NCC into an ILS already brings the number of crossings down (in addition to improving the costs of the solutions as well).

Average and Maximum Line Loading

After the structural observations we look into the results of the power flow simulations, starting with line loading. The loading of a transmission line is defined as the ratio of the current on that line according to the simulation to the rated current of that line.

The average line loadings, i. e., the average of the loadings across all edges with non-zero flow within an instance, are between 34 % and 62 % for the MILP and between 18 % and 61 % for NCC across all instances, with about 98 % of all cables between 40 % and 60 %. These small percentages arguably arise from a big amount of cables from the smallest type which connect the outer turbines. The mean average line loading across all instances is 0.58 percentage points smaller for the NCC algorithm than for the MILP. The difference seems neglectibly small, in particular since for the NCC, the instance with 18 % average line loading is a single outlier. In this instance, no optimization took place yielding an NCC ratio of 1.9937 as adressed in Section 4.7.2.

An overwhelming majority of instances (934 for NCC and 966 for MILP) show a maximum line loading, i. e., the highest line loading on any edge within an instance, between 90 % and 91 %. In these cases, at least one cable is fully saturated to its capacity in the network flow. Twelve instances for NCC and eleven instances for MILP have a smaller maximum loading. The maximum loading of eleven cable layouts from NCC and four from MILP exceeds 95 % (maximum values of 153 % and 107 %, respectively). Those extremely high loadings are worrisome.

[9] In this context, "lazily" should be understood as "use such an escaping strategy only if the cable layout does have crossing cables."

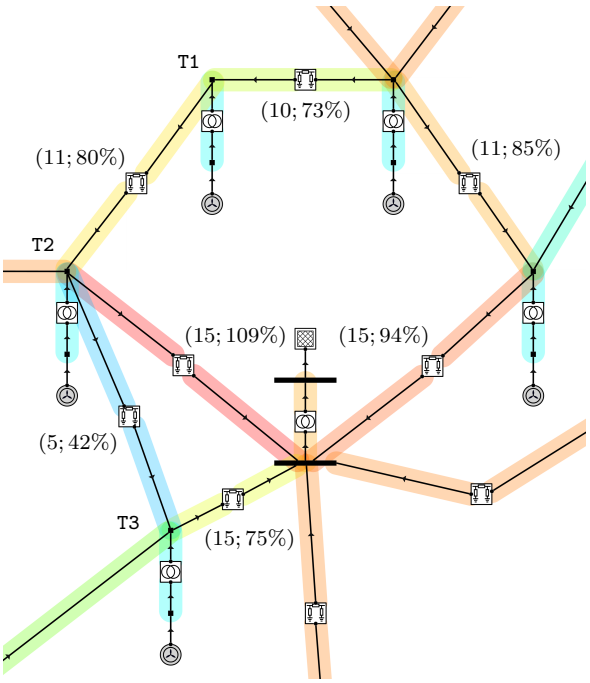


Figure 4.21: A cycle in a power flow model causes an overload. Annotations to lines show the absolute loading in the network flow model in units of flow and the line loading in the power flow model as percentage values. The latter is also color-coded. The arrows indicate the direction of active power flow.

Further inspection of those cable layouts reveals that all share one of two properties: The cable layout includes a cycle or a turbine has paths to different substations. In both scenarios, there is a turbine where the incoming flow is split up. With the graph-theoretic flow, any split is allowed. This does not hold for electrical flow. Thus, electrical flow and network flow can take different values, which may result in the observed overloads. However, not all layouts with cycles or connected substations yield excessive line loadings: Only for about half of those instances (9 out of 18 with cycles, and 2 out of 3 with connected substations) does the maximum line loading exceed the aforementioned threshold of 95 %. Those scenarios are included in the recommentation to the algorithm engineers in Section 4.9.5.

Figure 4.21 shows an example of an overload in a cycle. On each transmission line, a tuple states the absolute loading in the graph-theoretic flow model and the relative loading in the power flow model. From turbine T2, 20 units of flow are split up between the connection to turbine T3 (five units on a cable with a capacity of 5) and the connection to the substation (15 units on a cable with a capacity of 15). In the power flow model, this split is not maintained. Instead, more current flows on the transmission line incident to the substation. This causes an overload, while the graph-theoretic flow respects all cable capacities. Nonetheless, as seen

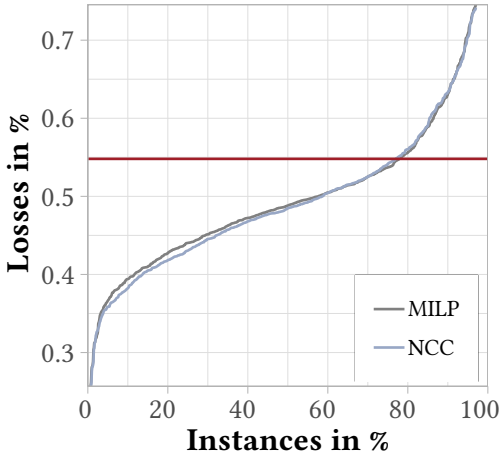


Figure 4.22: Comparison of total losses in NCC and MILP generated cable layouts. For each algorithm, the loss ratios of all instances are sorted ascendingly. Loss ratios are computed as the difference between active power injected at transformers in turbines and active power received at external grids in substations, normalized by the total active power injection at transformers in turbines. The horizontal line gives a possible reference value of 0.548 % [AGUZ17].

above, such overloads occur only very rarely and the presence of cycles or connected substations is not sufficient for overloads either. For a vast majority of instances both algorithms yield cable layouts with satisfactory line loading, which affirms that our procedure of determining electrical parameters is sensible as far as can be checked by the line loading.

Active Power Losses

In this section, the active power losses along the transmission lines are investigated. As the absolute losses of the transmission lines we consider the difference between the active power injected by the transformers in the turbines into the internal cabling and the active power received at the external grid in the substations. To ensure comparability across all instances, we use ratios: absolute losses divided by turbine transformer active power output. Notably, losses inside the transformers are not considered. The losses at the turbines do not depend on the layout but are a constant depending on the parameters of generators and transformers. The impedance of transformers in substations was set as purely reactive (cf. Section 4.9.3).

Figure 4.22 shows the losses in the cable layouts computed by our NCC algorithm and the MILP. For both approaches, the losses are computed for all 1000 instances and sorted increasingly. NCC and MILP produce very similar results with a mean difference of only 0.0033 percentage points in favor of NCC. The comparison to a reference value is more helpful: The horizontal line at $y = 0.548$ % shows a reference value from the literature for losses in a case study on an offshore wind farm with 95 turbines at a rated power of 4 MW with a tree-like cable layout [AGUZ17].

Table 4.12: Distribution of power factors at substations per instance

Quantiles	max	median	75 %	90 %	min
Smallest power factor per instance					
NCC	0.9951	0.9879	0.9872	0.9862	0.9734
MILP	0.9927	0.9879	0.9871	0.9864	0.9840
Average power factor per instance					
NCC	0.9952	0.9893	0.9886	0.9876	0.9840
MILP	0.9927	0.9892	0.9886	0.9876	0.9840

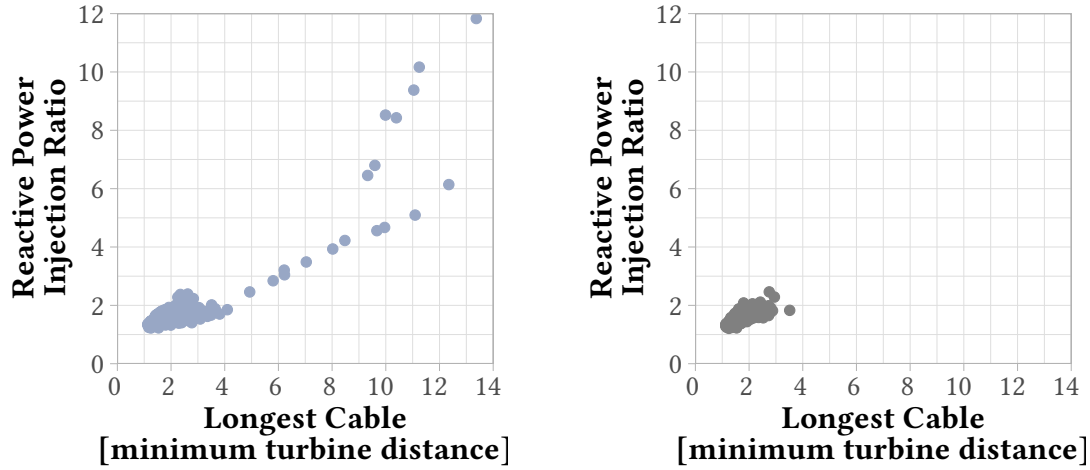
In comparison, NCC and MILP provide cable layouts with fewer losses on approximately 77 % of all instances. Both algorithms yield a loss ratio of less than 0.75 on 97 % of all instances. The maximum loss ratios are 0.98 % for both MILP and NCC.

Thus, the optimization algorithms perform similarly with respect to active power losses and show a reasonable performance compared to a case study from the literature.

Reactive Power Injection

There are two main factors influencing the reactive power injection in the power flow models: transmission lines and transformers. We measure reactive power injection by the power factors at the external grid connected to each substation. For each algorithm and each instance we obtain the smallest and average power factor among the substations and report those results in Table 4.12. For example, the (NCC, 75 %)-value in the upper part of Table 4.12 is the biggest y (rounded to four digits) such that the power factors at all substations in 750 out of 1000 cable layouts computed by NCC are at least y. The inner quantiles for both algorithms are virtually the same and the maximum values are very close. Only the overall smallest power factors for both algorithms show a notable difference, which is investigated below. In the literature, an average power factor of 0.9983 has been reported for a 17.56 MW experimental wind farm with 24 turbines and an internal cabling at 20 kV [DCCA07]. While a direct comparison should be taken with a grain of salt due to different equipment, this reference suggests that cable layouts from NCC and MILP show sensible power factors.

Further inspection of the NCC layout with the overall smallest power factor reveals that this layout includes two connected substations and an exceptionally long cable (7.3 km, i. e., 10.4 times the minimum turbine distance) to the substation with the smallest power factor. The simulation shows that on this line a disproportionate amount of reactive power (356.4 kvar) is injected into the system, whereas the average reactive power injection on all transmission lines of this wind farm is 42.29 kvar. With this observation in mind, we computed this ratio of maximum absolute reactive power injection and average absolute reactive power injection over all transmission lines for all cable layouts and both algorithms.



(a) NCC cable layouts

(b) MILP cable layouts

Figure 4.23: Comparison of NCC and MILP cable layouts with respect to maximum cable length and highest amount of reactive power injected on transmission lines. The points (one per instance) show the length of the longest installed cable type (in multiples of the minimum turbine distance) on the abscissa and, on the ordinate, the ratio of the highest amount of reactive power injected on any line to the average amount of reactive power injection over all lines.

For each algorithm, Figure 4.23 shows 1000 points; one for each cable layout. The abscissa of a point is the length of the longest cable in that layout and the ordinate represents the aforementioned ratio. For both algorithms we see a strong concentration of cable layouts with a length factor of at most 4 and a reactive power ratio of at most 2.5. However, for the NCC algorithm we observe several outliers with a high cable length factor and a high reactive power injection on at least one transmission line. We verified on a small sample that both values originate from the same transmission line. This was expected since reactive power injection increases as line length increases. We cannot explain the two apparent tendencies within the outliers. However, to keep reactive power injection low, algorithm engineers should aim for avoiding excessively long connections. This coincides with the intuition that transmission lines in wind farms connect close-by turbines, as seen for example in our study on the Hornsea One wind farm (Section 4.8).

We take again a quick look at the cable layouts from the same ILS we have already looked at in our section on structural observations. The reactive power injection ratios are greatly reduced from the NCC cable layouts: All but one instance have a ratio below 2.5 and a length factor smaller than 3.9. The remaining instances, which has a ratio of 7.4 and a length factor of 7.8, is also the instance with the highest number of cable crossings. It may well be that this long edge is the same edge that is involved in all five crossings. Therefore, an escaping strategy with the purpose of avoiding long edges or with the purpose of removing crossings may also remove this outlier.

Voltage Stability

For an investigation of voltage stability in the cable layouts, we consider the deviation of voltage levels from their nominal values in the corresponding power flow model. We use a per-unit-measurement; the unit is the nominal voltage level at each bus. For the LV buses in the turbines and the HV buses in the substations we expect a measurement of 1 p. u. since they are generator and slack buses, respectively. Our expectation was verified by the simulations. Across all instances, the lowest voltages are 0.9946 p. u. for NCC and 0.9957 p. u. for MILP. The highest voltages are 1.019 p. u. for NCC and 1.011 p. u. for MILP. Those values are sufficiently close to 1 p. u. so that no negative impact to the complete system is to be expected after a meaningful grid is attached to the substations.

4.9.5 Lessons Learned for Algorithm Engineers

In Section 4.9.4 we have evaluated cable layouts from our NCC algorithm and the MILP with respect to structural and electrical characteristics obtained from power flow simulations in eASiMOV. We have learned that cycles do occur, albeit rarely, and that they result in unexpectedly high line loadings in approximately one in two cases. Cable crossings occur approximately four times as frequently as cycles in the cable layouts from the standard NCC algorithm. This number is already halved for cable layouts from the ILS but it may still be worthwhile to check for crossings explicitly and have a dedicated escaping strategy if one wants to avoid crossings completely. Investigating the maximum line loadings showed that cycles and connected substations can result in overloads since the split of outgoing power along two edges at a turbine need not coincide with the split of network flow as it was computed by the algorithms. An evaluation of the length of installed cables and reactive power injection showed that in some NCC cable layouts have exceptionally long cables. Not unexpectedly, those instances also showed a high reactive power injection on at least one transmission line. Algorithm engineers should therefore find ways to avoid excessively long cables. These long cables may be a target for our escaping strategy *Deal with Bonbons*, be it by implicit application of the strategy or by targeting those long cables explicitly in a similar fashion to the bonbons. Since the length of longest cables is greatly decreased for cable layouts from our ILS, a possible remedy for long cables might already have been found in one of those neighborhood heuristics.

4.10 Discussion

To close this chapter and our elaborations on the applicability of Negative Cycle Canceling to the Wind Farm Cabling Problem we discuss various aspects of the model, the algorithm and our evaluation, with a particular focus on points of emphasis we have found in other research. The reader may want to revisit our literature overview in Sections 2.1 and 2.3 before continuing.

Our evaluation is based on the synthetic benchmark instances from [LRWW17] and the Hornsea One wind farm. At the time of writing the underlying papers [Gri+18, Gri+19, GWW20], we were not aware of the benchmark instances mentioned in [FP18]. We believe that extending our evaluation on these instances may provide further insights into the performance of our algorithms.

In our wind farm model, turbines and substations have a distance given by the function len. In our evaluation, we assumed Euclidean distances based on the coordinates of arbitrary scale in the synthetic instances. There is, however, no reason why edge lengths could not represent other "distances". Edge length may incorporate other offsets, for example to account for buoyancy of cables as seen in floating wind farms [LDM21] or to represent certain trajectories of cables on the seabed in order to avoid obstacles. In the literature, Steiner points have been used to route cables around obstacles [FP18]. It seems within the realm of possibility, to precompute certain trajectories for various connections around obstacles with their real length and include them, possibly as multiedges, into the model. Still, we do not know to what extent our algorithm could deal with multiedges. In theory, there is no problem with multiedges since they would be considered as different entities in our Bellman-Ford algorithm, but in practice our algorithm might find more "bonbons" than expected.

Multiedges could also be a way to deal with over- and underground cable systems from onshore wind farms [Her+17]. Two turbines (for the sake of the example) can be connected by one edge that represents a potential underground cable with a length equal to, for example, the Euclidean distance of the two turbines, and by another edge that stands for an overground cable with a length corresponding to the road distance between the two turbines. Such an interpretation is facilitated by the fact that neither our model nor our algorithm need uniform cable types across all edges. There may well be two sets of cable types, one for overground and one for underground cables, each inducing their own cost function c_K.

