Karlsruhe Institute of Technology

Variable Neighborhood Search for the Solar Farm Cable Layout Problem

Bachelor Thesis of

Leonie Wahl

At the Department of Informatics
Institute of Theoretical Informatics

Reviewers:	PD Dr. Torsten Ueckerdt
	T.T.-Prof. Dr. Thomas Bläsius
Advisors:	Sascha Gritzbach
	Max Göttlicher

Time Period: 01 July 2022 – 02 November 2022

Statement of Authorship

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, November 2, 2022

Abstract

The Solar Farm Cable Layout Problem is an optimization problem concerned with finding a cable layout for a given solar farm resulting in the least amount of costs, while adhering to different constraints such as the capacity of components. In previous works, an MILP formulation and a heuristic method have been proposed for its solution. This thesis explores a third option of a metaheuristic method next to the existing exact and heuristic methods.

We propose an application of the metaheuristic Variable Neighborhood Search (VNS) to the Solar Farm Cable Layout Problem. The performance of our VNS is tested and compared to the existing MILP and heuristic. Overall, our VNS is able to find better solutions in less time than the heuristic on the same instances. VNS finds solutions for some instances the MILP could not solve within its given time limit, making them the best available ones on those instances so far.

Deutsche Zusammenfassung

Das Verkablungsproblem in Solarparks ist das Optimierungsproblem, für einen gegebenen Solarpark die kostengünstigste Verkablung zu finden, die zugleich verschiedene Bedingungen, wie das Einhalten von Kapazitäten der Komponenten, erfüllt. In früheren Arbeiten wurde bereits eine MILP Formulierung und eine heuristische Methode für dessen Lösung entworfen. Diese Arbeit beschäftigt sich mit einem metaheuristischem Lösungsansatz als dritte Option neben den bestehenden exakten und heuristischen Methoden.

Wir stellen eine Anwendung der Metaheuristik Variable Neighborhood Search (VNS) für das Verkablungsproblem in Solarfarmen vor. Die Performance dieser Metaheuristik wird getestet und verglichen mit der existierenden MILP Formulierung und Heuristik. Insgesamt findet VNS auf denselben Instanzen bessere Lösungen in kürzerer Zeit als die Heuristik. VNS findet Lösungen für manche der Instanzen, die die MILP Formulierung nicht in vorgegebener Zeit lösen konnte. Auf diesen Instanzen sind die Lösungen von VNS daher die besten bisher gefundenen.

Contents

1 Introduction **1**

2 Preliminaries **3**
2.1 Graphs . . . 3
2.1.1 Flow networks . . . 3
2.2 Variable Neighborhood Search . . . 4
2.2.1 Variants of VNS . . . 5
2.3 Components of a solar farm . . . 6

3 Related Work **9**
3.1 Cabling problem in solar farms . . . 9
3.2 Similar Problems . . . 10
3.2.1 Wind Farm Cabling Problem . . . 10
3.2.2 Steiner problem in graphs . . . 11
3.2.3 Capacitated Multi-level Facility Location Problem . . . 12
3.3 Contribution and Outline . . . 13

4 The Solar Farm Cable Layout Problem **15**
4.1 Modelling the layout of a solar farm . . . 15
4.1.1 Modelling the components . . . 15
4.1.2 Modeling the possible cables . . . 16
4.2 The Solar Farm Cable Layout Problem . . . 16
4.2.1 Complexity of the Problem . . . 17
4.3 Existing solution methods . . . 18

5 Algorithm **19**
5.1 Components of VNS . . . 19
5.2 Representation . . . 20
5.3 Evaluation of solutions . . . 21
5.4 Initial solution . . . 22

6 Experiments **25**
6.1 Solar farm instances and cable types . . . 25
6.2 Comparison of variants of VNS . . . 26
6.2.1 Initial solution . . . 26
6.2.2 Local search policy . . . 27
6.2.3 Number of neighborhood structures . . . 28
6.3 Comparison of VNS and Heuristic . . . 31
6.4 Combining VNS and Heuristic . . . 36
6.5 Summary and Discussion . . . 38

7 Conclusion **41**

Bibliography **43**

1. Introduction

In its fifth assessment report in 2014, the Intergovernmental Panel on Climate Change, IPCC, declared the energy supply sector as the largest sectoral contributor to global greenhouse gas emissions, being responsible for an approximated 35% of all emissions [IPC14]. Therefore, the transformation of current energy systems plays a key role in the mitigation of climate change. Renewable energy sources such as solar power have the potential to advance this process.

Photovoltaic (PV) power systems generate usable solar power by the means of photovoltaics, the direct conversion of sunlight into electricity by semi-conductive materials. Solar farms or photovoltaic power stations are large-scale, centralized PV power systems, that provide solar power at the utility level. In this regard they differ from smaller, distributed PV power systems, such as PV systems mounted on the rooftops of residential buildings, which are designed for the supply of specific local users.

The power in a solar farm is generated by multiple photovoltaic cells, which are encapsulated in PV modules. Multiple PV modules in turn are connected to form so-called strings. Solar farms are grid-connected PV systems, meaning that their generated power is fed into an electricity grid. For this purpose, the generated power must first be passed through inverters, which convert the direct current from the strings into alternating current, the form of electrical current delivered by the electricity grid. Finally the generated power must be directed to transformers before being fed into the grid. Transformers step up the voltage to the level the electricity grid operates at. Besides these three types of key components, a solar farm can contain several other components for parallel or serial connection of strings. The components of a solar farm are placed in layers and connected by cables running between those layers.

The costs of a solar farm are composed of various factors, ranging from the costs of the components and installation to those of the operation and maintenance of the solar farm. The total installation costs of a solar farm are typically divided into the costs for the PV modules themselves and all other costs, which are summarized by the term *Balance of System* (BoS) costs. BoS costs include for example the costs for other hardware, such as inverters, but also the costs for cabling, the actual labor work involved in the installation or for site preparation. Over the last decades, prices for photovoltaic modules have steadily declined [VMB+20]. At the same time, the balance of system (BoS) costs have declined at a slower rate [EABB18]. Therefore, it is not surprising that the contribution of BoS costs to the total installation costs of a solar farm is increasing, jumping from making up about

50% of total system costs in 2016 to estimated 65% in 2020, when excluding the inverter costs [IRE21]. The reduction of BoS costs as a whole is becoming increasingly important for furthering the decrease of costs of PV power systems, with the reduction of cabling costs as one factor of it.

In his master's thesis in 2022 [Sta22], Stampa formulated the design of a solar farm cable layout as an optimization problem in the form of a minimum cost flow problem. The author modeled a solar farm as a directed layered graph with a step cost function representing the costs of different cable types. For the problem's solution, the author proposed a mixed-integer linear program (MILP) and a heuristic method. In the cases in which both methods found a solution for a given instance, the MILP exceeded the heuristic method in terms of solution quality, but lagged behind in terms of runtime. With increasing instance sizes, both methods, but especially the MILP, struggled to find any feasible solution at all. Unfortunately, real-life solar farm instances tend to be large-scale.

The problem of optimizing cable layout costs in a solar farm, as formalized by [Sta22], is an NP-hard problem. As an exact and a heuristic method have already been proposed for its solution, a third option lies in the deployment of metaheuristics. As with heuristic methods in general, metaheuristics trade solution optimality for maintaining reasonable runtime. Their application becomes especially favorable in cases where the problem at hand is NP-hard and practical instances too large for the usage of exact methods. In contrast to heuristic methods such as the one deployed in [Sta22], metaheuristics are problem independent and can be tailored to any specific problem at hand.