These non-uniform cost functions may also prove helpful if one were to consider the expansion problem of a wind farm: Assume that a wind farm has already been built and is fully operational. Then, additional turbines are to be build around the wind farm and connected to the existing cable layout. Some cables might have to be replaced by cables of higher capacity to account for the additional load. The new cable layout should extend the previous with minimal cost for all necessary changes. An adaptation of the cost function could reflect the costs of updating the cable layout. In practice, however, it may very well be that this is a rather unrealistic scenario, since new wind farms seem to be built with a high degree of independence to existing ones nearby.

In its literal meaning, our cost function represents the installation cost of cables without considering operational cost over the lifetime of the wind farm. An adaptation of the cost function has been proposed in the literature to account for the long-term cost of power losses [FP18]. In that approach, one cable type has different costs depending on the flow it carries. This can be interpreted as having more steps in our cost function (possibly one step per unit of flow), even though the conductor is the same. As such, the proposed adaptation of the cost function is well within the capabilities of our algorithm. In fact, the comparison to its competitor, in particular the MILP might look a lot better: On the one hand, more steps yield more binary variables in the MILP formulation which might yield more time-consuming computations. On the other hand, more (or rather: smaller) steps reduce the number of negative short cycles, in particular since the losses (and therefore the costs) increase quadratically with the current [FP18]. Thus, the savings by reducing flow on a given edge by a fixed amount are outweighted by the costs for the same amount of additional flow—at least as long as the same conductor is used.

As we have noted on various occasions, the non-crossing constraint is widely used in the literature. Our model ignores this constraint but we have experimentally determined how often they occur in cable layouts computed by our algorithms. We have seen that approximately one in thirteen NCC cable layouts has at least one crossing and that this number is approximately halved by the ILS. Relatively recently, there has been research on Minimum-Cost Flow Problems with edge conflicts: Conflicts relating to actual crossings have been considered in [AAŞT19] on a layered graph embedded into a grid. The same set of authors provide a more general model with arbitrary conflicts [ŞAA21]. In both cases, they prove NP-completeness and provide MILP formulations. In the latter work, the non-crossing constraint is the same as in [FP18]. Two exact algorithms are proposed for the General Minimum-Cost Flow Problem with Edge Conflicts [ŞAA21]. Yet, their model employs the classical linear cost function, so that the question remains to what extent their results may transfer to our cost function.

Negative Cycle Canceling is only one of many algorithms to compute minimum-cost flows in the classical setting; a presentation of many of these can be found in [AMO93, Chapters 9–11]. In their Bachelor's Thesis, co-supervised by this author, Jenne [Jen20] considered several of them. They conclude that "the Primal-Dual Algorithm, the Out-Of-Kilter Algorithm and the Relaxation Algorithm [face] serious problems [...] due to the non-linear step cost function" and that an adaptation of "the Successive Shortest Path Algorithm [...] is suitable" [Jen20, p. 49]. They find by means of experimental evaluation that their Successive Shortest Path Algorithm is faster than NCC but that it cannot keep up with it in terms of solution quality. Thus, we deem our NCC algorithm to be the currently best adaptation of a Minimum-Cost Flow Algorithm to WCP.

Our proposal to use Negative Cycle Canceling for the Wind Farm Cabling Problem has been picked up in the literature: The author of [Pér22] combines NCC with structural constraints. They enforce a tree-structure at all times, repair cable crossings if possible and then run NCC. Their version of NCC is restricted in the sense that cycles are only canceled if the tree-structure and the absence of crossings is not violated. The evaluation shows that NCC improves solutions in 64 % of the considered instances and that the solutions can effectively be used as warm-start solutions to a branch-and-cut solver.

4.11 Conclusion

We have considered the optimization problem to design cost-minimal cable layouts in (offshore) wind farms as a Minimum-Cost Flow Problem with a Step Cost Function. We have analyzed this problem theoretically, proving strong NP-hardness and an analogon to the Integer Flow Theorem. We have engineered an algorithm based on the classical Negative Cycle Canceling-technique and incorporated it into an Iterated Local Search. An extensive evaluation on synthetic benchmark instances and a case study on the Hornsea One wind farm have revealed that our algorithms are able to find solutions within tens of seconds and that they are competitive compared to a Mixed-Integer Linear Program formulation and Simulated Annealing. We have analysed the cable layouts computed by our NCC algorithm with respect to structural and electrical parameters by means of a power flow analysis. The electrical parameters are within reasonable boundaries. The structural observations show that frequently used constraints from the literature are only rarely violated and that the violations on cable crossings can effectively be addressed by our ILS.

In terms of future research directions we see possible further improvements to the performance of our algorithms. There are further speed-up techniques in the literature to decrease the number of iterations in the Bellman-Ford algorithm [Yen70, BE12] which could be considered for incorporation. Our ILS might benefit from the usage of dedicated frameworks for metaheuristics and special-purpose escaping strategies may help to lazily enforce additional constraints. Further theoretical improvement could possibly achieved by being able to cancel more complex structures than only cycles. The theory of flows shows that two flows differ in a set of cycles (in a suitable flow model) and that the cost difference of the flows is exactly the cost of the cycles [AMO93, Thm. 3.7]. In our setting, the assertion on the costs does not hold, see Figure 4.3 (e) and (g). The issue is that the edge vu_2 must be traversed twice but at different cost. The ability to efficiently detect the overlapping cycles to improve the solution could greatly improve the solution quality of our algorithms. Alternatively, a dedicated escaping strategy could be developed around canceling overlapping cycles.

To broaden the scope of our algorithms, the adjustments to other problem settings such as floating wind farms could be evaluated as we have outlined in Section 4.10. There may even be additional optimization problems with similar cost functions where our adaptation of Negative Cycle Canceling may prove helpful.

5 The Solar Farm Cable Layout Problem

The second renewable power plant with decentralized generators we consider in this work are solar farms. In Section 2.2 we have explained how solar farms are structured. In this chapter, we translate this structure into a graph-theoretic model and formulate the task of designing a cable layout as an optimization problem. We assume that the positions of PV strings are already determined and that the solar farm planner can choose both the positions of all other components from a set of candidate positions and the cable types used for their interconnection.

We define the Solar Farm Cable Layout Problem (SoFaCLaP) in Section 5.1 and give insights into its complexity in Section 5.2. SoFaCLaP is translated into an MILP formulation in Section 5.3. In order to evaluate the MILP we propose a framework to generate synthetic benchmark instances in Section 5.4 and populate this framework with parameters based on real-world solar farms. The evaluation itself can be found in Section 5.5 and is followed by a discussion of the model, the MILP and the benchmark instances in Section 5.6. We conclude with a summary and an outlook in Section 5.7. This chapter is based on joint work with Dominik Stampa and Matthias Wolf [GSW22b]. The proof of Theorem 7 is unpublished. Its idea goes back to [Sta22] but has been reworked in crucial points.

5.1 The Optimization Problem

The Solar Farm Cable Layout Problem is a Minimum-Cost Flow Problem on a directed layered graph $G = (V, E)$. The vertex set consists of the strings V_S that need to be connected, as well as of the potential Y-connectors V_Y, combiner boxes V_C, recombiner boxes V_R, (central) inverters V_I, and transformers V_T. Edges have a length $\text{len}\colon E \to \mathbb{R}_{\geq 0}$ and only exist between

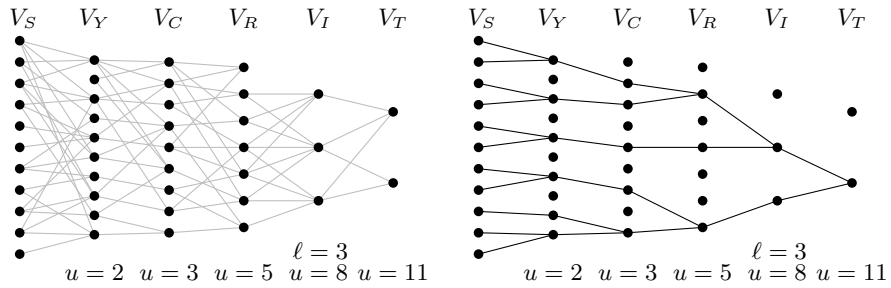


Figure 5.1: An example instance of SoFaCLaP showing the layered graph and the capacities of all layers *(left)* and—assuming that there is a cable type with sufficient capacity—a feasible cable layout *(right)*. The flow values on the edges are omitted in this visualization. They are, however, uniquely determined, since the cable layout is cycle-free, and can be computed by counting the strings in the respective subtrees of the layout.

one layer and the next, i. e.,

$$V = V_S \cup V_Y \cup V_C \cup V_R \cup V_I \cup V_T, \tag{5.1}$$

$$E \subseteq V_S \times V_Y \cup V_Y \times V_C \cup V_C \times V_R \cup V_R \times V_I \cup V_I \times V_T. \tag{5.2}$$

For easier reference, the layers are enumerated from $V_1 = V_S$ to $V_6 = V_T$. Vertices from layers $V_2, \dots, V_6$ have an upper capacity $u_i \in \mathbb{N}$ for $i = 2, \dots, 6$ on the amount of strings that can be routed via these vertices. Inverters also have a lower capacity ℓ_5. We may refer to the capacity bounds of a layer by a vertex of that layer, e. g. $u_2 = u(v)$ for any $v \in V_2$. The left-hand side of Figure 5.1 shows an example of layered graph with capacities as given by a SoFaCLaP instance.

The inverters split the instance into an AC-side with the transformers and a DC-side with the strings. Denote the edges on the AC-side by $E_{AC} = \{(i, j) \in E \colon i \in V_I, j \in V_T\}$ and by $E_{dc} = E \setminus E_{AC}$ the edges on the DC-side. For either side, there is a set of possible cable types K_{AC} and K_{DC}. Each cable type κ has a thermal capacity $cap_K(\kappa)$ and a cost per unit of length $c_K(\kappa)$. For easier notation, the subscripts AC and DC are omitted. The cable types include a dummy cable type of capacity and cost 0.

We define the flow in accordance with our definition in Section 3.2: The strings are the source vertices and we assume uniform generation, i. e., $p(v) = 1$ for all $v \in V_S$. The transformers are target vertices with their respective capacity $cap_V = u_6$. All other nodes are transshipment nodes and we can model their (upper) capacities as edge capacities on their outgoing edges: Consider an edge $e = (v, w)$ with $v \in V_i$ and $w \in V_{i+1}$ for some $i \in [5]$. Let, by abuse of notation, $cap_K(e)$ be the maximum cable capacity of any cable type that can be chosen for e. Then, the edge capacity is defined as

$$cap_E(v, w) = \min\{u(v), cap_K(v, w)\}.$$

Since electric current must be transmitted from a vertex to the next layer, we define $\mathrm{cap}_E(w, v) = 0$ for all $(v, w) \in E$. The additional constraints that the cable layout must be a forest and that the lower capacities at inverters must be respected, if they are used, do not readily translate into any of the Equations (3.2) to (3.5).

The goal of SoFaCLaP is to find a minimum-cost flow on G, i. e., a function $f : E \to \mathbb{R}$ subject to the flow conservation constraints

$$f_{\text{net}}(u) = -1 \qquad \forall u \in V_S, \tag{5.3}$$
$$f_{\text{net}}(u) \leq \mathrm{cap}_V(u) \qquad \forall u \in V_T, \tag{5.4}$$
$$f_{\text{net}}(u) = 0 \qquad \forall u \in V \setminus (V_S \cup V_T), \tag{5.5}$$

the edge capacity constraints

$$0 \leq f(u, v) \leq \mathrm{cap}_E(u, v) \qquad \forall uv \in E, \tag{5.6}$$

the lower capacities at inverters

$$\sum_{(u,v)\in E} f(u, v) > 0 \Rightarrow \sum_{(u,v)\in E} f(u, v) \geq \ell_5 \qquad \forall v \in V_I, \tag{5.7}$$

and subject to each vertex having at most one outgoing edge with positive flow

$$\sum_{(u,v)\in E} \chi_{\{f(u,v)>0\}} \leq 1 \qquad \forall u \in V \setminus V_T \tag{5.8}$$

of minimal cost

$$\mathrm{Cost}(f) = \sum_{e\in E} c(f(e)) \cdot \mathrm{len}(e). \tag{5.9}$$

Here, χ is the indicator function, which is 1 if the corresponding statement is true and 0 otherwise. The net flow is defined in Equation (3.1) and $c(f(e))$ is the same as for wind farms (Equation (4.7)) accounting for possible differences between AC and DC cable types. As before we may assume that the cable types are defined so that any type may realize the minimum in Equation (4.7) (cf. the assumptions leading up to Proposition 1).

There is obviously room for expanding this formulation to incorporate further aspects. We discuss some of them in Section 5.6 but mention one already here as it will be necessary for Section 5.4: Strings can be understood as having multiple connection points (*outlets*), out of which only one is chosen for the cable layout. Each outlet may have different Y-connectors it can be connected to. The model above already covers this variant: For any pair of string and Y-connector, only the connection point closest to the Y-connector could possibly be used in an optimal solution. Therefore, all outlets of a string can be contracted into a single one (and therefore identified with the string itself). The edge lengths from the string to Y-connectors are then adjusted to reflect the actual distance between Y-connector and closest outlet. The synthetic benchmark instances proposed in this work employ this variant using three connection points per string.

In Section 5.3 we show how SoFaCLaP can be modelled by an MILP formulation. But first, we look into the complexity of SoFaCLaP.

5.2 On the Complexity of SoFaCLaP

SoFaCLaP is a computationally difficult problem as we show in this section. We prove that is already strongly NP-hard to determine if a feasible solution, i. e., a flow, exist (Theorem 6) and that it remains strongly NP-hard if we apply simplifications to the problem. These simplifications represent a more applied setting of SoFaCLaP by using Euclidean distances and by assuming that any two vertices in neighboring layers can be connected by a cable. For this proof, however, we need to make use of the solution value. We conclude with a conjecture stating that determining feasibility is allegedly possible in polynomial time and explain our evidence why we believe the conjecture to be true.

Theorem 6. *It is strongly NP-complete to decide if an instance of SoFaCLaP has a feasible solution.*

Proof. A candidate for a feasible solution to SoFaCLaP can be provided by specifying the flow values on the edges. To verify feasibility, it must be checked that flow conservation holds and that all vertex and cable capacities are respected. This is possible in polynomial time. Thus, membership in NP is shown. Note that it is not necessary to compute the costs, which could involve computations with real numbers.

The proof uses a reduction from the strongly NP-complete problem 3-Partition [GJ79, SP15]: Let $m, T \in \mathbb{N}$ and let $S \coloneqq \{s_1, \dots, s_{3m}\}$ be a multiset of natural numbers such that $T/4 < s < T/2$ for all $s \in S$ and $\sum_{s \in S} s = mT$. Can S be partitioned into triplets $S_1, \dots, S_m$ such that $\sum_{s \in S_i} s = T$ for all $i = 1, \dots, m$?

In the reduction, the Y-connectors represent the elements of the multiset and the combiner boxes the triplets. The edges between strings and Y-connectors force an outflow from the Y-connector equal to the respective element of the multiset. So given an instance of 3-Partition, we construct an instance of SoFaCLaP with $3m$ Y-connectors $y_1, \dots, y_{3m}$, each with capacity $T/2$, and mT strings such that the s_i strings have an edge only to y_i for all $i = 1, \dots, 3m$. The instance has m combiner boxes $c_1, \dots, c_m$, each with capacity T. Recombiner boxes, inverters and transformers are not needed for this reduction, and neither are cable types, so there is one of each, with capacity mT. All layers are fully connected except for the string layer as mentioned above. This graph has $mT + 3m + T + 3$ vertices and $mT + 3mT + m + 2$ edges. Thus, the construction is possible in polynomial time since we may assume that T is polynomial in the size of the input.

We show that the instance of 3-Partition is a yes-instance if and only if the SoFaCLaP instance has a feasible solution. If the instance of 3-Partition is a yes-instance, then we connect a Y-connector y_i to a combiner box c_j if and only if $s_i \in S_j$. The edges between the strings and the Y-connectors yield that y_i has an outflow of exactly $s_i < T/2 = \mathrm{u}(y_i)$. Since $\sum_{s \in S_j} s = T$, the capacity of c_j is not exceeded either. Thus, we obtain a feasible SoFaCLaP instance.

On the other hand, let the constructed instance of SoFaCLaP be feasible. By means of $T/4 < s$, it follows that each combiner box has at most three Y-connectors connected to it. If one combiner box had only two Y-connectors, another one would have four. Thus, each combiner box is connected to exactly three Y-connectors. Since the total capacity of all combiner boxes is

exactly mT, the inflow at each combiner box equals T. Thus, the assignment of Y-connectors to combiner boxes gives an assignment of the elements of S to m triplets with the desired property. □

Note that we have used the same NP-hard problem for the hardness proofs for WCP (Theorem 3) and for SoFaCLaP. For SoFaCLaP the fact that we enforce a forest layout simplified the proof so that even the feasibility problem is hard. For WCP we have to use the solution value to count the number of edges and thereby obtain a forest layout.

We have shown that already finding a feasible solution is a difficult task, at least in the most general setting of SoFaCLaP. One might hope that the whole problem becomes a lot easier with additional simplifications. On the positive side, we suspect that finding a feasible solution is easy (Conjecture 1). On the negative side, it can be shown the optimization remains difficult:

Theorem 7. *It is strongly NP-hard to decide if a SoFaCLaP instance admits a solution with a value below a given threshold, even if the instance respects all of the following:*

- *Vertex positions are given by points in $\mathbb{Q}^2$ and no two vertices have the same positions.*
- *Edge lengths are given by the Euclidean distance of the endvertices.*
- *All layers are fully connected.*
- *There is only one cable type, which has unlimited capacity.*
- *Lower vertex capacities do not apply.*

Before we come to the proof of Theorem 7, we need some auxiliary results. For the reduction, we use a variant of the 3-Partition Problem. We refer to this variant as 3-Partition with bounds and distinct numbers:

Let $m, T \in \mathbb{N}$ and let $S := \{s_1, \dots, s_{3m}\}$ be a set (!) of natural numbers (i. e., $s_i \neq s_j$ for all $i \neq j$) such that $T/4 < s < T/2$ for all $s \in S$ and $\sum_{s \in S} s = mT$. Can S be partitioned into triplets $S_1, \dots, S_m$ such that $\sum_{s \in S_i} s = T$ for all $i = 1, \dots, m$?

Contrary to the 3-Partition variant used in the proof of Theorem 6, we now require the elements of S be distinct. It has been shown that this variant is strongly NP-hard without the $T/4 < s_i < T/2$ constraint [HWW08]. It is straightforward to see that from any such instance without bounds an equivalent instance can be constructed that observes the bounds: Let

$$a := 1 + \max\{2 \max s_i - T, T - 4 \min s_i, 0\}.$$

Furthermore, define $T' = T + 3a$ and $s'_i = s_i + a$ for all $i = 1, \dots, 3m$. Then it holds that $T'/4 < s'_i < T'/2$ for all i and

$$s'_{i_1} + s'_{i_2} + s'_{i_3} = T' \qquad \Leftrightarrow \qquad s_{i_1} + s_{i_2} + s_{i_3} = T.$$

Thus, 3-Partition with bounds and distinct numbers is also strongly NP-hard. We observe that $s_i > T/4 > 3m$ by combining the bounds and the fact that $3m$ different number must fit between $T/4$ and $T/2$.

In the proof of Theorem 7 we construct a solar farm instance from an instance of 3-Partition with bounds and distinct numbers. Some of the vertices are placed using the following proposition.

Proposition 2. *Let $C \in \mathbb{Q}^2$ and let $p \in \mathbb{N}$ and $\varepsilon \in \mathbb{Q}_{>0}$. Let*

$$\Gamma(t) \coloneqq \varepsilon \cdot \left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2} \right) + C$$

and let $t_i = -1 + 2i/p-1$ for $i = 0, \ldots, p-1$. Then, $\Gamma(t_0), \ldots, \Gamma(t_{p-1})$ are p points on a half-circle of radius ε around C such that

1. *all points have rational coordinates,*
2. *the coordinates are polynomially bounded in C, p, and ε, and*
3. *the distance of any two such points is at least $2\varepsilon/\sqrt{p^2-4p+5}$.*

Proof. A straight-forward computation shows $\|\Gamma(t) - C\| = \varepsilon$ for all $t \in \mathbb{R}$. Let $t_i = -1 + 2i/p-1$ for $i = 0, \ldots, p-1$. The p points $\Gamma(t_i)$ have rational coordinates. Since the respective first coordinates are all non-negative, all points lie on a half-circle. For any $i = 0, \ldots, p-1$ it holds that

$$\begin{aligned}
\Gamma(t_i) &= \varepsilon \left(\frac{1-t_i^2}{1+t_i^2}, \frac{2t_i}{1+t_i^2} \right) + C \\
&= \varepsilon \left(\frac{1-\left(-1+\frac{2i}{p-1}\right)^2}{1+\left(-1+\frac{2i}{p-1}\right)^2}, \frac{2\left(-1+\frac{2i}{p-1}\right)}{1+\left(-1+\frac{2i}{p-1}\right)^2} \right) + C \\
&= \varepsilon \left(\frac{(p-1)^2-(2i-(p-1))^2}{(p-1)^2+(2i-(p-1))^2}, \frac{2(p-1)(2i-(p-1))}{(p-1)^2+(2i-(p-1))^2} \right) + C
\end{aligned}$$

Each numerator and each denominator is in absolute values bounded by $10p^2$. Thus, the coordinates are polynomially bounded in C, p, and ε.

We observe that the minimum distance between any two points is realized by two consecutive points. Thus, we consider the auxiliary function $f_p \colon \mathbb{R} \to \mathbb{R}^2$ given by

$$f_p(t) = \|\Gamma(t + 2/(p-1)) - \Gamma(t)\|^2$$

Using $\Delta = 2/(p-1)$, thorough computations yield

$$f_p(t) = \left\| \frac{2\varepsilon\Delta}{(1+(t+\Delta)^2)\cdot(1+t^2)} \cdot \left(-2t-\Delta, 1-t\Delta-t^2\right) \right\|^2$$
$$= 4\varepsilon^2\Delta^2 \cdot \frac{(2t+\Delta)^2 + \left(t^2+\Delta t-1\right)^2}{(1+(t+\Delta)^2)^2\cdot(1+t^2)^2}$$

and after derivation

$$f_p'(t) = -\varepsilon^2(2t+\Delta)\cdot\frac{(2t+\Delta)^2+4-\Delta^2}{2\,(1+(t+\Delta)^2)^2\cdot(1+t^2)^2}$$

where it was used that $4(t^2+t\Delta-1) = (2t+\Delta)^2+4-\Delta^2$. We observe that $f_p(t)$ has a global maximum at $t = -\Delta/2$, and that it is strictly decreasing for $t > -\Delta/2$ and strictly increasing for $t < -\Delta/2$. Thus, the minimum distances between any two points are realized between the first two, as well as between the last two points.

Again by thorough computations, we obtain

$$f_p(-1) = \varepsilon^2 \cdot \frac{2\Delta^2}{\Delta^2-2\Delta+2}$$
$$= \varepsilon^2 \cdot \frac{\frac{8}{(p-1)^2}}{\frac{4}{(p-1)^2}-\frac{4}{p-1}+2} \cdot \frac{(p-1)^2}{(p-1)^2} = \varepsilon^2 \cdot \frac{4}{p^2-4p+5}.$$

□

With that, we have all ingredients for the main proof, except for one not-so-exciting technicality which we prove afterwards.

Proof of Theorem 7. Let (S,T) be an instance of 3-Partition with bounds and distinct numbers (in the following refered to as 3-Partition only). We construct a solar farm instance, in which vertices are placed in half-circles around other vertices. For the half-circles, we use radii of 1, $\varepsilon := 1/36m^2$, and $\delta := \varepsilon/T$. The placement of vertices on a half-circle is according to Proposition 2.

We construct an instance of SoFaCLaP as visualized in Figure 5.2: Place a recombiner box with capacity mT at the origin. On a half-circle of radius ε around the origin place m combiner boxes of capacity T. On a half-circle of radius 1 around the origin place $3m$ Y-connectors of capacity $\lfloor T/2 \rfloor$. On a half-circle of radius δ around the i-th Y-connector place s_i strings (with respect to those strings, the Y-connector is called the *intuitive* Y-connector). Finally, place an inverter of capacity mT at $(2,0)$ and a transformer of capacity mT at $(3,0)$. Proposition 2 and Lemma 2 include proofs that no two vertices have the same coordinates and that the coordinates are polynomially bounded. The single cable type is defined to have a cost of 1 per unit of length and it has unlimited capacity by the statement of the theorem.

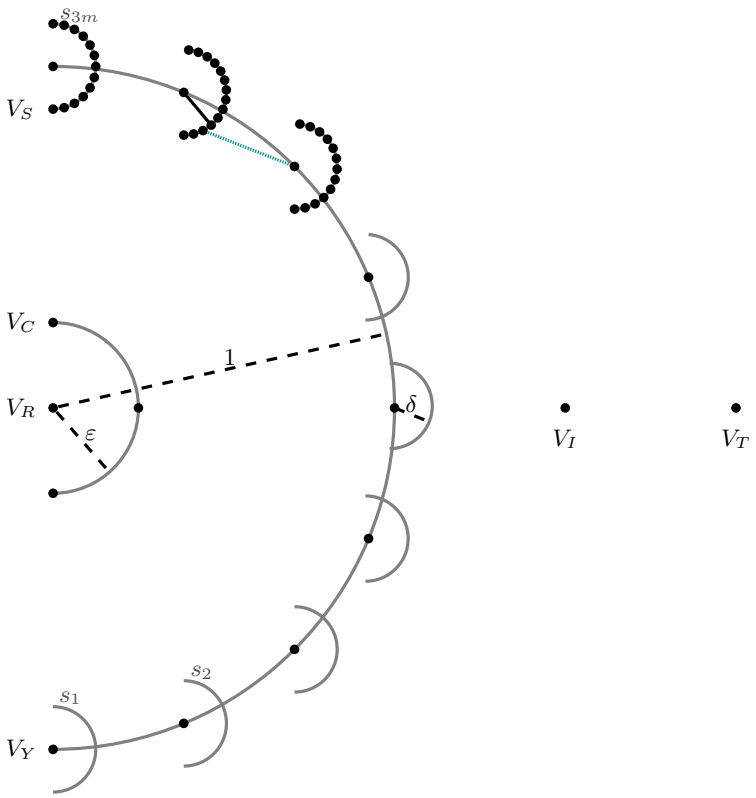


Figure 5.2: Visualization of a SoFaCLaP instance constructed from a 3-Partition instance with bounds and distinct numbers. Strings, Y-connectors and combiner boxes are placed on half circles marked by gray lines, some of the strings are omitted. The dashed lines show the radii of the half-circles. Adjacent layers are fully connected by assumption. In the second group of strings from the top, two edges are included: The black edge shows an "intuitive" connection, the dotted blue edge is an "unintuitive" connection.

The claim is that the 3-Partition instance is a yes-instance if and only if the SoFaCLaP instance has a solution of value at most

$$mT\delta + 3m + 4m\varepsilon + 3. \tag{5.10}$$

Assume that the 3-Partition instance is a yes-instance. Let S_j be the triplets of equal weight that partition S. Consider the cable layout in which the Y-connectors are connected exactly to the strings on the circle around them and in which the i-th Y-connector is connected to the j-th combiner box if and only if $s_i \in S_j$. The capacities of mT at the recombiner box, the inverter, and the transformer are respected since there are exactly mT strings. The capacities at the combiner boxes are not violated since $\sum_{s \in S_j} s = T$ and the capacities at Y-connectors are

respected since $s < T/2$ for all $s \in S$. The cable layout constructed from the 3-partition has total costs of at most

$$mT\delta + 3m(1+\varepsilon) + m\varepsilon + 2 + 1 = mT\delta + 3m + 4m\varepsilon + 3. \tag{5.11}$$

The summands of the first sum are (in this order) for the connection of strings to Y-connectors, Y-connectors to combiner boxes, combiner boxes to the recombiner box, recombiner box to inverter, and inverter to transformer. Thus, we constructed a feasible cable layout of sufficiently low costs as desired.

For the other direction, assume an arbitrary feasible cable layout. All mT strings have exactly one outgoing edge. Since the combiner box capacities sum up to mT, all combiner boxes are used. Thus, all feasible cable layouts differ only by the number of Y-connectors used or by the connections used between strings and Y-connectors as well as Y-connectors and combiner boxes. We show that only those cable layouts which use all Y-connectors and in which strings are connected to the respective Y-connector around which they are placed (i. e., all strings are connected *intuitively*) have costs at most (5.10). But first, we show some inequalities which we need later in the proof. A single string connected to a non-intuitive Y-connector instead of the intuitive one incurs a cost difference of at least

$$\frac{2}{\sqrt{(3m)^2 - 4(3m) + 5}} - \delta - \delta > \frac{2}{3m} - 2\delta = \frac{2}{3m} - \frac{1}{18Tm^2} > 0 \tag{5.12}$$

by Proposition 2, since $12m > 5$, and by the definition of δ. In Equation (5.12), one delta originates from cost of the intuitive layout and the other from estimating the distance to another Y-connector. Furthermore, by using the definitions of δ and ε we can even show

$$\frac{2}{3m} - 2\delta - 6m\varepsilon = \frac{2}{3m} - \frac{1}{36Tm^2} - \frac{1}{6m} > 0. \tag{5.13}$$

If the j-th Y-connector is not used, we save costs of at least $1 - \varepsilon$ from the connection to a combiner box. Since all strings around this Y-connector need to be connected unintuitively, the combined cost difference for all s_j strings is at least

$$\begin{aligned} & s_j \left(\frac{2}{3m} - 2\delta \right) - (1 - \varepsilon) \\ &= \left(\frac{2s_j}{3m} - 1 \right) + (\varepsilon - 2\delta s_j) \\ &> \left(\frac{6m}{3m} - 1 \right) + 0 > 6m\varepsilon > 0 \end{aligned} \tag{5.14}$$

plugging in the definitions of ε and δ and using that $3m < s_j < T/2$ from the 3-Partition variant.

Going back to the arbitrary feasible cable layout, let $J \subseteq \{1, \ldots, 3m\}$ be the set of indices of Y-connectors that are not used in the cable layout and let $k \geq \sum_{j \in J} s_j$ be the number of strings

that are not connected intuitively. We show that if $J \neq \emptyset$ or $k > \sum_{j \in J} s_j$, then the total costs of the layout exceed (5.10). The total cost of this layout is greater than

$$\begin{aligned} & k\left(\frac{2}{3m} - \delta\right) + (mT - k) \cdot \delta + (3m - |J|)(1 - \varepsilon) + m\varepsilon + 2 + 1 \\ = & k\left(\frac{2}{3m} - 2\delta\right) + mT\delta - 2m\varepsilon + 3m - |J|(1 - \varepsilon) + 3 \end{aligned} \tag{5.15}$$

The summands in the first line are (in this order) for the unintuitive and intuitive connections of strings, the connections of Y-connectors to combiner boxes, the connections of combiner boxes to the recombiner box, and for the connections to the inverter and the transformer. The difference to the threshold value from Equation (5.10) is

$$\begin{aligned} (5.15) - (5.10) &= k\left(\frac{2}{3m} - 2\delta\right) - 6m\varepsilon - |J|(1 - \varepsilon) \\ &= \left(k - \sum_{j \in J} s_j\right)\left(\frac{2}{3m} - 2\delta\right) - 6m\varepsilon + \sum_{j \in J} -(1 - \varepsilon) + s_j\left(\frac{2}{3m} - 2\delta\right). \end{aligned}$$

We distinguish cases on how the arbitrary cable layout differs from an intuitive one. If $J \neq \emptyset$, we use that the first summand is non-negative by Equation (5.12) and obtain

$$\begin{aligned} &= \left(k - \sum_{j \in J} s_j\right)\left(\frac{2}{3m} - 2\delta\right) - 6m\varepsilon + \sum_{j \in J} -(1 - \varepsilon) + s_j\left(\frac{2}{3m} - 2\delta\right) \\ &\geq -6m\varepsilon + \sum_{j \in J} -(1 - \varepsilon) + s_j\left(\frac{2}{3m} - 2\delta\right) > 0 \end{aligned}$$

by Equation (5.14). If $J = \emptyset$, the sum at the end of the line equals zero and we have $k > \sum_{j \in J} s_j$ by assumption. Thus

$$\begin{aligned} &= \left(k - \sum_{j \in J} s_j\right)\left(\frac{2}{3m} - 2\delta\right) - 6m\varepsilon - \sum_{j \in J}(1 - \varepsilon) + s_j\left(\frac{2}{3m} - 2\delta\right) \\ &\geq \left(\frac{2}{3m} - 2\delta\right) - 6m\varepsilon > 0 \end{aligned}$$

by Equation (5.13).