Variable Neighborhood Search is a metaheuristic that builds upon the heuristic method local search. Local search algorithms attempt to find the optimal solution to a given problem by approaching it incrementally. In each step, starting from an initial solution, the algorithm searches the current solution's close vicinity or so-called neighborhood for a better solution and moves to it. If no such improvement can be made, the algorithm stops. Of course, if the algorithm stops, there is no guarantee that it has done so because it found the *global optimum*, rather than only a local one. Several metaheuristics have been designed to avoid the problem of getting stuck at a local optimum that the classic local search suffers from. Variable Neighborhood Search deals with this problem by using several neighborhoods structures instead of only one. If no improvement can be made in respect to one neighborhood, the algorithm moves to a different neighborhood, in the hopes of escaping the local optimum.

We propose and evaluate the application of Variable Neighborhood Search to the cabling problem in solar farms. The performance of our Variable Neighborhood Search algorithm is judged by comparing it to the existing MILP formulation and heuristic method.

2. Preliminaries

This chapter serves as a brief overview of the foundations of the thesis, from graphs and flow networks to metaheuristics and VNS and the components of a solar farm.

2.1 Graphs

A *directed graph* is a pair $G = (V, E)$ of a set V of *vertices* and a set $E \subseteq V^2$ of *edges* between them. We picture vertices by drawing a dot for every vertex $v \in V$ and edges by drawing an arrow pointing from v to w for every $(v, w) \in E$.

A *path* from some vertex v_1 to some vertex v_k in graph G is a tuple $(v_1, v_2, \ldots, v_k)$ with $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, k-1$. A *cycle* is a path $c = (v_1, \ldots, v_k)$ with $v_1 = v_k$. A graph is *acyclic* if it does not contain cycles.

An *arborescence* is a directed acyclic graph that contains a *root* vertex v and for every other vertex w exactly one path from v to w. A *directed forest* is a directed graph that contains one or multiple arborescences with no path between any of them.

An *anti-arborescence* or *in-tree* is a directed acyclic graph with a root vertex v and for every other vertex w exactly one path *from* w to v. An *in-forest* is a forest of anti-arborescences instead of arborescences.

2.1.1 Flow networks

Let $G = (V, E)$ be a directed graph, $s, t \in V$ two fixed vertices called the *source* s and *sink* t, and $c\colon E \to \mathbb{R}^+$ a *capacity function*. The tuple $N := (G, s, t, c)$ is then called a *flow network*. A function $f\colon E \to \mathbb{R}^+$ defines a *flow* in N if it satisfies the following conditions:

(F1) $f(e) \leq c(e)$ for all $e \in E$,

(F2) $\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w)$ for all $v \in V \setminus \{s, t\}$.

The *total value* of f or the *total flow* in N is given by

$$F := \sum_{(s,w)\in E} f(s, w) - \sum_{(u,s)\in E} f(u, s)\,.$$

In the following, the *inflow* at a vertex $v \in V$ is defined as

$$\mathrm{in}_f(v) = \sum_{(u,v)\in E} f(u, v)\,.$$

2.2 Variable Neighborhood Search

Many optimization problems of practical importance, such as the Solar Farm Cable Layout Problem, are NP-hard and practical instances often too large for the usage of exact methods. In such cases, a possible resort lies in the design of heuristic methods. Heuristic methods trade optimality in exchange for better runtime, with the goal of finding a suboptimal (but preferably still decent) solution within reasonable time.

Metaheuristic methods are frameworks for designing such heuristics, providing general templates that can be tailored to any specific problem at hand. *Variable Neigborhood Search* (VNS) is a metaheuristic method proposed by Hansen and Mladenović in 1997 [MH97]. The following section is based on the description of VNS by Hansen, Mladenović and Pérez [MHP08].

The heuristic method *local search* attempts to solve an optimization problem by incrementally moving towards the optimal solution. For this purpose, a *neighborhood relation* N must be defined on the solution space, with $N(x)$ designating all neighbors of a given solution x. Intuitively, neighboring solutions should be in some way "similar" or "close" to another. Local search explores the solution space by starting from an initial solution and repeatedly moving to a neighboring better solution. There are two main strategies for picking the next neighbor: *First Improvement*, which draws neighbors from the neighborhood $N(x)$ of the current solution x in some assigned order and picks the first better neighbor it comes across, and *Best Improvement*, which chooses the best solution out of the entire neighborhood. The algorithm stops if no more improving step can be made. Of course, if the algorithm stops, there is no guarantee that it has found the *global optimum*, rather than only a local one.

VNS extends on the basic local search method, by searching for improvements not only in one, but several, successive neighborhoods of the current solution. As shown in Figure 2.1, VNS iteratively picks an initial neighbor in each neighborhood (a process referred to as "Shaking" in the literature) and uses it as a starting point for a basic local search as illustrated above. VNS repeats this process for all neighborhoods until the local search succeeds in finding a new and better local minimum.

The local search enables VNS to descend into found local minima, the shaking enables it to break out of them. The inclusion of several neighborhoods allows VNS to skip to a different structure if one neighborhood structure does not lead to further improvements.

Algorithm 2.1 provides an outline of Variable Neighborhood Search as used in this thesis. This algorithm is also referred to as *Basic VNS* in [MHP08]. For implementing this template, the design of several components has to be considered, which are illustrated in more detail below.

An *objective function* is a real-valued function f, that evaluates the quality of solutions to a given problem instance. For minimization problems, a solution x is *optimal* if $f(x) \leq f(x')$ for all other solutions x'. When designing a heuristic method, we must decide if we only allow the construction of feasible solutions or if we include infeasible ones as well. In the latter case, f adds a penalty to infeasible solutions.

Furthermore, a family of neighborhood relations $(\mathcal{N}_k)_{k\in\{1,\ldots,n\}}$ must be defined on the solution space, with $\mathcal{N}_k(x)$ the set of *k-neighbors* for a solution x and $n \in \mathbb{N}$.

Variable Neighborhood Search expects an *initial solution* x as a starting point. Its construction can be conducted completely at random or guided by problem-specific, heuristic information about promising starting points.

For each $k = 1, \ldots, k_{\max}$, with $k_{\max}$ selectable and part of the input, the algorithm conducts a *shaking phase* for obtaining an initial k-neighbor $x' \in \mathcal{N}_k(x)$ of the current solution x.