With these estimates, we have shown that in a cable layout for the constructed SoFaCLaP instance of cost at most (5.10), all Y-connectors are used and all strings are connected to the respective Y-connector around which they are placed. In such a cable layout, the i-th Y-connector has a total inflow of exactly s_i. Since all Y-connectors together provide an outflow of mT and there are m combiner boxes of capacity T, all combiner boxes need to be connected and used to their full capacity. To provide exactly T units of flow to a combiner box, exactly three Y-connectors are needed since $T/4 < s < T/2$ for all $s \in S$. So, in the cable layout of cost at

most (5.10), each combiner box obtains flow from exactly three Y-connectors and no Y-connector can supply two combiner boxes. Thus, for the 3-Partition instance we obtain triplets S_j by assigning s_i to S_j if the i-th Y-connector is connected to the j-th combiner box and it holds that $\sum_{s \in S_j} s = T$ as this is the flow value into the combiner boxes. □

One lemma is missing in order to complete the proof.

Lemma 2. *Let G be the solar farm graph constructed in the proof of Theorem 7. Then, no two vertices have the same coordinates.*

Proof. To show that no two vertices have the same coordinates it remains two show that no two half-circles intersect as well as that the inverter (and thus the transformer) is farther from the origin than the farthest possible position for a string. The distance from the origin to any string is at most

$$1 + \delta = \frac{36Tm^2 + 1}{36Tm^2} < 2.$$

Thus, the strings cannot be co-located with the inverter. As for the half-circles, we show that two half-circles of strings do not intersect each other and that a half-circle of strings does not intersect the half-circle of combiner boxes. By Proposition 2, the minimum distance of two strings on different half-circles is greater than

$$\frac{2}{\sqrt{(3m^2) - 4(3m) + 5}} - 2\delta > 0 \qquad \text{by Equation (5.12).}$$

The distance of a string to the origin is at least

$$1 - \delta = \frac{36Tm^2 - 1}{36Tm^2} > \frac{1}{2} > \frac{1}{36m^2} = \varepsilon,$$

which is the distance of the combiner boxes to the origin. Thus, no two vertices are co-located. □

Thus, we have completed the proof of Theorem 7 showing that SoFaCLaP remains strongly NP-hard even under more realistic or simplifying assumptions. However, we do believe that it is easy to find feasible solution in that case.

Conjecture 1. *In the setting of Theorem 7, a feasible solution can be computed in polynomial time.*

The idea for this conjecture goes back to discussions with Thomas Bläsius and Matthias Wolf in early 2022. The crucial observation in the setting of Conjecture 1 and Theorem 7 is that the vertices within a layer need not be distinguished for the sake of finding a feasible solution. This may yield a dynamic program as follows: A solution may be described by a sequence of numbers,

each entry determines the number of vertices used within a layer. As an example, consider the right-hand side of Figure 5.1. The depicted solution may be encoded as $(11, 6, 5, 3, 2, 1)$ as it connects eleven strings via six Y-connectors, five combiner boxes and so on. This solution may be put together from the sequences representing the trees rooted at the inverters, namely $(6, 3, 3, 2, 1)$ and $(5, 3, 2, 1, 1)$, and at this step the vertex capacities of the last layer have to be checked. One way for the dynamic program to be run is to collect all non-dominated solutions corresponding to the number of layers from the bottom and the number of strings connected, where a solution is dominated if there is another solution that uses not more vertices of any layer (and less vertices in at least one layer). The difficulty here is to ensure that the number of solutions corresponding to a number of layers and a number of strings does not increase too much. The vertex capacities and their interactions may come into play here in order to bound the number of solutions.

If Conjecture 1 is true, this will allows us to efficiently compute initial solutions which can then be improven (meta-)heuristically. The resulting heuristics can then be evaluated in comparison with the MILP formulation of the next section.

5.3 MILP Formulation

SoFaCLaP can be formulated as an MILP using variables $f(u,v) \geq 0$ for the flow on $(u,v) \in E$ and binary variables $x_{uv\kappa}$ stating whether cable type κ is used on edge (u,v). Here, the zero-capacity cable type is omitted. This way of modelling the cable types is the same as we have seen for wind farms Section 4.3. The following formulation is based on the standard variant of SoFaCLaP with the implicit inclusion of string connection points as outlined in Section 5.1.

The goal of SoFaCLaP is to minimize the total installation cost

$$\min \sum_{(u,v)\in E} \sum_{\kappa\in \mathrm{K}} x_{uv\kappa} \cdot \mathrm{c_K}(\kappa) \cdot \mathrm{len}(u,v). \tag{5.16}$$

The total flow leaving each string is the production of the string

$$\sum_{(u,v)\in E} f(u,v) = 1 \qquad \forall u \in V_S, \tag{5.17}$$

and flow must be conserved at intermediate vertices

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w) \qquad \forall v \in V \setminus (V_S \cup V_T). \tag{5.18}$$

The capacities at the vertices must be respected

$$\sum_{(u,v)\in E} f(u,v) \leq \mathrm{u}(v) \qquad \forall v \in V \setminus V_S, \tag{5.19}$$

$$\sum_{(u,v)\in E} f(u,v) \geq \ell_5 \cdot \sum_{(u,v)\in E} \sum_{\kappa\in \mathrm{K}} x_{uv\kappa} \qquad \forall v \in V_I, \tag{5.20}$$

whereas lower capacities only apply if a vertex is indeed used.
For all vertices except transformers, only one outgoing edge may have flow

$$\sum_{(u,v)\in E}\sum_{\kappa\in \mathrm{K}} x_{uv\kappa} \leq 1 \qquad \forall u \in V \setminus V_T, \tag{5.21}$$

and thereby non-zero capacity, which also implies that only one cable type is used per edge. This cable type must have sufficient capacity to hold the flow

$$f(u,v) \leq \sum_{\kappa\in \mathrm{K}} x_{uv\kappa} \cdot \mathrm{cap}_{\mathrm{K}}(\kappa) \qquad \forall (u,v) \in E. \tag{5.22}$$

It is easy to see that the MILP formulation is correct. We will, however, point out that we translated the enforcement of vertex capacities from outgoing edges in Section 5.1, in particular Equation (5.6), to the incoming edges (Equation (5.19)). This also includes transformer capacities which we needed to enforce separately in Section 5.1 Equation (5.4).

In total, this linear program has $\mathcal{O}(|\mathrm{K}| \cdot |E|)$ binary and $|E|$ real variables as well as $\mathcal{O}(|V| + |E|)$ constraints.

5.4 Generation of Benchmark Instances

An evaluation of the MILP formulation from Section 5.3 on example instances shall give the reader insights into the performance of the MILP and thereby establish it as a (first) solution method to SoFaCLaP. Since we have not been able to obtain suitable real-world instances[1] to evaluate the MILP, we resort to generating synthetic instances as described in the following. The benchmark instances use the GraphML format [Bra+02] and are available from [GSW22a], where a thorough specification of the format can be found as well.

If the benchmark instances are sufficiently variable to cover a wide range of plausible inputs to the solution approaches, the use of synthetic instances has multiple advantages. First, a larger number of different instances can be generated so that tendencies in the comparison of two approaches are less likely to be a result of statistical noise. Second, the solar farms in consideration can include more strings than any currently existing solar farms. Thus, the applicability of the algorithms for a future increase in solar farm sizes can already be investigated now.

5.4.1 Framework for Benchmark Generation

We describe the general approach we propose for the generation of benchmark instances on a high level in this section. The next section is dedicated to breathing life into the framework by describing the process in more detailed steps with concrete values for the parameters. These values are based on real-world examples of devices used in solar farms as evidenced by Tables 5.1 and 5.2 in Section 5.4.2.

[1] Note that it does not suffice to merely have the positions of the strings and other devices in an existing solar farm. We rather need all potential positions of devices. This represents an intermediate step in the planning process which is, understandibly, not publicly available.

The first steps in the framework deal with placing the strings. For synthetic instances we opt for randomly sampling the strings with a minimum distance. Next, the location of vertices of higher layers are determined, starting at the Y-connectors. We let the number of vertices in a layer depend on the number of vertices in a previous layer. We assume fully connected layers, which leads to the maximum number of possible connections between layers. After the graph is determined, vertex capacities are set. We choose these randomly with bounds based on the minimum capacity necessary to connect all strings. Lastly, cable types are included which we derive from real-world photovoltaic cables.

Users of the proposed framework may want to alter any intermediate step to realize certain additionally desired properties on the solar farm instances. For example, users could prespecify all the string positions or fix the number of vertices within a layer. They may want to change the sampling process, for example to have all recombiner boxes close to each other. The edge set can be thinned out if, for example, cables should not exceed a certain length. Users can define their own cable types.

5.4.2 Generating Benchmark Instances

For our purposes, the framework introduced in the previous section is used as follows. Benchmark instances are generated in three categories: *Small* (120–180 strings per instance), *Medium* (500–750 strings), and *Large* (1200–1500 strings). We describe the process of generating an instance: First, a rectangle is fixed that simulates the area of the solar farm. One of the corners is defined as the origin giving rise to a coordinate system. The (virtual) unit of length is irrelevant since SoFaCLaP solution values scale with the underlying unit of length. Next, an angle α for the orientation of the strings in relation to the coordinate system is chosen randomly. The total number of strings is drawn from the intervals above and, one after another, the strings are placed in the rectangle: A random point P is sampled from the rectangle serving as one of the connection points of the string. For the purpose of describing the process, we refer to this point as the *base* of the string. If P respects a globally specified minimum distance from the bases of all previously successfully sampled strings, the connection points are defined. On a ray starting at P with angle α to the coordinate system's x-axis two more connection points are placed equidistantly, with the distance being the same for all strings. If, however, P is too close to another base, a new base P is sampled. If the process of placing a string fails too often, the whole instance is discarded and a new rectangle is chosen.

With all the strings placed, the higher layers of vertices are randomly placed, starting with the Y-connectors. For each layer, a ratio according to Table 5.1 is chosen randomly and the number of vertices on the next layer is defined as $V_{\text{next}} = \lceil V_{\text{reference}}/\text{ratio} \rceil$. The ratios are chosen in a way to ensure decreasing sizes of layers. For the *small* instances, the number of transformers and inverters is fixed as 1. Y-connectors and combiner boxes are placed in close proximity to string connection points. Recombiner boxes are placed close to combiner boxes. Inverters and transformers are placed on random points in the rectangle, respecting minimum distances to the string bases as well as previously placed inverters and transformers. The reasoning is that

Table 5.1: Design parameters for graph layer sizes

Layer		Category		
in consideration	reference	*Small*	*Medium*	*Large*
V_S	1	$120 - 180$	$500 - 750$	$1200 - 1500$
V_Y	V_S	$1.5 - 3$	$1.5 - 3$	$1.5 - 3$
V_C	V_S [(1)]	$10 - 20$	$10 - 20$	$10 - 20$
V_R	V_C [(2)]	$3 - 8$	$3 - 10$[(3)]	$3 - 10$[(3)]
V_I	V_S	$\lvert V_I \rvert = 1$	$200 - 300$	$200 - 300$
V_T	V_I	$\lvert V_T \rvert = 1$	$1 - 3$[(4)]	$1 - 3$[(4)]

[(1)] In the example solar farm by ABB [ABB19, Annex B], each combiner box connectes 13 or 14 strings. A combiner box by LS Electric has possible inputs of 12, 16, or 20 [LS21]. Combiner boxes by SMA connect up to 32 strings, but no recombiner boxes are used [SMA20].

[(2)] based on the recombiner boxes in [Sol]

[(3)] 10 or (26 − ratio $V_C : V_S$), whichever is lower, based on the idea that parallel connections of many strings in both combiner and recombiner boxes yields excessive amounts of electric current

[(4)] Gajda [Gaj16] shows two inverters connected to a transformer.

strings, inverters and transformers are themselves bigger components and need to be more easily accessible for maintenance.

Picking (upper and lower) capacity values for the different layers can be a delicate issue from an algorithmic point of view: With very tight capacities, the design questions of picking one position for a, say, combiner box over another becomes easy: all possible positions have to be used. Very loose capacities mitigate the algorithmic difficulty of picking a suitable subset of, say, strings to be assigned to a specific Y-connector. We deem the assumption fair that all vertices of a layer have the same capacities since planners presumably would not design solar farms with two different versions of, say, inverters. Given a layer V_i a capacity of at least $\mathrm{lb}_i := \lceil {}^{\lvert V_S \rvert}/_{\lvert V_i \rvert} \rceil$ is required. For each layer, a random integer between lb_i and $\lambda \cdot \mathrm{lb}_i$ is chosen, where $\lambda = 1.2$ for inverters and $\lambda = 1.5$ for all other layers. The smaller value for inverters is selected with the idea in mind that solar farm planners will not overly deviate the capacities of inverters from the capacity best suited for the intended use. For recombiner boxes and inverters only, the capacity will, however, be at least twice the capacity of the previous layer to ensure that two fully-used vertices of the previous layer can be connected. The randomly chosen lower capacity for inverters is at least 0.5 and at most 0.8 times the inverters' upper capacity.

The edge set is assumed to be complete, i. e., equality holds in Equation (5.2). We have no reason to prohibit specific edges (although solar farm planners might). Thus, we leave it to the algorithms to pick the most suitable edges from the complete set.

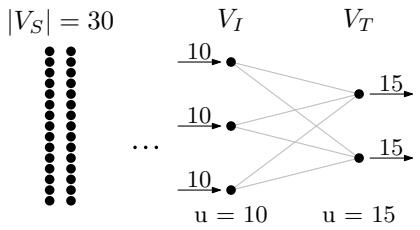


Figure 5.3: A toy example of a solar farm, in which all vertex capacities are sufficient but their interaction implies infeasibility. Due to 30 strings, each inverter needs an inflow of 10 and each transformer an inflow of 15. But with only one outgoing edge per vertex allowed, inverters and transformers cannot be connected.

Table 5.2: Cable types used in simulations

Cable Type		Specifications as in [Hel21]	
cost c_K	capacity cap_K	Ampacity [A] at 60 °C	Copper Weight [kg km^{-1}]
4	5	55	38.4
34	22	218	336
120	50	488	1152
230	80	775	2304
750	180	extrapolated	
2300	400	extrapolated	

The cable types for the simulations as shown in Table 5.2 are based on the photovoltaics cable SOLARFLEX-X PV1-F by HELUKABEL [Hel21]. Four sizes are considered, the forth one being the biggest size available. The rated ampacity is approximately translated into a capacity on the amount of strings, where a current of 10 A per string is assumed [ABB19, p. 121]. Since cost quotations are not readily available, we assumed that the costs are in essence proportional to the amount of copper used in a cable. The exact unit for the costs is irrelevant since the optimization function in SoFaCLaP can be scaled arbitrarily. For aesthetical reasons, we opted for somewhat round values in the bigger cable types. No influence on the performance of algorithms is expected from those changes. These four cable types are not big enough to be used in the largest benchmark instances. Thus, we included two artificial cable types to conduct meaningful experiments on the biggest instances. These cable types were extrapolated from the smaller cables types, roughly following the motto "Doubling the capacities yields triple the cost."

Some of the benchmark instances generated as outlined above may be infeasible. Since capacities are chosen in a way that the total capacity of a layer is at least the number of strings and since the edge set is complete, infeasibility can only arise from the interaction of the sizes and capacities of the layers as a whole. A toy example of such a constellation can be seen in Figure 5.3. We purposefully keep such instances because it may be a feature of algorithms developed for SoFaCLaP to be able to detect infeasibility (see also Section 5.5.1).