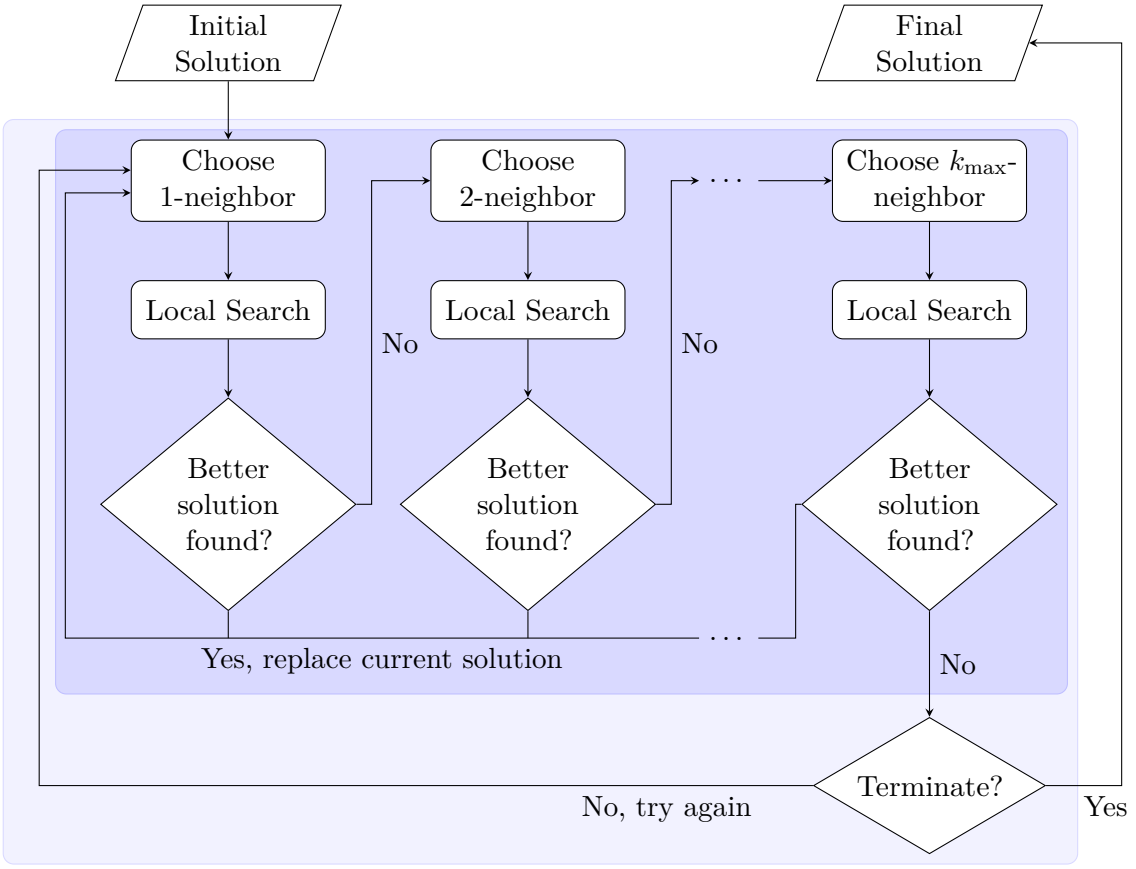


Figure 2.1: The basic process of Variable Neighborhood Search

The shake operation in Line 4 of Algorithm 2.1 can again consist in the random draw from $\mathcal{N}_k(x)$ or can be guided by heuristic information.

After the shaking phase, a *local search* is conducted, with the initial neighbor x' as its starting point (see Line 5). The used policy can vary, with the most common ones being the aforementioned Best Improvement and First Improvement policy. A local search algorithm, as noted above, uses a single neighborhood relation for searching for local improvements to a current solution. It is important to note that this neighborhood relation used in the local search does not have to be among the ones used by the VNS for obtaining k-neighbors in the shaking phase.

The local search leads to a new solution x''. This solution must be evaluated in Line 6 by the objective function and compared to the current solution x. If x'' is an improvement of x', k is reset to 1 and x replaced with the improved solution. Otherwise, k is increased and x remains the current solution. In some versions of VNS, a worse solution can be accepted with some probability. The first version is illustrated by Algorithm 2.2, which is also referred to as "Move or not" in some literature.

Finally, different stopping criteria are possible, e.g. maximum computing time, maximum amount of iterations or a minimum quality of the current solution.

2.2.1 Variants of VNS

As with most metaheuristics, many variants of Variable Neighborhood Search have been proposed. The following section is a brief overview of other important variants, with

Algorithm 2.1: VARIABLE NEIGHBORHOOD SEARCH

Input: Initial solution x, number of neighborhoods $k_{\max}$
Output: Best found solution x

repeat
  $k \leftarrow 1$
  repeat
    `// Obtain initial solution` x' `from` $\mathcal{N}_k(x)$
    $x' \leftarrow \text{SHAKE}(x, k)$
    `// Descend to local minimum` x''
    $x'' \leftarrow \text{LOCALSEARCH}(x')$
    `// Increase` k `or replace` x `with` x'' `and reset` k `to 1`
    $\text{NEIGHBORHOODCHANGE}(x, x'', k)$
  until $k = k_{\max}$
until *Stopping criteria*

Algorithm 2.2: NEIGHBORHOODCHANGE

Input: Current solution x, new solution x'', current neighborhood number k

if $f(x'') < f(x)$ **then**
  `// Move to improved solution`
  $x \leftarrow x''$
  $k \leftarrow 1$
else
  `// Move to next neighborhood`
  $k \leftarrow k + 1$

the naming convention and description again following Hansen, Mladenović and Pérez [MHP08].

The variant *Reduced VNS* skips the local search phase of the basic VNS, thus only trying to obtain improvements in the shaking phase.

Instead of choosing one k-neighbor for each k and using that neighbor for a local search, the method *(Basic) Variable Neighborhood Descent* (VND) searches for the best solution in a k-neighborhood and skips the additional local search phase completely.

Lastly, the variant *General VNS* embeds VND into VNS by replacing the local search step in the VNS algorithm with a VND. As noted with local search before, the outer VNS and the inner VND do not have to use the same neighborhood structures.

2.3 Components of a solar farm

The following section provides an overview of the basic components of a photovoltaic system and their usage. We focus on the components considered in the formalization of the Solar Farm Cable Layout Problem in Chapter 4. For consistency, the naming conventions and description of components follow the work of Stampa [Sta22]. For further details, we also refer to the introductory works by Neill, Stapleton and Martell [NSM17] and Jieb and Hossain [JH22].

Photovoltaic cells (*PV cells*) are the smallest power generating unit. Multiple PV cells connected in series and/or parallel form a *PV module*. Multiple modules in a series connection form a *PV string*. PV strings be connected in parallel are called *PV arrays*.

Strings produce direct current, which must be transformed by *inverters* into alternating current before it can be fed into the power grid. There are two different types of inverters. Central inverter have a high power capacity and strings are connected via combiner boxes before a connection with them. String inverters on the other hand have a lower power capacity and are directly connected with strings. Inverters typically contain a maximum power point tracker (*MPP tracker*) for maximizing the generated power by regulating the voltage.

Multiple strings can be connected in parallel by *combiner boxes*. Direct current combiner boxes (*DC combiners*) connect strings before the connection to the inverters, alternating current combiner boxes (*AC combiners*) connect them afterwards. *Recombiner boxes* are combiner boxes that connect the output of multiple other combiner boxes in parallel and can also be located before and/or after the inverters. Additionally, strings can be directly connected in parallel by *Y-connectors* before any connection to combiners or inverters.

Before the generated power can be fed into the power grid, the voltage must be stepped up by a *transformer*.

The power generated by the strings is conducted by *PV cables* connecting the different components. As with combiner boxes, cables can be differentiated into *AC cables* and *DC cables*, depending on which type of current they conduct. During the conduction of power, some of it is lost. The amount of power lost depends on the length, material and cross-section of the cable as well as the conducted current. The power losses and cabling costs are generally higher on the AC side than on the DC side [JH22, pp. 151 f.].

3. Related Work

The following sections serve as a brief overview of related work. First, in Section 3.1, we describe existing methods for solving the problem of optimizing a solar farm's cable layout. Section 3.2 discusses similar optimization problems and the methods proposed for their solution, with a focus on metaheuristics and VNS.

3.1 Cabling problem in solar farms

The model and problem formulation for the Solar Farm Cable Layout Problem used in this thesis and illustrated in detail in Chapter 4 is adapted from the master's thesis of Stampa [Sta22] and the subsequent paper [GSW22]. Stampa [Sta22] proposes an exact solution method in the form of an MILP and a heuristic method. The heuristic method consists of finding paths for every string to a transformer while adhering to the capacity constraints of the components. The proposed optimization methods only consider solar farm instances with fully connected layers, which means that every component can be connected to any component in the following layer.

As stated in the introductory Chapter 1, the author noted a trade-off between solution quality and runtime when opting for one or the other method. The MILP as an exact method exceeded the heuristic in terms of solution quality, however, it needed more runtime for computing these solutions. For larger problem instances, the MILP could not even compute a feasible solution in a time that the author deemed reasonable. While the heuristic also struggled with finding feasible solutions for increasingly large instances, it was still capable to do so within given runtime limits for some instances the MILP failed at.