5.5 Evaluation

The MILP formulation from Section 5.3 is translated into C++14 code. The code uses the Open Graph Drawing Framework (Catalpa release) [Chi+13] for parsing the benchmark instances and the C++-API provided by Gurobi [Gur18]. For compilation, GCC 10.3.0 is used with the `-Ofast -march=native` flags. All simulations are run on a 64-bit architecture with four 12-core AMD CPUs each at 2.1 GHz with the openSUSE Leap 15.3 operating system. The MILP formulation is solved by Gurobi 9.5.0 in single-threaded mode with a maximum running time of one day per instance, with 40 instances running simultaneously. For notational ease, we will use "Gurobi" and "the MILP" interchangibly.

For the simulations, 80 randomly generated instances of each of the three categories *Small*, *Medium*, and *Large* are considered. An overview of the characteristics of these 240 instances is shown in Table 5.3.

Table 5.3: Characteristics of instances used in simulations

Category	$\lvert V_S \rvert$			$\lvert V \rvert$			$\lvert E \rvert$		
	min	median	max	min	median	max	min	median	max
Small	120	145	180	176	224.5	305	5566	10 528	20 515
Medium	502	632	750	781	966.5	1234	108 654	190 792	345 969
Large	1200	1248	1500	1726	1920	2459	538 923	732 858	1 369 140

The ranges for the number of strings are almost fully used. It stands out that the median of number of strings in the *large* instances is much smaller than 1350, which is what one would naïvely expect. A possible reason is that the process of placing a large number of strings, inverters and transformers, each of which observe a minimum distance to each other, failed too often. In that case, the instances would be discarded and a new number of strings would be drawn randomly. A possible remedy is to enlarge the initial rectangle for the solar farms for large instances.

The number of vertices is linear in the number of strings with a small coefficient. This is as expected from the ratios shown in Table 5.1. The number of edges grows fast with over 1.3 million edges for instances with up to 1500 strings. From a theoretical perspective, a fast increase is expected: Layers are fully connected and the number of vertices per layer is linear in the number of strings. Thus, the number of edges grows quadratically with the number of strings.

The remainder of this section is structured as follows: In Section 5.5.1 we evaluate the MILP with respect to its ability to determine whether an instance is feasible or infeasible. Some instances are solved to (proven) optimality and we investigate the running time needed to do so in Section 5.5.2. In Section 5.5.3 we look at those instances which have been determined to be feasible and investigate the MIP gaps (cf. Section 4.3) computed by Gurobi. The goal of this evaluation is to provide baseline results to which future algorithmic approaches to SoFaCLaP can be compared to. The results of our evaluation are publicly available [GSW22c].

5.5.1 Optimality and (In-)Feasibility

In our setting, Gurobi terminates with one of four optimization statuses: Ideally, an instance is solved to (proven) optimality (status: *optimal*). Secondly, Gurobi may find a feasible solution but fail to prove that it is optimal (status: *feasible*). In that case, the best proven lower bound (lb) does not match the solution value (ub), and the feasible solution may or may not be optimal. Thirdly, Gurobi may be able to prove that an instance does not admit any feasible solution (status: *infeasible*). Lastly, Gurobi may not find any feasible solution but fail to prove infeasibility. In that case, the instance may or may not be feasible (status: *unknown*).

In the following, we want to investigate how often Gurobi terminates in each of those states across the different categories of instances to see what we can learn about the applicability of the MILP formulation and Gurobi to solve SoFaCLaP.

The upper part of Table 5.4 shows how often Gurobi terminates in the different states at the end of the maximum running time of one day. Nearly all of the small instances are solved to optimality within one day. A solution is considered optimal once the MIP gap (defined in Section 4.3) is below the optimality tolerance of 0.0001 %. Since the number of variables and constraints in the MILP formulation grow linearly with the number of edges, Gurobi can most probably not uphold the performance on bigger instances.

Indeed, for the medium instances, the number of instances that are solved to optimality decreases by a big margin compared to the small instances, but still one in five instances is solved optimally. About one in seven instances are shown to be infeasible. The fact that Gurobi does not find any feasible solution on 4 out of 80 instances within one day (optimization status: unknown) makes us suspect that those instances are infeasible. On instances of those sizes one would expect that Gurobi would find a feasible solution within one day if a solution existed. We come back to those "unknown" instances later in this section.

As for the biggest instances, Gurobi is not able to solve any instance to optimality. For more than one in four instances, Gurobi can neither find an optimal solution nor prove infeasibility. We speculate that allowing Gurobi more time or simplifying the model can give further insights into the true status of these instances.

Thus, we run two additional sets of simulations on the remaining "unknown" instances. The first set is intended to find feasible solutions. We suspect any then remaining unknown instance to be infeasible. On those instances, we use a more restricted but (in the context of infeasibility) equivalent MILP formulation which we believe helps Gurobi to prove infeasibility. We describe

Table 5.4: Amount of instances per optimization status for different stages of MILP simulations. The specifications for the experiments are in the main text body.

Category	MILP (Section 5.3), one day				
	total	optimal	feasible	infeasible	unknown
Small	80	79	1	0	0
Medium	80	16	48	12	4
Large	80	0	50	5	25
	MILP (Section 5.3), one cable type, one day				
	total	optimal	feasible	infeasible	unknown
Small	0	—	—	—	—
Medium	4	—	0	2	2
Large	25	—	14	5	6
	MILP with Equation (5.23), one cable type				
	total	optimal	feasible	infeasible	unknown
Small	0	—	—	—	—
Medium	2	—	0	1	1
Large	6	—	0	1	5

the two sets of simulations in more detail and see what we can learn about the instances of unknown status.

To make Gurobi's life (supposedly) easier on the remaining instances of *unknown* status, we allow only one cable type. This greatly reduces the number of binary variables at the start of the optimization. If the capacity of the cable type is set to a value higher than the maximum vertex capacity, the simplified instance is feasible if and only if the original instance is, since the biggest cable capacity exceeds the biggest vertex capacity in all instances in question here. Again, we give Gurobi a maximum running time of one day per instance in single-threaded mode. The outcome of these simulations is shown in the middle part of Table 5.4. Gurobi can prove infeasibility on two additional medium and on five additional large instances. Gurobi also finds a first feasible solution on 14 out of 25 large instances. Yet, two medium and six large instances remain unknown.

Given the amount of help we gave Gurobi to find feasible solutions, we believe that the remaining instances are in fact infeasible. To facilitate Gurobi's proof of infeasibility, we again change the MILP formulation. Since the capacities within a layer are equal and layers are fully connected, we observe that any feasible solution can be transformed into other feasible solutions by a permutation of the vertices of a layer. In order to prove infeasibility, Gurobi needs to

outrule all these possibilities. Presumably, this can be facilitated by imposing a fixed order on the vertices of each layer, in which the inflow into the vertices decreases. In formulae, fix a layer V' (other than the strings) and enumerate the vertices inside this layer as $V' = \{v_1, \ldots, v_k\}$. Then, add the constraints

$$|V_S| \geq \sum_{(u,v_1)\in E} f_{uv_1} \geq \ldots \geq \sum_{(u,v_k)\in E} f_{uv_k} \geq 0 \tag{5.23}$$

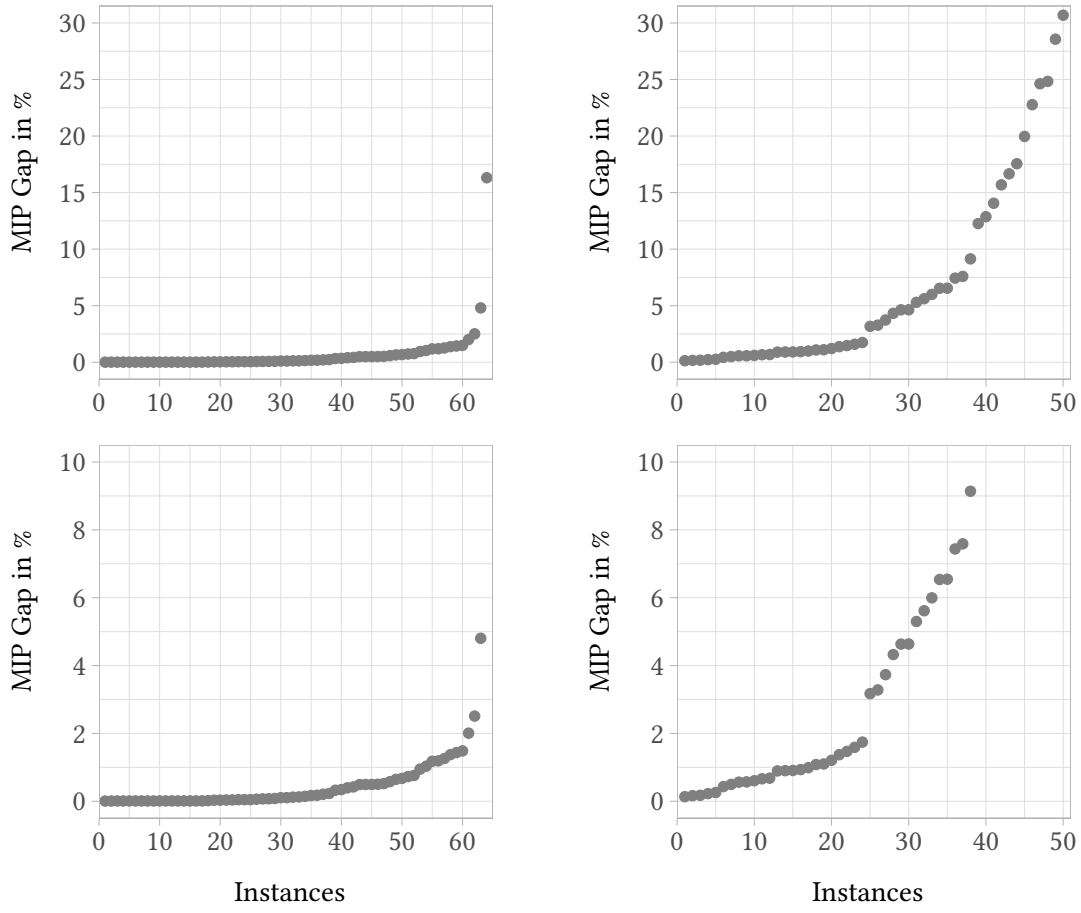
for all layers V'. Additionally, we set the target function to a constant, so that Gurobi terminates once it finds a feasible solution. With this formulation, we allow Gurobi four threads per instance and lift the time limit. Surprisingly, these changes do not help Gurobi much. Even with a total computation time over all threads per instance of more than six weeks, only two additional instances are proven infeasible (cf. lower part of Table 5.4).

We conclude from these investigations on the optimization status with which Gurobi terminates that Gurobi is very well suited for solving small instances and obtaining feasible solutions on quite a number of instances representing medium and large solar farms. We also note that Gurobi is able to prove infeasibility quite often in reasonable time (with a bit of help), even though our trick with the additional constraints from Equation (5.23) proved less useful than hoped for. For some large instances, Gurobi cannot find a feasible solution within one day even though these instances are in fact feasible. While allowing Gurobi more time is always an option, one could think about other approaches to find a feasible solution (cf. Conjecture 1), give it to Gurobi as a warmstart and see what Gurobi can do from there.

5.5.2 Running Time

We take another look into the instances that are solved to optimality concerning the time Gurobi needs to reach this state. 79 out of 80 small instances are solved to optimality. Gurobi reaches this result within half a minute on 55 instances and within five minutes on 78 instances. The longest proof of optimality on the small instances needs just under ten minutes. We suspect that Gurobi has no difficulties on the small instances, since the number of potentially non-zero binary variables can be greatly reduced by combining the constraints: For string edges it is never beneficial to use another than the smallest cable type. Also, the biggest cable type is never used on small instances since the second biggest suffices. More sophisticatedly, the small number of strings and therefore small number of vertices in further layers limits the number of cable types that can be used to due the vertex capacities. For example, consider a solar farm with 180 strings and 60 Y-connectors. By the formulae outlined in Section 5.4.2, Y-connectors can only have a capacity of 3 or 4. In both cases, the smallest cable type also suffices for the edges between Y-connectors and combiner boxes. For all these reasons, we believe Gurobi is able to cut off many suboptimal solutions very easily, which results in short running times.

One medium instance is solved to optimality within one hour (within 20 minutes actually) and seven more within four hours. The longest time for a proof of optimality here is 18 hours. Clearly, with more running time, the number of instances solved to optimality would increase further. Unfortunately, it cannot be known how long Gurobi needs. So it is more insightful to look into the worst-case deviation of a best solution from the (unknown) optimal solution value. For this, we look at the MIP gaps in the following section and in Figure 5.4.



(a) Instances of category *Medium* **(b)** Instances of category *Large*

Figure 5.4: MIP gaps sorted in ascending order for instances from two of three categories. Almost all *small* instances are solved optimally. Depicted are 64 medium and 50 large instances, including those solved to optimality. The upper figures show the gaps for all instances, the lower figures are zoomed in for better view.

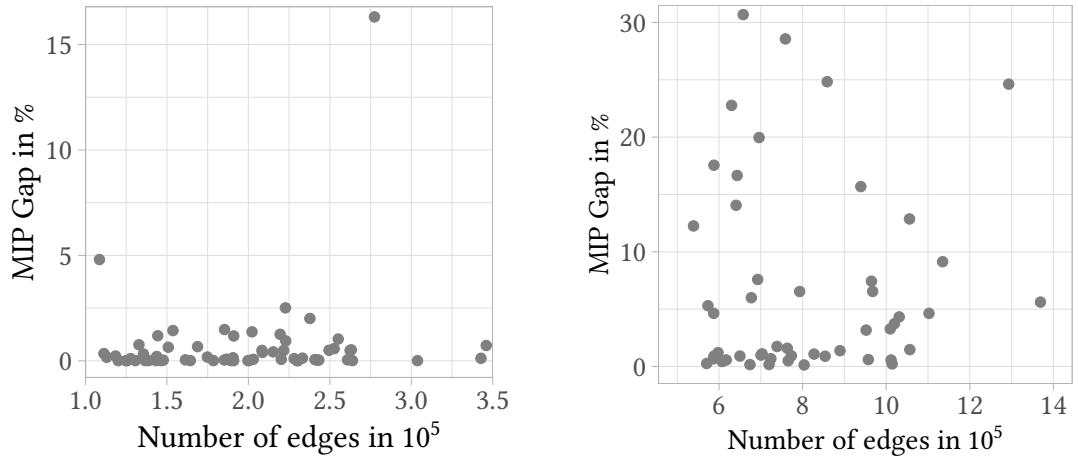
5.5.3 Solution Quality

We look into the quality of the solutions Gurobi has found within one day using the original MILP formulation. By the upper part of Table 5.4, the instances in consideration are all with status *optimal* or *feasible*. These are 80 small, 64 medium and 50 large instances. There is not much to say about the small instances; they are almost all solved to optimality. The remaining instance has 180 strings (the maximum) and 11749 edges. On this instance, Gurobi terminates with a MIP gap of 0.0161 %. That means between the best solution found is at most 0.0161 % worse than the optimal solution which is very close to being optimal.

In the other two categories, there are a lot more feasible but not optimal instances. Figure 5.4 depicts the gaps sorted increasingly for the medium and large instances. Note that these instances include all instances for which Gurobi has found a solution, whether these solutions have been proven to be optimal or not. All but one medium instance are solved with a gap of less than 5 %, that means that the best solution found by Gurobi is at most 5 % worse than the optimal solution. A gap of less than 1 % has been computed for 53 out of 64 instances and a gap of less than 0.1 % for 29 instances. On the large instances, the gap remains bigger than 10 % on twelve instances. A gap of less than 5 % has been proven for 30 instances and of less than 1 % for 17 instances. The smallest gap on the large instances is at approximately 0.13 %.

One may reasonably expect that the bigger the instances, the more difficult it is for Gurobi to prove optimality or infeasibility. While this is certainly true when comparing medium with large instanceas (evidence shown in Figure 5.4 and Table 5.4), it is not clear when only somewhat similarly sized instances are considered, e. g. only the medium instances. Since the number of provably infeasible and of "unknown" instances is very small, we will not address that relation in the context of infeasibility. Any observation on such a small data set might be pure coincidence. Concerning the proof of optimality, however, we depict the MIP gaps as a function of the number of edges for the aforementioned 64 medium and 50 large instances in Figure 5.5. For the medium instances it stands out is that the smallest instance in terms of edges has the second highest gap and that the three biggest instances have very small gaps, including the fifth lowest. Looking at the bulk of the instances, no tendency between MIP gap and number of edges is apparent. In the large instances, the gaps are more spread but also no pattern can be seen. With these pieces of evidence, we conclude that the number of edges cannot solely explain the different performance of Gurobi within the categories of instances. Presumably, the interaction of layer sizes with vertex and cable capacities also plays a role here.

As a side note: Figure 5.5 shows only a small number of instances at the upper end of the edge set sizes. We have made a similar observation concerning the number of vertices of large instances in the discussion of Table 5.3 in the introduction to Section 5.5. We believe that the problem that many of the bigger instances in the category *large* are discarded in the process of placing strings, inverters, and transformers also persists for the medium instances (and maybe even for the small instances).



(a) Instances of category *Medium*

(b) Instances of category *Large*

Figure 5.5: The MIP gap as a function of the number of edges on *medium* and *large* instances. Depicted are 64 medium and 50 large instances, including those solved to optimality. Be aware of the differently scaled y-axes.

Comparison to the Gaps from WCP. Having small MIP gaps provides certainty that the best solution found is close to the optimal. As we have seen, Gurobi delivers this certainty on a vast amount of instances across all categories. To contextualize these numbers, we compare them to the MIP gaps computed from the MILP formulation in Section 4.3, which we have reported in Figure 4.11.

The main differences between WCP and SoFaCLaP lie in the layered structure of solar farms including vertex capacities and the tree-structure enforced by Equation (5.21). The benchmark instances for WCP have a number of edges approximately linear in the number of vertices, since an approach employing k-nearest neighbours is used to define the edge set [LRWW17]. To the contrary, the edge sets of the solar farm instances proposed here grow quadratically in the number of vertices. One benchmark set $\mathcal{N}_3$ of wind farm instances has between 80 and 180 turbines per instance and is therefore a bit smaller but still comparable to our small instances. The $\mathcal{N}_4$ set of wind farms has between 200 and 499 turbines. These wind farms are a lot smaller than our medium instances. The MIP gaps reported for these wind farm benchmark sets are roughly between 25 % and 30 % with no outliers after a running time of one day per instance. In comparison, all but one small solar farm have been solved to optimality and all but one medium solar farm have a MIP gap below 5 %. Even for the large solar farms, the MIP gaps look better, at least as long Gurobi finds a feasible solution. To be fair, the experiments underlying Figure 4.11 have used Gurobi 8.0.0, while we use Gurobi 9.5.0 for solar farms. Still, this cannot explain the remarkable difference in the MIP gaps. The reason for the low MIP gaps probably lies within the structural properties of SoFaCLaP and the corresponding MILP formulation. Again, a lot

of variables are probably easily dealt with by Gurobi, as we mentioned earlier in this section. Furthermore, the layered structure of the graph yields MILP constraints that only involve a local part of the graph. Thus, the matrix combining the variables into constraints may have a more block-like structure, which may also help the solver.

5.6 Discussion

Before we come to a conclusion, let us discuss various aspects of the model, the MILP and our benchmark instance generation, some of which may be of interested to solar farm planners interested in adapting our model and algorithmic approaches built upon it.

As we have mentioned in Section 5.1, our model already incorporates the idea of strings having multiple outlets: For each string, one outlet has to be chosen to connect the string to the next layer, but different outlets may not necessarily be connected to the same Y-connectors and the distances may also differ. We have seen that we can contract all outlets of a string into a single outlet and thereby identify it with the string itself. The edge lenths have to be recomputed to account for the shortest distance between a Y-connector and one of the outlets of the strings.

This obviously relies on the fact that edge lengths need not represent Euclidean distances; any "distance" function can be represented. A solar farm planner may decide that only "vertical" and "horizontal" cable layouts are allowed, so edge lengths could be measured in the ℓ_1 metric ("Manhattan metric", as in [Luo+21]). The edge lengths may also include constant additive offsets to account for necessary cables used inside the various components or they may represent actual distances in rough terrain. A planner may also decide to prohibit certain connections in the solar farm by removing the corresponding edges from the edge set.

Solar farm planners have further degrees of freedom in the ways to adopt our model: They may wish to use a different set of components, for example they may choose not to use Y-connectors at all. In that case, one layer can be removed from our model. The order of layers can also be changed, for example to account for string inverters instead of central inverters. In that case, the inverters will be represented by a lower layer in our hierarchy.

The use of cable types is again very flexible, even more than with wind farms. The distinction between DC- and AC-cables is arbitrary. One set of cable types may be used for all edges, or each edge may have "its own" set. This can be used to account for power losses (as seen for wind farms) or for prizing different degrees of loads in cables, but also to incorporate installation costs for intermediate vertices. The latter may be more applicable to solar farms than to wind farms, since in wind farms the turbines need to be built anyway.

Not only may the various parts of the model be reinterpreted, but one may also extend the model to incorporate further decision decisions. The literature on wind farms provides some hints: A balancing constraint has been considered in [CP22] which enforces that subtrees must be of roughly equal size. One may also consider using the non-crossing constraints, e. g. [FP18], even though its importance for solar farms has not been addressed. It may even be plausible

that some crossings are allowable since some cables may be installed on racks and others dug into the ground.

The framework to generate benchmark instances also offers a high degree of freedom to adjust parameter and design decisions: Our placement of strings is knowingly chosen to generate somewhat irregular looking instances to model irregular solar farms such as Monte Mele. It is entirely possible, though, to enforce strings being placed in a regular grid. The parameters from Table 5.1 can be also chosen in different ways and different set of cable types may also be considered.

5.7 Conclusion

We have presented one of the first optimization problems to find cost-minimal cable layouts in large-scale solar farms. Our model, SoFaCLaP, is built around a hierarchy of different components used in solar farms. It involves a step cost function obtained from considering a set of cable types from which a solar farm planner may choose and includes the design question where to place the components from a set of given candidate positions. This model generalizes a model previously introduced [Luo+21]. We have shown that SoFaCLaP is strongly NP-hard even under various assumptions. We have translated our model into an MILP formulation and have reported performance results of the MILP solver Gurobi on a set of benchmark instances so that the MILP approach can serve as a baseline for future algorithms. These benchmark instances have been created from a framework proposed in this work with parameters based on real-world components of solar farms. The MILP finds optimal solutions on the smallest of our instances within minutes but struggles to find feasible solutions on the biggest instances within one day of running time. The instances we used in the evaluation, as well as the experimental results are publicly available [GSW22a, GSW22c].

The obvious direction of future research is to develop algorithms to tackle the Solar Farm Cable Layout Problem, after all, that is the whole purpose of defining the model and providing a baseline algorithm. In their Bachelor's Thesis under the co-supervision of this author, Wahl proposes to use Variable Neighborhood Search as a metaheuristic approach [Wah22]. It is noted that this approach "was for the most part not better than the MILP in terms of solution quality [but] it achieves solid solutions" and provides the best known solutions on some of the largest instances [Wah22, p. 41]. Further inspiration for algorithmic approaches to SoFaCLaP may be found in our overview in Section 2.3 of various (classical) optimization problems that bear some degree of resemblance to SoFaCLaP.

6 Summary

We have considered two types of utility-scale power plants with decentralized generators based on renewable energy sources: (offshore) wind farms and solar farms. For both, we address the optimization problem of designing a cost-minimal cable layout to connect the generators to the transformers from where generated power is fed into the transmission grid.

Our literature overview has revealed that the research on cable layout optimization in solar farms is scarce at best. In an attempt to facilitate further algorithmic research, we have proposed an optimization problem based on a layered graph that represents the hierarchy of electrical components in solar farms. We have generated publicly available synthetic benchmark instances and evaluated a Mixed-Integer Linear Program formulation solved by Gurobi on these instances. The results are also publicly available so that any future algorithmic research can build the evaluation on this baseline. Our research on solar farms is complemented by a discussion of design parameters of our optimization problem and the benchmark generation.

Our literature overview has also revealed that wind farms have received a lot of attention in algorithmic research in general but none from a viewpoint of Minimum-Cost Flow Algorithms. Based on a theoretical analysis of the underlying Minimum-Cost Flow Problem to highlight the theoretical complexity and the challenges arising from the characteristic step-cost function, we fill this gap with an adaptation of the classical Negative Cycle Canceling (NCC)-technique. By means of an experimental evaluation on synthetic benchmark instances from the literature, we show that our NCC algorithm is able to provide competitive solutions to approaches using Mixed-Integer Linear Programming and Simulated Annealing, with running times ranging between milliseconds on wind farms with up to 80 turbines and tens of seconds on wind farms with up to 500 turbines. We present how our algorithm can be incorporated into a framework of Iterated Local Search to trade a bit of running time for an improvement in

solution quality. The experimental evaluation is complemented by a case study on the Hornsea One wind farm. Structural and Power Flow Analyses on the cable layouts computed by our NCC algorithm suggest that the cable layouts are suitable for real-world implementation. A discussion of the underlying assumptions in the optimization model and of key steps in our algorithm provides further insights into the applicability of our algorithm to different settings seen in the literature.

With this, we have brought Energy Informatics and Theoretical Computer Science a bit closer together. As it turned out, network flow algorithms can deal with modern day wind turbines more effectively than a courageous knight and a lance can with their medieval counterparts.

Bibliography

[4C20] 4C Offshore Ltd. **Global Offshore Renewable Map**. 2020. URL: https://www.4coffshore.com/offshorewind/index.aspx?lat=53.883&lon=1.922&wfid=UK81 (visited on 02/07/2020).
Cited on page 64.

[AAŞT19] İ. Kuban Altınel, Necati Aras, Zeynep Şuvak, and Z. Caner Taşkın. **Minimum Cost Noncrossing Flow Problem on Layered Networks**. In *Discrete Applied Mathematics* Volume 261, 2019. DOI: 10.1016/j.dam.2018.09.016.
Cited on page 80.

[ABB19] ABB Ltd. **Photovoltaic Plants**. 2019. URL: https://library.abb.com/d/9AKK107492A3277 (visited on 01/20/2022).
Cited on pages 7, 97, 98.

[AGUZ17] Iñaki Arrambide, Pedro Maria García, Juan José Ugartemendia, and Itziar Zubia. **Evaluation of Electrical Losses in MVAC Collector Systems in Offshore Wind Farms**. In *Renewable Energy & Power Quality* Volume 15, 2017. DOI: 10.24084/REPQJ15.296.
Cited on page 75.

[Alp22] Alpiq Holding AG. **Società Agricola Solar Farm 2**. URL: https://www.alpiq.com/power-generation/new-renewable-energy-sources/solar-energy/societa-agricola-solar-farm-2/ (visited on 01/24/2022).
Cited on page 8.

[Amm97] Christine Ammer (editor). **The American Heritage Dictionary of Idioms**. Houghton Mifflin, Boston, Massachusetts, USA, 1997.
Cited on page 1.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. **Network Flows: Theory, Algorithms, and Applications**. Prentice-Hall, Upper Saddle River, New Jersey, USA, 1993.
Cited on pages 14, 15, 16, 17, 22, 24, 29, 80, 81.

[BE12] Michael J. Bannister and David Eppstein. **Randomized Speedup of the Bellman-Ford Algorithm**. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics 2012 (ANALCO '12)*, 2012. DOI: 10.1137/1.9781611973020.6.
Cited on pages 17, 81.

[Bel58] Richard Bellman. **On a Routing Problem**. In *Quarterly of Applied Mathematics* Volume 16, 1958. DOI: 10.1090/qam/102435.
Cited on page 16.

[BEPS14] Martin Bischoff, Hendrik Ewe, Kai Plociennik, and Ingmar Schüle. **Multi-objective Planning of Large-Scale Photovoltaic Power Plants**. In *Operations Research Proceedings 2012*. Springer International Publishing, Cham, Switzerland, 2014. DOI: 10.1007/978-3-319-00795-3_49.
Cited on page 8.

[BHMP00] Pavol Bauer, Sjoerd W. H. De Haan, C. R. Meyl, and Jan T. G. Pierik. **Evaluation of Electrical Systems for Offshore Windfarms**. In *Conference Record of the IEEE Industry Applications Conference*. Volume 3, 2000. DOI: 10.1109/IAS.2000.882070.
Cited on pages 5, 6, 72.

[BMB20] Ratnakar Babu Bollipo, Suresh Mikkili, and Praveen Kumar Bonthagorla. **Critical Review on PV MPPT Techniques: Classical, Intelligent and Optimisation**. In *IET Renewable Power Generation* Volume 14:9, 2020. DOI: 10.1049/iet-rpg.2019.1163.
Cited on page 8.

[Bra+02] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M. Scott Marshall. **GraphML Progress Report Structural Layer Proposal**. In *Graph Drawing*. Springer, Berlin, Heidelberg, Germany, 2002.
Cited on pages 67, 95.

[Bri+03] Jack Brimberg, Pierre Hansen, Keh-Wei Lih, Nenad Mladenović, and Michèle Breton. **An Oil Pipeline Design Problem**. In *Operations Research* Volume 51:2, INFORMS, 2003.
Cited on page 11.

[BRT14] Erika Buson, Roberto Roberti, and Paolo Toth. **A Reduced-Cost Iterated Local Search Heuristic for the Fixed-Charge Transportation Problem**. In *Operations Research* Volume 62:5, 2014. DOI: 10.1287/opre.2014.1288.
Cited on page 51.

[BVMO11] Constantin Berzan, Kalyan Veeramachanenil, James McDermott, and Una-May O'Reilly. **Algorithms for Cable Network Design on Large-Scale Wind Farms**. Technical report, Massachusetts Institute of Technology, 2011. URL: https://alfagroup.csail.mit.edu/sites/default/files/documents/msrp_techreport.pdf (visited on 10/30/2020).
Cited on pages 5, 39, 63, 69, 72.

[Çak+23] Hüseyin Çakmak, Veit Hagenmeyer, Sascha Gritzbach, Pascal Mehnert, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. **Wind Farm Data Supplement for 'Cable Layout Optimization Problems in the Context of Renewable Energy Sources'**. Dataset. 2023. DOI: 10.35097/896.
Cited on pages 39, 54, 71.

[CBP14] Adelaide Cerveira, José Baptista, and Eduardo J. Solteiro Pires. **Optimization Design in Wind Farm Distribution Network**. In *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*. Springer International Publishing, Cham, Switzerland, 2014. DOI: 10.1007/978-3-319-01854-6_12.
Cited on page 5.

[CCCR21] Gustavo Cassiolato, Esdras P. Carvalho, Jose A. Caballero, and Mauro A. S. S. Ravagnani. **Optimization of Water Distribution Networks using a Deterministic Approach**. In *Engineering Optimization* Volume 53:1, Taylor & Francis, 2021. DOI: 10.1080/0305215X.2019.1702980.
Cited on page 11.

[Cer15] Miguel de Cervantes Saavedra. **Don Quijote de la Mancha: Edición Conmemorativa IV Centenario Cervantes**. Edited by Real Academia Española Asociación de Academias de la Lengua Española. 2nd edition. Penguin Random House, Madrid, Spain, 2015.
Cited on page 1.

[Cer61] Miguel de Cervantes Saavedra. **The Ingenious Gentleman Don Quixote de la Mancha**. Translated by Samuel Putnam. Volume 1 of 2. The Folio Society, London, United Kingdom, 1961.
Cited on page 1.

[CFF20] Davide Cazzaro, Martina Fischetti, and Matteo Fischetti. **Heuristic Algorithms for the Wind Farm Cable Routing problem**. In *Applied Energy* Volume 278, 2020. DOI: 10.1016/j.apenergy.2020.115617.
Cited on pages 4, 6.

[CG99] Boris V. Cherkassky and Andrew V. Goldberg. **Negative-Cycle Detection Algorithms**. In *Mathematical Programming* Volume 85:2, 1999. DOI: 10.1007/s101070050058.
Cited on page 17.