A different formulation of the problem pertaining to the minimization of cabling costs in solar farms is proposed by Luo et al. [LQC+21]. The authors combine the problem of designing a cable layout with the problem of placing combiner boxes and formulate their problem as a generalized Capacitated Minimum Spanning Tree Problem (CMST). Given a graph $G = (V, E)$ with weights w_e for every edge $e \in E$ and a capacity c, the solution for CMST is a tree $T = (V, E')$ in G that includes all vertices of G, such that all subtrees incident to the root of T have no more than c vertices and $\sum_{e \in E'} w(e)$ is minimal. Given a set of strings and an inverter placed on a grid, the cabling problem defined by Luo et al. consists in finding a minimum spanning tree representing a cable layout such that the inverter is at its root and the subtrees incident to the root consist of strings. Furthermore, the combiner boxes' are installed at the root of each subtree and have capacities that must not be exceeded. For the problem's solution the authors propose a Branch-and-Price-and-Cut algorithm.

3.2 Similar Problems

The Solar Farm Cable Layout Problem (SoFaCLaP) is related to several other NP-hard optimization problems. The following is a brief overview of similar problems and their solution with a focus on the use of metaheuristics and VNS.

3.2.1 Wind Farm Cabling Problem

The most obviously related problem is the Wind Farm Cabling Problem (WCP), as it is also an optimization problem concerning the design of energy networks, specifically of their cable layouts. A formalized version of WCP can be given as by Problem 3.1..

Problem 3.1. (Wind Farm Cabling Problem)

Input: Directed Graph $G = (V, E)$, substations $S \subseteq V$ with capacities $\text{cap}_{\text{sub}} \colon S \to \mathbb{N}$, turbines $T = V \setminus S$ with power production P_t, possible connections $E \subseteq V^2$ of lengths $\text{len} \in \mathbb{R}_{>0}$ and with $(u, v) \in E$ implying $(v, u) \notin E$ for all $u, v \in V$, cable types L with costs and capacities $\text{cost}_{\text{cab}} \in \mathbb{R}^+, \text{cap}_{\text{cab}} \in \mathbb{N}$

Output: Flow $f \colon E \to \mathbb{R}$ such that there is no outflow from substations and such that the net flow (outflow minus inflow) in each turbine t is $-P_t$ and in each substation s at most $\text{cap}_{\text{sub}}(s)$

Objective: minimizes $\sum_{e \in E} \text{len}(e) \cdot c(|f(e)|)$ with $c(|f(e)|)$ the cheapest cable in L with capacity at least $|f(e)|$

Different formulations of this problem exist, the one given above by Problem 3.1. is mainly adapted from Gritzbach et al. [GUW+19]. Their definition additionally contains the constraint that substations cannot be connected to other substations, and they assume that all turbines $t \in T$ have a standardized power production of $P_t = 1$. In this form, a wind farm can be modeled as a flow network, with the outflow of each turbine corresponding to its producing power. Solutions for a given WCP instance then consist of a cable layout that connects every turbine to some substation, while adhering to the cable and substation capacities. In comparison, solutions for a given SoFaCLaP instance likewise must connect every string to some transformer. However, while any SoFaCLaP solution, including an optimal solution, is always an in-forest or in-tree, a solution for WCP does not have to be either as it also allows cycles, as pointed out by Gritzbach, Wagner and Wolf [GWW20]. For both problems, instances contain a set of cable types with different costs and capacities, and both problems have the same objective, namely minimizing the costs of the cable layout. The main difference are the vertex layout and capacities. Solar farms are organized in layers of different components with their own capacities, while wind farms only have two types of components, which can be connected among each other in no particular order and with no respect to component capacities.

Numerous metaheuristic and likewise heuristic methods have been applied to the Wind Farm Cabling Problem. One heuristic method for WCP is an algorithm based on Negative Cycle Canceling as presented in [GUW+19], which was able to compute more efficiently solutions of similar quality than the compared metaheuristic Simulated Annealing proposed by [LRWW17]. A different heuristic method proposed in the bachelor thesis of Jenne [Jen20] is based on the Successive Shortest Path algorithm for the minimum cost flow problem, which is similar to WCP and hence also to SoFaCLaP. While failing to reach the solution quality of the mentioned Negative Cycle Canceling heuristic overall, it was still able to compute qualitatively good solutions in less time.

A comparative study by Cazarro, Fischetti and Fischetti [CFF20] provides an overview of the application and success of different metaheuristics for WCP, among them being

Variable Neighborhood Search, but also Simulated Annealing, a genetic algorithm, Tabu Search and Ant Colony Optimization. The authors' definition of WCP differs in some aspects from Problem 3.1.. For one, solutions are not defined by flows but by the placement of cables so that there is a path from every turbine to a substation with the *resulting* flow adhering to the cables' and substations' capacities. While, as noted above, the general model allows for solutions containing cycles, the authors prohibit reconnections that result in such cycles. Additionally, the authors only consider instances with one substation with its capacity referring to the number of ingoing cables instead of power flow. They also penalize solutions that include crossings of cables. The authors define k-neighborhood structures $\mathcal{N}_k$ as introduced in Chapter 2 for their VNS by the redirection of cables. Given a specific solution, a solution in its neighborhood $\mathcal{N}_k$ can be obtained by randomly reconnecting k turbines to different successors. For the local search method used in VNS, the authors note that a Best Improvement policy outperforms a First Improvement policy. In their experiments, the best setting for the amount of neighborhoods is concluded to be $k_{\max} = 4$. Choosing a good limit for the amount of neighborhoods is crucial for obtaining a balance between exploring the solution space on the one hand and retaining efficiency on the other. The authors' proposed Variable Neighborhood Search method was able to outperform the other in its solution quality. For each run, the authors considered a time limit of ten minutes, with VNS being the only method that continued to find improvements, even if only slight ones, after passing the ten-minute mark. The authors additionally compare the metaheuristics' performances when starting with a random initial solution with the usage of a construction heuristic, within the same time limit of ten minutes. While all compared metaheuristics benefit greatly from the usage of such a construction heuristic as a starting point, VNS is also able to reach good solutions on its own.

3.2.2 Steiner problem in graphs

A different problem similar to SoFaCLaP is the Steiner Tree Problem in graphs (STP), which can be formulated as in Problem 3.2.. The directed variant of the problem is called the Steiner Arborescence Problem in graphs (SAP) (see Problem 3.3.).

Problem 3.2. (Steiner tree problem in graphs)
Input: Undirected Graph $G = (V, E)$, Terminals $S \subseteq V$, edge weights $w\colon E \to \mathbb{R}^+$
Output: Tree $T = (V_T, E_T)$ in G with $S \subseteq V_T$
Objective: minimizes $\sum_{e \in E_T} w(e)$

Problem 3.3. (Steiner arborescence problem in graphs)
Input: Directed Graph $G = (V, E)$, Terminals $S \subseteq V$ with designated root $r \in S$, edge weights $w\colon E \to \mathbb{R}^+$
Output: Arborescence $T = (V_T, E_T)$ in G rooted in r with $S \subseteq V_T$
Objective: minimizes $\sum_{e \in E_T} w(e)$

Every cable layout for a solar farm can be represented as an in-forest. The problem of finding a path from every string to a transformer is similar to finding a Steiner tree in the underlying undirected solar farm graph with an additional root vertex connected to all the transformers, with the strings and this root vertex being the terminals. However, a valid Steiner tree in such a graph could contain components with two outgoing cables to the next layer, which does not result in a valid anti-arborescence or in-forest in the corresponding directed graph. On this point, SoFaCLaP is more similar to SAP than to STP. Finding a valid in-forest representing a cable layout is directly related to finding an anti-arborescence in the solar farm graph with reversed edge orientations and with

the strings as terminals and an additional vertex connected to all transformers as the designated root. Of course, besides the inclusion of capacities and the orientation of the edges, the main difference that still sets SoFaCLaP apart from SAP and also STP, and in fact many other optimization problems in graphs, is the usage of a set of *possible* edge weights instead of a weight function.

The Steiner Tree Problem in graphs is a thoroughly researched problem and there have been different proposals of metaheuristic methods for its solution. A recent review of STP, SAP and other variants and proposed solution methods, including many metaheuristics can be found in [Lju21]. Variable Neighborhood Search and variants such as Variable Neighborhood Descent have been applied to it mainly in the combination with other metaheuristics, forming so-called *hybrid metaheuristics*. Examples are the hybridization with GRASP (greedy randomized adaptive search procedure) in [MRRP00] and [RUW02]. Two types of neighborhood structures that are commonly used are *node-based* and *key-path-based* neighborhoods. An extended evaluation of these neighborhood types and their inclusion in local search methods has been provided by [UW12]. Node-based neighborhoods are obtained by removing or adding nodes to the current Steiner tree. Key-path-based neighborhoods are obtained by the replacement of *key-paths* in the current Steiner tree or elimination of keynodes. A keynode is a vertex with degree at least three in a given Steiner tree. The keynodes and terminals of a Steiner tree are called its crucial nodes. A keypath is a subpath in a Steiner tree between two crucial nodes, which only has non-crucial nodes as intermediate vertices. While not explicitely labeling it as VNS, Rayward-Smith and Wade propose a local search method for STP in [WRS00] that switches between the two types of neighborhoods to escape local minima, which is the main idea of VNS.

3.2.3 Capacitated Multi-level Facility Location Problem

A third problem related to the Solar Farm Cable Layout Problem is the Capacitated Multi-level Facility Location Problem (CMLFL). Different formulations of this problem exist, with the one in Problem 3.4. being adapted from [Mar10], with the addition of facility capacities.

Problem 3.4. (Capacitated Multi-level Facility Location Problem)

Input:	Set of customers D, set of possible facilities $F = F_1 \sqcup \cdots \sqcup F_k$ at each level from 1 to k with set-up costs s_i and capacities c_i for every $i \in F$ and transportation costs $t_{i,j}$ for $i, j \in F \cup D$
Output:	For every customer $d \in D$ a sequence of facilities $p \in (F_1 \times \cdots \times F_k)$, such that no facility supplies more customers than it has the capacity for
Objective:	minimizes sum of set-up costs of all used facilities and transportation costs from each customer to the first facility and between each next facility in the sequence

Structurally, SoFaCLaP and this problem are very similar. Just as every customer must be assigned to a sequence of facilities, every string must be assigned to a path of components, with both facilities and components being partitioned into layers and having certain capacities. While every customer and string must be considered by the solution, not every facility and not every component that is not a string must be used by it. However, since CMLFL as formulated in Problem 3.4. allows transports between any two facilities of subsequent layers, this analogy only works when considering SoFaCLaP instances with fully connected layers. Another difference, as with the Steiner Arborescence Problem in graphs, is that the cabling costs used for SoFaCLaP are given as a step function, but the transportation costs in CMLFL are not. Additionally, the Solar Farm Cable Layout

Problem does not consider costs of the used components which would equal the set-up costs of used facilities.

An overview of different versions of the Multi-Level Facility Location Problem and proposed solutions has been provided by Ortiz-Astorquiza, Contreras and Laporte [OACL18]. In this overview, the authors note that the research for the Multi-level Facility Location Problem and its variants seems to be focused on the uncapacitated cases. In respect to (meta-)heuristic solution methods, a genetic algorithm for the uncapacitated version has been proposed by Marić [Mar10] and later improved by the hybridization with a local search method [MSDS14]. Additionally, a genetic algorithm has been proposed for the capacitated problem in the case of two facility levels [FRA+14]. For the uncapacitated facility location problem with two levels, Gendron, Khuong and Semet [GKS15] proposed a VNS variant called Multilevel VNS (MLVNS), which divides the neighborhood structures $\mathcal{N}_k$ into layers and invokes a VNS for each layer, which uses a recursive call to MLVNS as its local search method. The used neighborhood structures are based on exchanging the facilities assigned to a customer and opening and closing facilities. The authors compared their metaheuristic to an integer program and a heuristic algorithm while using two different objective functions. While their VNS performed in comparison well overall, it clearly outperformed the integer program and to a lesser extent the heuristic when using large instances and a complex objective function. Additionally, each layer of the MLVNS contributed to the solution quality.

3.3 Contribution and Outline

In this chapter we have seen that previous optimization methods for the Solar Farm Cable Layout Problem have been focused on exact and heuristic methods. Since the Solar Farm Cable Layout Problem is an NP-hard problem, metaheuristics such as VNS constitute a different type of optimization methods worth trying out. To our knowledge, no version of VNS has been applied to this particular problem. However, variations of VNS have been formulated for different similar problems. Especially the reported success of VNS for the related Wind Farm Cabling Problem makes this metaheuristic a promising candidate for the problem at hand.

In this thesis, we present an application of VNS to the Solar Farm Cable Layout Problem. To evaluate its performance, we compare it to the heuristic and the MILP formulation by [Sta22] mentioned in Section 3.1. On the same problem instances, VNS generally reaches better solutions in less time than the heuristic. It reaches more often feasible solutions than both the heuristic and the MILP, specifically on large instances, on which both the heuristic and more so the MILP struggle to do so. While overall not outperforming the MILP's solutions when those are available, our VNS reaches the best solutions so far on some large instances the MILP could not solve to feasibility.

In the next chapter, we formalize the Solar Farm Cable Layout Problem, for which we then present our VNS in Chapter 5. For each of the components of VNS, we present different possible choices. In Chapter 6, we conduct and evaluate experiments to determine the best VNS variant, which we then compare to the MILP and the heuristic mentioned in Section 3.1.

4. The Solar Farm Cable Layout Problem

Given a solar farm, in which the positions of strings and potential other components are already preassigned, we now want to find the best possible cable layout connecting every string to a transformer. The solution must satisfy certain constraints, e.g. adherence to capacities of the components. Our goal is to find the solution with the least amount of costs. The formulation of the problem in Section 4.2 and the chosen model for solar farms and their components follow the definition of the Solar Farm Cable Layout Problem presented by Stampa [Sta22] and by Gritzbach, Stampa and Wolf in the subsequent work [GSW22].

The first section of this chapter describes the model we use to represent solar farms. Along with the description we briefly mention some aspects of solar farms that are not included in the model. The second section then defines the Solar Farm Cable Layout Problem on the basis of that model.

4.1 Modelling the layout of a solar farm

We represent a solar farm as a layered directed graph $G = (V, E)$. The vertices V in G represent the components of the solar farm, the edges E represent all *possible* connections between components. The set of vertices V is partitioned from bottom up into subsequent layers $V_1, \dots, V_n$. The bottom layer V_1, i.e. the layer without incoming edges, always consists of strings, the top layer, i.e. the layer without outgoing edges, of transformers.

4.1.1 Modelling the components

Along with the partition into layers, the set of vertices V can be partitioned into the disjoint subsets V_{con}, V_I, V_C, V_R, V_Y and V_T, with the subsets representing the different types of components as laid out in the following. Each layer of vertices only contains one type of component, but multiple layers can be of the same component type. The naming of the subsets is consistent with [Sta22].

A solar farm contains a set S of strings. Every string has connection points for the connection with a cable. For every string $s \in S$, V contains the subset V_{con}^s of connection points of s. The subset of vertices representing all connection points of all strings is $V_{\text{con}} = \bigcup_{s \in S} V_{\text{con}}^s$. We assume that all strings have the same length and tilt angle.