[Cha+22] Anatoli Chatzipanagi, Arnulf Jaeger-Waldau, Charles Cleret De Langavant, Simon Letout, Cynthia Latunussa, Aikaterini Mountraki, Aliki Georgakaki, Ela Ince, Anna Kuokkanen, and Drilona Shtjefni. **Clean Energy Technology Observatory: Photovoltaics in the European Union – 2022 Status Report on Technology Development, Trends, Value Chains and Markets**. Technical report JRC130720, Publications Office of the European Union, Luxembourg, Luxembourg, 2022. DOI: 10.2760/812610.
Cited on page 2.

[Chi+13] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. **The Open Graph Drawing Framework (OGDF)**. In *Handbook of Graph Drawing and Visualization*. Edited by R. Tamassia. CRC Press, Boca Raton, Florida, USA, 2013.
Cited on page 99.

[ÇKKH18] Hüseyin Çakmak, Michael Kyesswa, Uwe Kühnapfel, and Veit Hagenmeyer. **eASiMOV – A New Framework for Power Grid Analysis**. In *7th Annual Conference of the KIT Energy Center*, 2018. DOI: 10.13140/RG.2.2.15952.20480.
Cited on page 67.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. **Introduction to Algorithms**. 3rd edition. MIT Press, Cambridge, Massachusetts, USA, 2009.
Cited on pages 16, 17.

[CP22] Davide Cazzaro and David Pisinger. **Balanced Cable Routing for Offshore Wind Farms with Obstacles**. In *Networks* Volume 80:4, 2022. DOI: 10.1002/net.22100.
Cited on pages 4, 106.

[Dah+15] Ouahid Dahmani, Salvy Bourguet, Mohamed Machmoum, Patrick Guerin, Pauline Rhein, and Lionel Josse. **Optimization of the Connection Topology of an Offshore Wind Farm Network**. In *IEEE Systems Journal* Volume 9:4, 2015. DOI: 10.1109/JSYST.2014.2330064.
Cited on page 4.

[DCCA07] Eloy Diaz-Dorado, Camilo Carrillo, José Cidras, and Elena Albo. **Estimation of Energy Losses in a Wind Park**. In *9th International Conference on Electrical Power Quality and Utilisation*, 2007. DOI: 10.1109/EPQU.2007.4424125.
Cited on page 76.

[DIg16] DIgSILENT GmbH. **DIgSILENT PowerFactory User Manual Version 2016**. Gomaringen, Germany, 2016.
Cited on page 70.

[Dij59] Edsger W. Dijkstra. **A Note on Two Problems in Connexion with Graphs**. In *Numerische Mathematik* Volume 1:1, 1959. DOI: 10.1007/BF01386390.
Cited on page 37.

[DO11] Sudipta Dutta and Thomas J. Overbye. **A Clustering Based Wind Farm Collector System Cable Layout Design**. In *IEEE Power and Energy Conference at Illinois (PECI)*, 2011. DOI: 10.1109/PECI.2011.5740480.
Cited on pages 5, 63, 69.

[DS16] Annelies De Corte and Kenneth Sörensen. **An Iterated Local Search Algorithm for Water Distribution Network Design Optimization**. In *Networks* Volume 67:3, 2016. DOI: 10.1002/net.21673.
Cited on pages 11, 51.

[EC20] The European Economic and Social Committee and the Committee of the Regions. **An EU Strategy to Harness the Potential of Offshore Renewable Energy for a Climate Neutral Future**. Technical report COM:2020:741:FIN, The European Commission, 2020. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM:2020:741:FIN (visited on 11/28/2020).
Cited on page 1.

[Eve16] Eric Every. **Using Y Connectors in String Inverter Systems**. 2016. URL: https://solectria.com/blog/using-y-connectors-in-string-inverter-systems-part-i/ (visited on 01/24/2022).
Cited on page 8.

[For56] Lester R. Ford Jr. **Network Flow Theory**. RAND Corporation, Santa Monica, California, USA, 1956.
Cited on page 16.

[FP18] Martina Fischetti and David Pisinger. **Optimizing Wind Farm Cable Routing Considering Power Losses**. In *European Journal of Operational Research* Volume 270:3, 2018. DOI: 10.1016/j.ejor.2017.07.061.
Cited on pages 5, 72, 79, 80, 106.

[FP19] Martina Fischetti and David Pisinger. **Mathematical Optimization and Algorithms for Offshore Wind Farm Design: an Overview**. In *Business & Information Systems Engineering* Volume 61:4, 2019. DOI: 10.1007/s12599-018-0538-0.
Cited on pages 3, 4.

[Gaj16] John W. Gajda. **Solar Farms: Design & Construction**. 2016. URL: https://standards.ieee.org/wp-content/uploads/import/documents/presentations/nesc_workshop__solar_farms-design_construction.pdf (visited on 02/01/2022).
Cited on page 97.

[Gil67] Edgar N. Gilbert. **Minimum Cost Communication Networks**. In *The Bell System Technical Journal* Volume 46:9, 1967. DOI: 10.1002/j.1538-7305.1967.tb04250.x.
Cited on page 10.

[GJ79] Michael R. Garey and David S. Johnson. **Computers and Intractability: a Guide to the Theory of NP-Completeness**. W. H. Freeman & Co., New York City, New York, USA, 1979.
Cited on pages 23, 86.

[GP19] Michel Gendreau and Jean-Yves Potvin. **Tabu Search**. In *Handbook of Metaheuristics*. Edited by M. Gendreau, J.-Y. Potvin. Springer International Publishing, Cham, Switzerland, 2019. DOI: 10.1007/978-3-319-91086-4_2.
Cited on page 52.

[GRG03] Ioannis Gamvros, S. Raghavan, and Bruce Golden. **An Evolutionary Approach to the Multi-Level Capacitated Minimum Spanning Tree Problem**. In *Telecommunications Network Design and Management*. Edited by G. Anandalingam, S. Raghavan. Springer US, Boston, Massachusetts, USA, 2003. DOI: 10.1007/978-1-4757-3762-2_6.
Cited on page 11.

[Gri+18] Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. **Towards Negative Cycle Canceling in Wind Farm Cable Layout Optimization**. In *Energy Informatics* Volume 1:1, 2018. DOI: 10.1186/s42162-018-0030-6.
Cited on pages 19, 20, 27, 79.

[Gri+19] Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. **Engineering Negative Cycle Canceling for Wind Farm Cabling**. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Volume 144 of Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2019. DOI: 10.4230/LIPIcs.ESA.2019.55. arXiv: 1908.02129.
Cited on pages 17, 19, 20, 27, 39, 42, 79.

[Gri+21] Sascha Gritzbach, Hüseyin Çakmak, Pascal Mehnert, Torsten Ueckerdt, and Veit Hagenmeyer. **An Evaluation of Graph Algorithms for the Wind Farm Cable Layout Problem under Electrical Aspects**. In *Proceedings of the 56th International Universities Power Engineering Conference (UPEC)*, 2021. DOI: 10.1109/UPEC50034.2021.9548251.
Cited on page 67.

[GSW22a] Sascha Gritzbach, Dominik Stampa, and Matthias Wolf. **Benchmark Instances for the Solar Farm Cable Layout Problem**. Dataset. 2022. DOI: 10.35097/676.
Cited on pages 9, 95, 107.

[GSW22b] Sascha Gritzbach, Dominik Stampa, and Matthias Wolf. **Solar Farm Cable Layout Optimization as a Graph Problem**. In *Energy Informatics* Volume 5:1, 2022. DOI: 10.1186/s42162-022-00200-z.
Cited on pages 7, 83.

[GSW22c] Sascha Gritzbach, Dominik Stampa, and Matthias Wolf. **Solar Farm Cable Layout Optimization as a Graph Problem (MILP results)**. Dataset. 2022. DOI: 10.35097/683.
Cited on pages 9, 100, 107.

[Gur18] Gurobi Optimization, Inc. **Gurobi Optimizer Reference Manual**. 2018. URL: http://www.gurobi.com.
Cited on pages 43, 58, 99.

[GWRT12] Francisco M. González-Longatt, Peter Wall, Pawel Regulski, and Vladimir Terzija. **Optimal Electric Network Design for a Large Offshore Wind Farm Based on a Modified Genetic Algorithm Approach**. In *IEEE Systems Journal* Volume 6:1, 2012. DOI: 10.1109/JSYST.2011.2163027.
Cited on page 4.

[GWW20] Sascha Gritzbach, Dorothea Wagner, and Matthias Wolf. **Negative Cycle Canceling with Neighborhood Heuristics for the Wind Farm Cabling Problem**. In *Proceedings of the Eleventh ACM International Conference on Future Energy Systems (e-Energy '20)* (Virtual Event, Australia). Association for Computing Machinery, New York City, New York, USA, 2020. DOI: 10.1145/3396851.3397754.
Cited on pages 25, 51, 54, 63, 79.

[Hel21] HELUKABEL GmbH. **Cables and Cable Systems for Photovoltaic Installations**. URL: https://www.helukabel.com/media/publication/de/brochures/pv_20/PV_BROCHURE_Photovoltaik_EN~1.pdf (visited on 12/11/2021).
Cited on page 98.

[Her+17] Alain Hertz, Odile Marcotte, Asma Mdimagh, Michel Carreau, and François Welt. **Design of a wind farm collection network when several cable types are available**. In *Journal of the Operational Research Society* Volume 68, 2017. DOI: 10.1057/s41274-016-0021-6.
Cited on pages 10, 79.

[HHC16] Peng Hou, Weihao Hu, and Zhe Chen. **Optimisation for Offshore Wind Farm Cable Connection Layout using Adaptive Particle Swarm Optimisation Minimum Spanning Tree Method**. In *IET Renewable Power Generation* Volume 10:5, 2016. DOI: 10.1049/iet-rpg.2015.0340.
Cited on page 4.

[Hou+19] Peng Hou, Jiangsheng Zhu, Kuichao Ma, Guangya Yang, Weihao Hu, and Zhe Chen. **A Review of Offshore Wind Farm Layout Optimization and Electrical System Design Methods**. In *Journal of Modern Power Systems and Clean Energy*, 2019. DOI: 10.1007/s40565-019-0550-5.
Cited on pages 3, 4.

[HWW08] Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. **Multigraph Realizations of Degree Sequences: Maximization is Easy, Minimization is Hard**. In *Operations Research Letters* Volume 36:5, 2008. DOI: 10.1016/j.orl.2008.05.004.
Cited on page 87.

[IBM17] IBM. **IBM ILOG CPLEX 12.8 User's Manual**. 2017.
Cited on page 43.

[Jen20] Marc Jenne. **Minimum-Cost Flow Algorithms for the Wind Farm Cabling Problem**. Bachelor's thesis. Karlsruhe Institute of Technology (KIT), 2020. URL: https://i11www.iti.kit.edu/_media/teaching/theses/ba-jenne-20.pdf (visited on 12/16/2022).
Cited on page 80.

[KÇKH17] Michael Kyesswa, Hüseyin Çakmak, Uwe Kühnapfel, and Veit Hagenmeyer. **A Matlab-Based Dynamic Simulation Module for Power System Transients Analysis in the eASiMOV Framework**. In *Proceedings of the UKSim-AMSS 11th European Modelling Symposium on Computer Modelling and Simulation (EMS 2017)*, 2017. DOI: 10.1109/EMS.2017.36.
Cited on page 67.

[KR18] Thomas Kirchartz and Uwe Rau. **What Makes a Good Solar Cell?** In *Advanced Energy Materials* Volume 8:28, 2018. DOI: 10.1002/aenm.201703385.
Cited on page 8.

[KV00] Bernhard Korte and Jens Vygen. **Combinatorial Optimization: Theory and Algorithms**. Volume 21 of Algorithms and combinatorics. Springer, Berlin, Germany, 2000.
Cited on page 16.

[LDM21] Markus Lerch, Mikel De-Prada-Gil, and Climent Molins. **A Metaheuristic Optimization Model for the Inter-Array Layout Planning of Floating Offshore Wind Farms**. In *International Journal of Electrical Power & Energy Systems* Volume 131, 2021. DOI: 10.1016/j.ijepes.2021.107128.
Cited on pages 10, 79.

[Liu+19] Tao Liu, Xu Yang, Wenjie Chen, Yang Li, Yang Xuan, Lang Huang, and Xiang Hao. **Design and Implementation of High Efficiency Control Scheme of Dual Active Bridge Based 10 kV/1 MW Solid State Transformer for PV Application**. In *IEEE Transactions on Power Electronics* Volume 34:5, 2019. DOI: 10.1109/TPEL.2018.2864657.
Cited on page 8.

[Lju21] Ivana Ljubić. **Solving Steiner Trees: Recent Advances, Challenges, and Perspectives**. In *Networks* Volume 77:2, 2021. DOI: 10.1002/net.22005.
Cited on page 10.

[LMS19] Helena Ramalhinho Lourenço, Olivier C. Martin, and Thomas Stützle. **Iterated Local Search: Framework and Applications**. In *Handbook of Metaheuristics*. Edited by M. Gendreau, J.-Y. Potvin. Springer International Publishing, Cham, Switzerland, 2019. DOI: 10.1007/978-3-319-91086-4_5.
Cited on pages 51, 52.

[LRWW17] 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)* (Shatin, Hong Kong). Association for Computing Machinery, New York City, New York, USA, 2017. DOI: 10.1145/3077839.3077843.
Cited on pages 4, 5, 6, 7, 19, 38, 39, 44, 49, 54, 61, 63, 79, 105, 135.

[LS21] LS ELECTRIC Co., Ltd. **[PV] PV Combiner Box**. 2021. URL: https://www.ls-electric.com/products/category/Smart_Power_Solution/PV_System (visited on 01/20/2022).
Cited on page 97.

[Luo+21] Zhixing Luo, Hu Qin, T. C. Edwin Cheng, Qinghua Wu, and Andrew Lim. **A Branch-and-Price-and-Cut Algorithm for the Cable-Routing Problem in Solar Power Plants**. In *INFORMS Journal on Computing* Volume 33:2, INFORMS, 2021. DOI: 10.1287/ijoc.2020.0981.
Cited on pages 9, 106, 107.

[Mer19] Konrad Mertens. **Photovoltaics: Fundamentals, Technology, and Practice.** 2nd edition. Wiley, Chichester, West Sussex, United Kingdom, 2019.
Cited on pages 2, 7.

[Moo59] Edward F. Moore. **The Shortest Path Through a Maze**. In *Proceedings of an International Symposium on the Theory of Switching 1957*. Harvard University Press, Cambridge, Massachusetts, USA, 1959.
Cited on pages 16, 17.

[MS07] Philippe Mahey and Mauricio C. de Souza. **Local Optimality Conditions for Multicommodity Flow Problems with Separable Piecewise Convex Costs**. In *Operations Research Letters* Volume 35:2, 2007. DOI: 10.1016/j.orl.2006.02.005.
Cited on page 6.

[Nex08] Nexans. **Submarine Power Cables**. 2008. URL: https://www.nexans.de/Germany/2010/NEX_SubmPowCables_mai08_1.pdf (visited on 11/06/2020).
Cited on pages 69, 70.

[OCL18] Camilo Ortiz-Astorquiza, Ivan Contreras, and Gilbert Laporte. **Multi-Level Facility Location Problems**. In *European Journal of Operational Research* Volume 267:3, 2018. DOI: 10.1016/j.ejor.2017.10.019.
Cited on page 10.

[OM00] Adam Ouorou and Philippe Mahey. **A Minimum Mean Cycle Cancelling Method for Nonlinear Multicommodity Flow Problems**. In *European Journal of Operational Research* Volume 121:3, 2000. DOI: 10.1016/S0377-2217(99)00050-8.
Cited on page 6.

[OSM20] OpenStreetMap contributors. **Planet Dump**. 2020. URL: https://planet.openstreetmap.org (visited on 02/08/2020).
Cited on page 63.

[PC19] Juan-Andrés Pérez-Rúa and Nicolaos Antonio Cutululis. **Electrical Cable Optimization in Offshore Wind Farms—A Review**. In *IEEE Access* Volume 7, 2019. DOI: 10.1109/ACCESS.2019.2925873.
Cited on pages 4, 5.