The set of inverters is represented by the subset $V_I \subsetneq V$ of vertices. The current conducted to an inverter must either be zero or in range of the inverter's minimum or maximum

allowed current, also called the lower bound and the capacity. The lower bound of the inverter is denoted by the value $i_{\min} \in \mathbb{N}_0$. We only consider solar farms that use the same lower bound for all their inverters. Again along with [Sta22], we do not account for MPP trackers in our model. Furthermore, we assume that string inverters are located on AC side and central inverters on DC side.

The set of combiner boxes, recombiner boxes and and Y-connectors are represented by the subsets $V_C, V_R, V_Y \subsetneq V$ of vertices. We assume that there are no connections between combiner boxes of the same layer or layers.

The set of transformers is represented by the subset $V_T \subsetneq V$ of vertices. We only consider solar farms with exactly one layer of transformers.

We assume that every component (except for transformers) only has one output. For a string this means that only one of its connection points has an outgoing cable. We do not consider limitations on the amount of ingoing cables of inverters or combiner boxes.

All components $v \in V \setminus V_{\text{con}}$ have a capacity $\text{cap}(v) \in \mathbb{N}$. To simplify things, we assume vertices of the same component type have equal capacities. It is however possible to use different capacities for different vertices of the same type. Additionally, we will sometimes refer to the capacity cap_i of a layer V_i, which is equal to the capacity of the components the layer consists of.

4.1.2 Modeling the possible cables

Edges in G represent possible cables between components. Edges only exist between consecutive layers and only directed from lower to upper layers. The instances we consider for our VNS algorithm presented in the next chapter only consist of solar farms with fully connected layers, i.e. solar farm graphs with $E = \bigcup_{i=1}^{n-1} V_i \times V_{i+1}$.

The function $\text{len}\colon E \to \mathbb{R}$ assigns every possible cable $e \in E$ its length. The function $\text{cap}_{\text{cables}}\colon C \to \mathbb{R}$ assigns every cable type its capacity, the function $\text{cost}\colon C \to \mathbb{R}$ its cost per length unit. The special cable type $c_0 \in C$ has zero capacity and cost and shall be used in the solution to signify edges without cables. As edges can be partitioned into AC and DC edges, it should be noted that it is possible to allow different cable types for these two edge types. For the instances we consider in Chapter 6 however, we allow the same cable types for both AC and DC edges.

One important factor for the cost efficiency of solar farms we do not account for are power losses due to the chosen cable layout.

4.2 The Solar Farm Cable Layout Problem

Given a graph $G = (V, E)$ representing a solar farm as set out above, we want to find a solution consisting of a flow function $f\colon E \to \mathbb{R}$ and a function $h\colon E \to C$. The function f denotes the current flow on every edge, with $f(e) = 0$ for an edge $e \in E$ if and only if the chosen cable type for e is the "non-cable" c_0. The flow is measured in units of strings, with every string having a total outflow of one. The function h appoints either a cable or c_0 to every edge in E.

We call a solution *feasible* if it satisfies the following constraints:

(C1) Every string must be connected to a cable at exactly one of its connection points:

$$\forall s \in S : \exists u \in V^s_{\text{con}}, e = (u, v) \in E \text{ with}$$
$$f(e) = 1 \text{ and } \forall u' \in V^s_{\text{con}}, e' = (u', v') \in E : u' \neq u \Leftrightarrow f(e') = 0\,.$$

(C2) Only strings and transformers can increase or decrease the conducted current, all other components must conserve the flow:

$$\forall v \in V \setminus (V_{\text{con}} \cup V_T) : \sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w)$$

(C3) The capacity at each cable must not be exceeded:

$$\forall e \in E : f(e) \leq \text{cap}_{\text{cables}}(h(e))$$

(C4) The capacities of the components must not be exceeded:

$$\forall v \in V : \sum_{(u,v)\in E} f(u,v) \leq \text{cap}(v)$$

(C5) The current conducted to each inverter must either be zero or adhere to the inverter's lower bound:

$$\forall v \in V_I : \sum_{(u,v)\in E} f(u,v) \geq i_{\min} \vee \sum_{(u,v)\in E} f(u,v) = 0$$

(C6) Every inverter, (re)combiner and Y-connector is connected to at most one outgoing cable:

$$\forall u \in V \setminus (V_{\text{con}} \cup V_T) : |\{(u,v) \in E \mid f(e) > 0\}| \leq 1$$

The solution can be represented as a directed forest with the root of each anti-arborescence of the forest representing a transformer and leaves representing strings. Our goal is to find the (feasible) solution fulfilling

$$\text{cost}_{\text{cables}} := \sum_{e\in E} \text{cost}(h(e)) \cdot \text{len}(e) \stackrel{!}{=} \min,$$

i.e. the cable layout resulting in the minimal cost.

4.2.1 Complexity of the Problem

As shown by [Sta22], the problem of finding any solution satisfying the constraints set out above for a given problem instance is strongly NP-complete. The authors in [GSW22] suspect that the problem of finding a feasible solution is solvable in polynomial time for instances that possess the following attributes:

- layers are fully-connected and consist of vertices with unique positions of points in $\mathbb{Q}^2$,
- edge lengths equal the Euclidean distance of that points,
- the only one available cable type (besides c_0) has unlimited capacity,
- the inverters' lower bound is $i_{\min} = 0$.

However, it is shown in [Sta22] that the optimization problem on that subset of instances remains strongly NP-hard.

4.3 Existing solution methods

Besides the formulation of the Solar Farm Cable Layout Problem, Stampa also proposed two solution methods along with it in [Sta22], as mentioned in Chapter 3. One of the two methods is an exact one in the form of an MILP formulation. The MILP formulation is also presented in a simplified form in [GSW22]. The other proposed method is a heuristic. The basic outline of the heuristic is given in the following.

First, the heuristic tries to construct a feasible solution for the given problem instance. The construction is done by forming paths from each string to a transformer, one string at a time. When connecting a new path, the heuristic prioritizes choosing components with the most remaining capacity. After the construction, it tries to improve the initial solution while maintaining feasibility by repeatedly searching for possible cost reductions: for every vertex with an outgoing cable, the heuristic calculates the cost difference that would result from switching that cable to a different successor. The heuristic then chooses the change that results in the largest cost reduction while maintaining feasibility and repeats the process until no further improvement can be made.

This heuristic, however, is not optimal. This can be seen in Figure 4.1, which depicts a simple, theoretical solution that is suboptimal but cannot be further improved by the heuristic.

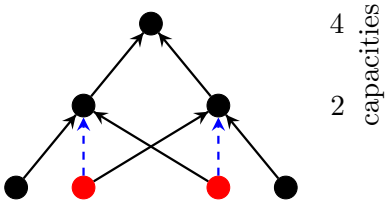


Figure 4.1: A solar farm graph with one transformer, two inverters and four strings from top to bottom. The black edges are the chosen connections in the current solution. Switching the current outgoing cables of the red strings to the blue dashed edges results in the optimal solution for this solar farm. However, when starting from the current solution, the heuristic cannot find the optimum, as it has to change the cables one at a time, which temporarily results in an infeasible solution.

The example solution in Figure 4.1 can be seen as a local minimum for the heuristic on that instance, as every move the heuristic could make results in worse solutions. As explained in Chapter 2, the metaheuristic VNS is designed to be able to escape such local minima. In the following chapter, we therefore present an application of VNS to the Solar Farm Cable Layout Problem. In Chapter 6, we show that practical instances exist on which our VNS is able to further improve solutions found by the heuristic which the latter can not.

5. Algorithm

In this chapter we outline our tailoring of Variable Neighborhood Search to the Solar Farm Cable Layout Problem. As described in Chapter 2, VNS is a metaheuristic that builds upon the simple local search scheme by introducing multiple neighborhood structures. VNS utilizes these different structures both for descending to local minima of the solution space, and for breaking out of those.