[Pér22] Juan-Andrés Pérez-Rúa. **Solver-Free Heuristics to Retrieve Feasible Points for Offshore Wind Farm Collection System**. In *Engineering Optimization*, Taylor & Francis, 2022. DOI: 10.1080/0305215X.2022.2108027.
Cited on page 80.

[QAAM07] Gustavo Quinonez-Varela, Graham W. Ault, Olimpo Anaya-Lara, and James R. McDonald. **Electrical Collector System Options for Large Offshore Wind Farms**. In *IET Renewable Power Generation* Volume 1:2, 2007. DOI: 10.1049/iet-rpg:20060017.
Cited on page 4.

[Rez22] Mohsen Rezapour. **Algorithmic approaches for network design with Facility Location: a Survey**. In *AUT Journal of Mathematics and Computing* Volume 3:2, Amirkabir University of Technology, 2022. DOI: 10.22060/ajmc.2022.21392.1086.
Cited on pages 10, 11.

[Rot+70] B. Rothfarb, H. Frank, D. M. Rosenbaum, Kenneth Steiglitz, and Daniel J. Kleitman. **Optimal Design of Offshore Natural-Gas Pipeline Systems**. In *Operations Research* Volume 18:6, 1970. DOI: 10.1287/opre.18.6.992.
Cited on page 11.

[ŞAA21] Zeynep Şuvak, İ. Kuban Altınel, and Necati Aras. **Minimum Cost Flow Problem with Conflicts**. In *Networks* Volume 78:4, 2021. DOI: 10.1002/net.22021.
Cited on page 80.

[San20] Priya Sanjay. **With 2,245 MW of Commissioned Solar Projects, World's Largest Solar Park is Now at Bhadla**. 2020. URL: https://mercomindia.com/world-largest-solar-park-bhadla/ (visited on 01/25/2022).
Cited on pages 2, 8.

[Sch12] Alexander Schrijver. **On the History of the Shortest Path Problem**. In *Documenta Mathematica* Extra Volume Optimization Stories, 2012.
Cited on page 17.

[Sed+16] Mohsen Sedighi, Mohammad Moradzadeh, Osman Kukrer, Murat Fahrioglu, and Lieven Vandevelde. **Optimal Electrical Interconnection Configuration of Off-Shore Wind Farms**. In *Journal of Clean Energy Technologies* Volume 4:1, 2016. DOI: 10.7763/JOCET.2016.V4.255.
Cited on page 4.

[She11] David J. Sheskin. **Handbook of Parametric and Nonparametric Statistical Procedures**. 5th edition. Chapman & Hall/CRC, Boca Raton, Florida, USA, 2011. DOI: 10.1201/9780429186196.
Cited on pages 17, 18.

[SMA20] SMA Solar Technology AG. **SMA String-Combiner**. 2020. URL: https://files.sma.de/downloads/DC-CMB-U-DEN1834-V16web.pdf (visited on 01/20/2022).
Cited on page 97.

[SMG08] Mauricio C. de Souza, Philippe Mahey, and Bernard Gendron. **Cycle-Based Algorithms for Multicommodity Network Flow Problems with Separable Piecewise Convex Costs**. In *Networks* Volume 51:2, 2008. DOI: 10.1002/net.20208.
Cited on page 6.

[Sol] SolarBOS, Inc. **Standard Recombiners**. URL: http://www.solarbos.com/Standard-Recombiners (visited on 01/20/2022).
Cited on page 97.

[Sta22] Dominik Stampa. **Theory and Algorithms of the Solar Farm Cable Layout Problem**. Master's thesis. Karlsruhe Institute of Technology (KIT), 2022. URL: https://i11www.iti.kit.edu/_media/teaching/theses/ma-stampa-22.pdf (visited on 04/03/2022).
Cited on page 83.

[Tel+20] Thomas Telsnig, Davide Magagna, Efstathios Peteves, Evangelos Tzimas, Beatrice Plazzotta, and Francesco Pasimeni. **Facts and Figures on Offshore Renewable Energy Sources in Europe**. Technical report JRC121366, The European Commission, 2020.
Cited on pages 1, 10, 69.

[TSMA17] Madjid Tavana, Francisco J. Santos Arteaga, Somayeh Mohammadi, and Moslem Alimohammadi. **A Fuzzy Multi-Criteria Spatial Decision Support System for Solar Farm Location Planning**. In *Energy Strategy Reviews* Volume 18, 2017. DOI: 10.1016/j.esr.2017.09.003.
Cited on page 8.

[UN15] United Nations General Assembly. **Transforming our World: the 2030 Agenda for Sustainable Development**. Resolution A/RES/70/1, 2015. URL: https://undocs.org/en/A/RES/70/1 (visited on 01/16/2023).
Cited on page iii.

[Vol+13] Marcus G. Volz, Marcus Brazil, Charl J. Ras, Konrad J. Swanepoel, and Doreen A. Thomas. **The Gilbert Arborescence Problem**. In *Networks* Volume 61:3, 2013. DOI: 10.1002/net.21475.
Cited on page 10.

[VSA14] Pedro Santos Valverde, António J. N. A. Sarmento, and Marco Alves. **Offshore Wind Farm Layout Optimization—State of the Art**. In *Journal of Ocean and Wind Energy* Volume 1:1, 2014.
Cited on page 3.

[Wah22] Leonie Wahl. **Variable Neighborhood Search for the Solar Farm Cable Layout Problem**. Bachelor's thesis. Karlsruhe Institute of Technology (KIT), 2022. URL: https://i11www.iti.kit.edu/_media/teaching/theses/ba-wahl-22.pdf (visited on 12/19/2022).
Cited on page 107.

[Weg20] Franziska Wegner. **Combinatorial Problems in Energy Networks – Graph-theoretic Models and Algorithms**. PhD thesis. Karlsruhe Institute of Technology (KIT), 2020. DOI: 10.5445/IR/1000120612.
Cited on page 14.

[Yen70] Jin Y. Yen. **An Algorithm for Finding Shortest Routes from all Source Nodes to a Given Destination in General Networks**. In *Quarterly of Applied Mathematics* Volume 27:4, Brown University, 1970. DOI: 10.1090/qam/253822.
Cited on pages 17, 81.

[Zha+17] Haoran Zhang, Yongtu Liang, Jing Ma, Chen Qian, and Xiaohan Yan. **An MILP Method for Optimal Offshore Oilfield Gathering System**. In *Ocean Engineering* Volume 141, 2017. DOI: 10.1016/j.oceaneng.2017.06.011.
Cited on page 11.

[Zia19] Hanna Ziady. **The World's Largest Offshore Wind Farm is Nearly Complete. It can Power 1 Million Homes**. In *Cable News Network (CNN)*, 2019. URL: https://edition.cnn.com/2019/09/25/business/worlds-largest-wind-farm/index.html (visited on 02/10/2020).
Cited on page 63.

[ZMT11] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. **MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education**. In *IEEE Transactions on Power Systems* Volume 26:1, 2011. DOI: 10.1109/TPWRS.2010.2051168.
Cited on page 67.

List of Figures

List of Tables

List of Acronyms

AC	Alternating Current Used on pages 7, 84, 85, 106
CLI	Command-Line Interface Used on page 68
DC	Direct Current Used on pages 7, 8, 84, 85, 106
eASiMOV	Software Framework for Electrical Grid Analysis, Simulation, Modeling, Optimization and Visualization Used on pages 67, 78
HV	High Voltage Used on pages 68, 78
ILS	Iterated Local Search Used on pages iv, 7, 11, 12, 19, 50–66, 72, 73, 77, 78, 80, 81, 109, 125, 127
LV	Low Voltage Used on pages 68, 78
MV	Medium Voltage Used on pages 68, 70
MILP	Mixed-Integer Linear Program Used on pages iv, 4–7, 9, 11, 12, 19, 25–27, 38, 43–45, 47, 48, 50, 51, 54, 57–59, 63, 64, 67, 69, 71–73, 75–78, 80, 81, 83, 85, 94, 95, 99–101, 104–107, 109, 125, 134

NCC	Negative Cycle Canceling Used on pages iv, 2, 6, 7, 12, 15–17, 19, 25, 27, 30, 31, 38, 39, 42, 44–57, 59–69, 71–73, 75–81, 109, 110, 125, 127, 134
SA	Simulated Annealing Used on pages 4, 6, 12, 38, 49, 50, 54, 61, 62, 81, 109, 125
SoFaCLaP	Solar Farm Cable Layout Problem Used on pages 9, 12, 83–87, 89, 90, 92–96, 98–100, 105, 107, 126
WCP	Wind Farm Cabling Problem Used on pages 12, 19, 21, 23–26, 31, 38, 39, 50, 55, 56, 67, 71, 73, 79, 80, 87, 105, 135
PV	Photovoltaic Used on pages 2, 7
UK	United Kingdom Used on page 1

List of Symbols

Set of Numbers

$\mathbb{N}$ The set of all natural numbers, without the number zero
Used on pages 13, 14, 20, 21, 23, 28, 84, 86–88, 134

$\mathbb{N}_0$ The set of all natural numbers, including the number zero
Used on pages 14, 20–22

$[k]$ The set of natural numbers $\{1, \ldots, k\}$
Used on pages 13, 21, 23, 24, 84

$\mathbb{Z}$ The set of all integral numbers
Used on page 35

$\mathbb{Q}$ The set of all rational numbers
Used on pages 87, 88

$\mathbb{Q}_{>0}$ The set of all positive (> 0) rational numbers
Used on page 88

$\mathbb{R}$ The set of all real numbers
Used on pages 14–16, 20, 21, 25, 28, 85, 88, 133

$\mathbb{R}_{\geq 0}$ The set of all non-negative (≥ 0) real numbers
Used on pages 15, 20, 21, 83

Graph Theory

C A cycle
Used on pages 13, 16, 29, 34, 37

$\mathcal{C}$ A set of cycles
Used on page 36

E The set of edges of a graph
Used on pages 13–17, 20–22, 24–26, 28, 29, 33, 34, 36, 83–85, 94, 95, 99, 102, 132, 133

e An edge $e = (u, v) \in E$ with startvertex u and endvertex v
Used on pages 13, 15, 16, 20–22, 24–29, 33–36, 66, 84, 85, 133

$\bar{e}$ The edge (v, u) for an edge $e = (u, v)$
Used on pages 13, 28, 33, 66

G A directed graph with vertex set V and edge set E
Used on pages 13–17, 20, 22, 25, 28, 29, 33, 83, 134

len A function specifying the length of edges in a graph, usually inferred from the coordinates of the respective start- and endvertex
Used on pages 20, 21, 26–28, 33, 37, 38, 79, 83, 85, 94, 133

P A path
Used on pages 13, 37

V The set of vertices of a graph
Used on pages 13–17, 20, 22, 28, 34–36, 83–85, 94–97, 99, 102, 132, 133

u A vertex $u \in V$
Used on pages 13–17, 20, 22–24, 26, 28–35, 37, 38, 81, 85, 94, 95, 132, 133

v A vertex $v \in V$
Used on pages 13–17, 20, 22–24, 26, 28–38, 81, 84, 85, 94, 95, 132, 133

w A vertex $w \in V$
Used on pages 14, 34, 35, 84, 85, 94, 133

x A vertex $x \in V$
Used on pages 22, 34

y A vertex $y \in V$
Used on page 22

W A walk
Used on pages 13, 33–36

Network Flows

c The cost $c(x)$ for sending x units of flow across one unit of length, obtained from choosing the cheapest cable type with sufficient capacity from a given set of cable types
Used on pages 21–23, 25, 27, 28, 85, 133

c_K The cost per unit of length of a cable type κ
Used on pages 21, 23, 25–27, 33, 39, 79, 84, 94, 98, 133

cap_E The capacity $\text{cap}_E(u, v)$ on flow from u to v
Used on pages 14–16, 20, 22, 24, 25, 84, 85

cap_K The capacity of a cable type κ in units of turbine or string production
Used on pages 21, 23, 25–27, 39, 69, 84, 95, 98, 133

cap_V The capacity $\text{cap}_V(u)$ of a vertex u
Used on pages 14, 15, 84, 85

cost The general-purpose cost $\text{cost}((u, v), x)$ of routing $x \in \mathbb{R}$ units of flow along the edge (u, v) from u to v; later replaced by $c(x) \cdot \text{len}(e)$
Used on page 15

Cost The total cost $\text{Cost}(f)$ of a flow f, computed as the sum of the cost across all edges
Used on pages 15, 21, 25, 29, 85

f A flow in a graph; by convention, $f(u, v) > 0$ means that flow leaves u and enters v
Used on pages 14–16, 20–22, 24–29, 33, 36, 85, 94, 95, 102, 133

f_{net} The net flow $f_{\text{net}}(u)$ at a vertex u for a given flow f is the difference from the total flow leaving u to the total flow entering u.
Used on pages 14, 15, 20, 22, 26, 85, 133

K A set of cable types out of which one cable type can be chosen per edge
Used on pages 21, 25–27, 33, 84, 94, 95

κ A cable type with a capacity $\text{cap}_K(\kappa)$ and a cost $c_K(\kappa)$ per unit of length
Used on pages 21, 25–27, 69, 84, 94, 95

p The production $p(u)$ of a source vertex $u \in V^{\uparrow}$; equal to the negative net flow of any flow in the graph
Used on pages 14, 15, 20, 84

$V^{\uparrow}$ The set of source vertices of a graph
Used on pages 14, 15, 133

$V^{\downarrow}$ The set of target vertices of a graph
Used on pages 14, 15

Solar and Wind Farms

cap_{Sub} The capacity $\text{cap}_{\text{Sub}}(v)$ of a substation w
Used on pages 20, 22, 26, 29, 70

ℓ_5 The lower capacity of inverters in a solar farm
Used on pages 84, 85, 94

u The upper capacity of vertices in a solar farm; may carry a vertex as argument, e. g., $u(w)$, or an index for a layer of vertices, e. g., u_3 for the upper capacity of combiner boxes
Used on pages 84, 86, 94, 133

V_C The set of candidate positions for combiner boxes in a solar farm
Used on pages 83, 84, 97

V_I The set of candidate positions for inverters in a solar farm
Used on pages 83–85, 94, 97

V_R The set of candidate positions for recombiner boxes in a solar farm
Used on pages 83, 84, 97

V_S The set of substations in a wind farm or the set of strings in a solar farm
Used on pages 20, 22, 26, 28, 83–85, 94, 97, 99, 102

V_T The set of turbines in a wind farm or the set of candidate positions for transformers in a solar farm
Used on pages 20, 22, 24, 26, 28, 83–85, 94, 95, 97

V_Y The set of candidate positions for Y-connectors in a solar farm
Used on pages 83, 84, 97

Other Concepts used in Algorithms

γ The edge weights in a residual graph for a given flow and a given $\Delta \in \mathbb{N}$, stating how the cost of the flow on an edge changes if Δ additional units of flow are sent over that edge
Used on pages 28, 29, 33–36

Δ A natural number representing how many additional units of flow would be sent over an edge.
Used on pages 28–33, 36–39, 53, 64, 66, 125, 134

dist_i A distance label used in the Bellman-Ford algorithm
Used on pages 17, 33–36

$\mathcal{L}(G)$ The linegraph of a graph G as a representation of adjacency of edges
Used on pages 13, 33, 66

L A subgraph of the linegraph of a residual graph used for Negative Cycle Canceling
Used on pages 33–36, 66

parent_i A parent pointer used in the Bellman-Ford algorithm to identify the vertex and edge responsible for a certain distance label
Used on pages 17, 34

R The residual graph used in Negative Cycle Canceling
Used on pages 16, 28, 29, 33–36, 66

s A virtual substation that connects all substations in the residual graph of a wind farm graph ("*super substation*")
Used on pages 24, 28–30

x A binary variable in a Mixed-Integer Linear Program
Used on pages 25–27, 94, 95

Miscellaneous

$\cos \phi$ A power factor
Used on page 70

$\mathcal{N}_i$ A set of benchmark instances used for the evaluation for algorithms for Wind Farm Cabling Problem, originally proposed in [LRWW17]
Used on pages 39, 40, 43–50, 56–59, 61–63, 72, 73, 105

P_{Trb} The rated active power of a turbine
Used on page 70

$S_{\text{Trb}}, S_{\text{Sub}}$ The rated apparent power of a turbine or substation
Used on pages 69, 70