Algorithm 5.1 illustrates our VNS scheme. Starting with an initial solution and an initial neighborhood structure, which are all based on a recabling move detailed in Section 5.1, VNS first chooses a neighbor from that neighborhood (Line 4) and then searches for improvements through local search (Line 5). If these two steps result in a better solution, the algorithm updates the incumbent solution and restarts with the first neighborhood structure (Lines 7 and 8). Otherwise, the algorithm moves to the next neighborhood (Line 10), repeatedly iterating over all $k_{\max}$ of them, until it either succeeds or does not find improvements in any of them. In the latter case, VNS either restarts with $k = 1$ or stops if the stopping criteria are met. The stopping criterium typically consists of exceeding a given time limit. For the experiments detailed in the next chapter, we will make use of such a time limit. However, it can be enforced at any step of our VNS.

In the following sections, we first expand on the design of the separate components of our VNS algorithm in Section 5.1. Then, we introduce our representation and evaluation of solutions in Sections 5.2 and 5.3. Finally, in Section 5.4, we describe how we construct initial solutions for our VNS.

5.1 Components of VNS

The key component of Variable Neighborhood Search are the neighborhood structures $\mathcal{N}_k$. Given a solution x, we define a recabling move on x as the reassignment of a random component to a different successor. The k-neighborhood of x is then the set of all solutions x' that can be obtained by k consecutive recabling moves on x. This results in nested neighborhood structures with $\mathcal{N}_k \subseteq \mathcal{N}_{k+1}$ for all $k \in \mathbb{N}$. In the shaking phase of VNS (Line 4 of Algorithm 5.1) we simply draw a random k-neighbor x' of the incumbent solution x.

Given a solar farm graph $G = (V, E)$ with $V = V_1 \cup \cdots \cup V_n$, we denote the different layer components as $V_i = \{v_1^i, \ldots, v_k^i\}$, $k = |V_i|$ for every $i = 1, \ldots, n$. Following [Sta22], we only consider instances with fully connected layers.

Algorithm 5.1: VARIABLE NEIGHBORHOOD SEARCH

Input: Initial solution x, number of neighborhoods $k_{\max}$
Output: Best found solution x

```
1  repeat
2      k ← 1
3      repeat
           // Obtain neighbor x' through k random recabling moves on x
4          x' ← RECABLE(x, k)
           // Improve initial neighbor through repeated recabling
5          x'' ← LOCALSEARCH(x')
           // Restart with improved solution or move to next
              neighborhood
6          if f(x'') < f(x) then
7              x ← x''
8              k ← 1
9          else
10             k ← k + 1
11     until k = k_max
12 until Stopping criteria
```

For the local search in Line 5, we use two different method, one following a *Best Improvement* policy and the other a *First Improvement* policy. For the Best Improvement method as detailed in Algorithm 5.2, we repeatedly determine the *best* recabling move for the current solution, until no further improvement can be made. For every component $v \in V \setminus V_n$ we only employ one recabling move, so in every iteration we only consider $|V \setminus V_n|$ out of $\sum_{i=1}^{n-1} |V_i| \cdot (|V_{i+1}| - 1)$ possibilities for solar farms with fully connected layers. For the First Improvement method as detailed in Algorithm 5.3, we conduct a recabling move on every component in random order (Line 5), stopping as soon as an improvement is found (Lines 6 to 7), repeating this process until no further improvement can be made.

If the solution x'' obtained by the local search is better than the incumbent x, we replace x with x'' and reset the neighborhood counter k (Lines 7 and 8 of Algorithm 5.1). Otherwise, we increase k by one until it reaches the maximum value $k_{\max}$ (Line 10).

5.2 Representation

In any solution, every component can at most have one outgoing cable. Thus, for every component we only have to store up to one successor in the next layer, except for transformers, which have no successors. Let $G = (V, E)$ again be a solar farm graph with successive component layers $V = V_1 \cup \cdots \cup V_n$. We define a function $\text{succ}_i \colon V_i \to V_{i+1}$ for the layers $i = 1, .., n-1$. A component v_j^i of layer V_i is then adjacent to component v_k^{i+1} of layer V_{i+1} in a solution, if $\text{succ}(v_j^i) = v_k^{i+1}$. Note that a solution assigns a successor to *every* component besides the transformers, even if it does not lie on a path from a string to a transformer, which must be considered when evaluating the cabling costs of a solution. With this representation, every solution can be represented as an in-forest and every string is always connected to a transformer.

If a string of a solar farm graph has multiple connection points with edges to the same vertex, those edges can be unified by removing all except the shortest one [Sta22]. For a solar farm with fully connected layers, we then do not have to represent connection points

Algorithm 5.2: BEST IMPROVEMENT

Input: Current solution x', solar farm layers $V = V_1 \cup \cdots \cup V_n$
Output: New solution x''

$x_{\text{last}} := x'$
repeat
  $x_{\text{best}} \leftarrow x_{\text{last}}$
  for $i = 1, .., n-1$ **do**
    forall $v \in V_i$ **do**
      $x''.\text{RECABLE}(v)$
      if $f(x'') < f(x_{\text{best}})$ **then**
        $x_{\text{best}} \leftarrow x''$
  $x'' \leftarrow x_{\text{best}}$
until $f(x'') \geq f(x_{\text{last}})$
$x'' \leftarrow x_{\text{last}}$
return x''

Algorithm 5.3: FIRST IMPROVEMENT

Input: Current solution x', solar farm layers $V = V_1 \cup \cdots \cup V_n$
Output: New solution x''

$x_{\text{last}} := x'$
repeat
  $x'' \leftarrow x_{\text{last}}$
  forall $v \in V \setminus V_n$ **do**
    $x''.\text{RECABLE}(v)$
    if $f(x'') < f(x_{\text{last}})$ **then**
      break
  $x'' \leftarrow x_{\text{last}}$
until $f(x'') \geq f(x_{\text{last}})$
return x''

explicitly. We therefore consider V_1 to consist of strings instead of connection points and set $\text{len}(s, v)$ to $\min\{\text{len}(c, v) \mid c\ c \in V^s_{\text{con}}\text{connection point of}\, s\}$ for every string $s \in V_1$ and vertex $v \in V_2$.

5.3 Evaluation of solutions

We allow for the inclusion of infeasible solutions in our VNS that violate the capacity constraints of components (C4), of cables (C3) and/or the lower bound constraint of inverters (C5). Therefore, we introduce a real-valued penalty function p, with

$$p(x) = P_I \cdot (c_{\text{cab}}(x) + c_{\text{comp}}(x) + c_{\text{inv}}(x))$$

for every solution x, where

- $c_{\text{cab}}(x)$ is the amount of overloaded cables in x,
- $c_{\text{comp}}(x)$ is the summed amount of flow over the capacity of overloaded components in x,

- $c_{\text{inv}}(x)$ is the summed margin of current flow and lower bound of all underloaded inverters in x,
- $P_I = \max_{c \in C} \text{cost}(c) \cdot \max_{e \in E} \text{len}(e) \cdot |V|^2$ is a constant penalty term for the problem instance I that x is a solution for.

The objective value of a solution x is then given by $f(x) = \text{cost}(x) + p(x)$, with $f(x) = \text{cost}(x)$ if and only if x feasible. Note that the penalty term P_I is chosen in such a way that every feasible solution is better (in terms of the objective function f) than any infeasible solution. Therefore, our VNS always favors feasible over infeasible solutions.

We do not assign cables to edges of the solar farm graph explicitly. Instead, when determining the objective value $f(x)$ of a solution x, we first compute the *inflow* $\text{in}_f(v)$ at every vertex $v \in V \setminus V_1$, which in our case is the amount of strings connected to a component v. The inflow at every vertex can easily be computed recursively for every layer from the bottom up, with the inflow of vertices $v \in V_2$ being the amount of strings directly connected to them. The cabling cost $\text{cost}(x)$ and penalty $p(x)$ of a solution x is then determined as follows: For every string, we choose the cheapest cable type with capacity > 0. For every other vertex v with $\text{in}_f(v) > 0$ save for transformers, we choose the cheapest cable type with capacity $\geq \text{in}_f(v)$ as the outgoing cable. If such a cable type does not exist, we account for this case through the term $c_{\text{cab}}(x)$ of the penalty function. For every vertex v in a layer V_i with an outgoing cable, $\text{len}(v, \text{succ}_i(v))$ times the cost of the chosen cable type is added to the total cost. For every overloaded vertex $v \in V \setminus V_1$ the amount of inflow at v above its capacity $\text{in}_f(v) - \text{cap}(v)$ is added to the penalty term $c_{\text{comp}}(x)$. Similarly, for every underloaded inverter w with $0 < \text{in}_f(w) < i_{\min}$, we add the margin $i_{\min} - \text{in}_f(w)$ to the penalty term $c_{\text{inv}}(x)$.

5.4 Initial solution

We use and compare two different ways of obtaining an initial solution: randomization and a greedy construction heuristic.

A random initial solution can be constructed in $\mathcal{O}(|V \setminus V_n|)$ by assigning a random successor to every vertex $v \in V_i$ for $i = 1, \ldots, n-1$.

The second construction algorithm is presented in Algorithm 5.4 in form of a greedy construction heuristic. This heuristic constructs a solution by successively connecting every vertex to the closest possible vertex of the following layer, starting with the strings as the lowest layer. During the construction, the heuristic keeps track of the inflow of each component and always chooses the closest vertex with enough capacity left as a successor, if possible. This way, the heuristic tries to avoid violating capacity constraints of components. To ensure that this holds for instances with layers V_i, V_j with $i < j, \text{cap}_i > \text{cap}_j$, we limit the inflow $\text{in}_f(v)$ of vertices v of a layer V_i to $\text{in}_f(v) \leq \min\{\text{cap}_j \mid j \in \{i, \ldots, n\}\}$. Additionally, we attempt to only construct solutions that do not violate cable capacities if possible, by limiting the inflow of vertices to $\max_{c \in C} \text{cap}_{\text{cables}}(c)$. The heuristic does not, however, ensure feasibility for its constructed solutions. A simple instance, for which the greedy construction heuristic can compute a solution that violates every considered constraint, is given by Figure 5.1.

Additionally, as part of our experiments in Chapter 6, we consider the performance of VNS on the solutions of the heuristic proposed by Stampa [Sta22], which takes all solution constraints into account.

Algorithm 5.4: GREEDY CONSTRUCTION

Input: Solar farm graph $G = (V, E)$ with $V = V_1 \cup \cdots \cup V_n$, edge lengths $\text{len}(\cdot)$, layer capacities cap_i, Cables C with capacities $\text{cap}_{\text{cables}}(\cdot)$
Data: Priority queue Q of pairs $(v, \text{key}(v))$, inflow $\text{in}_f(v)$ at all vertices V
Output: Initial solution x, successors of vertices in x given by $x.\text{SUCC}(\cdot)$

```
// Initialization
forall v ∈ V do
    if v ∈ V_1 then
        in_f(v) ← 1
    else
        in_f(v) ← 0
    if v ∉ V_n then
        x.SUCC(v) ← ⊥
cap_max := max_{c∈C} cap_cables(c)
for i = 1, ..., n − 1 do
    for v ∈ V_i do
        forall w ∈ V_{i+1} do
            Q.INSERT(w, len(v, w))
        c := Q.GETMIN()
        // Choose closest upper vertex with enough capacity
        while Q is not empty and x.SUCC(v) = ⊥ do
            w ← Q.DELETEMIN()
            if in_f(w) + in_f(v) ≤ min({cap_j | j ∈ {i + 1, ..., n}} ∪ {cap_max}) then
                x.SUCC(v) ← w
        // If no vertex with enough capacity left, choose closest one
        if x.SUCC(v) = ⊥ then
            x.SUCC(v) ← c
        in_f(x.SUCC(v)) ← in_f(x.SUCC(v)) + in_f(v)
        Q ← ∅
return x
```

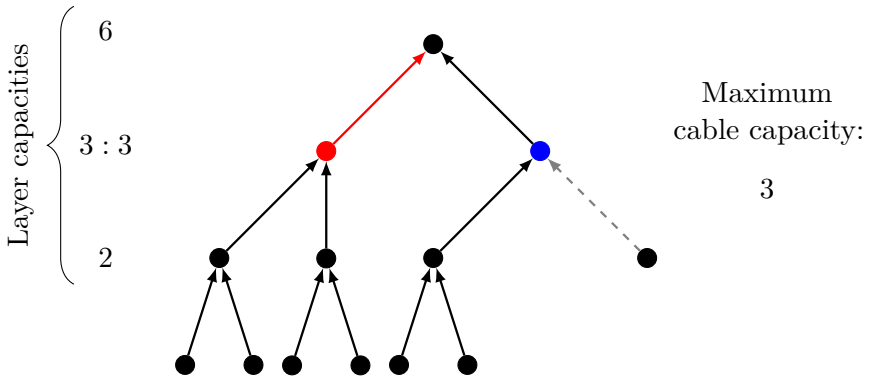


Figure 5.1: The figure shows a possible solution computed by the greedy construction heuristic that violates the capacity constraint for cables and for components, as well as the inverter minimum constraint. On the left, the component capacities of each layer is given, with the third layer being the inverter layer and $3 : 3$ denoting the inverter minimum as the first number and the inverter capacity as the second. Red nodes and edges indicate overloaded components and cables, blue nodes underloaded inverters. A different, feasible solution can easily be found, for example by reconnecting the third from left string to rightmost component of the second layer.

6. Experiments

In this chapter, we test and analyze the performance of our VNS as described in the previous chapter. First, we provide an overview of the problem instances we used for our test sets in Section 6.1. Then, in Section 6.2, we compare different variants of our VNS to each other. After determining the best variant, we compare it to the heuristic method proposed by Stampa [Sta22] in Section 6.3. We also compare VNS to Gurobi on the MILP formulated by the same author as well and use its solutions as a baseline to judge the performance of our VNS variants.

Our code is written in C++14 and compiled with GCC 11.3.0. We used a modified version of mhlib[1] 4.6 to implement and run our VNS and the Open Graph Drawing Framework[2] (Catalpa release), to parse solar farm graphs given in graphML format. We modified the mhlib library to make its VNS template match the VNS algorithm presented in Chapter 2. All experiments, ours and the ones conducted by Stampa, were run on a SuperMicro H8QG6 Server with four 12-core AMD CPUs and 256 GB of memory. The experiments were run in single-threaded mode to ensure comparability.

6.1 Solar farm instances and cable types

To test our VNS, we use the solar farms that were generated by Stampa [Sta22] to evaluate their MILP and heuristic. An extensive explanation of the generation process can be found in the aforementioned work.

The instances are divided into three size categories: small, medium and large. Small instances contain between 120 and 180 strings and only one inverter and transformer. If the small instance is feasible, its inverter therefore has a lower bound at most and a capacity at least as high as the amount of strings. In contrast, medium and large instances can contain multiple inverters and transformers. Medium instances encompass 500 to 700 strings, large instances contain from 1200 up to 1500 strings.

We also use the same cable types that were used in [Sta22], which are also shown in Table 6.1.

[1] https://bitbucket.org/ads-tuwien/mhlib
[2] https://ogdf.uos.de/

capacity	5	22	50	80	180	400
cost	4	34	120	230	750	2300

Table 6.1: The used cable types, sorted by their capacity in ascending order.

6.2 Comparison of variants of VNS

In the first set of experiments, we compare different variants of VNS in terms of initial solution construction, local search policy and $k_{\max}$. We reduce the number of compared combinations by determining each of the aforementioned parameters one at a time in that order.

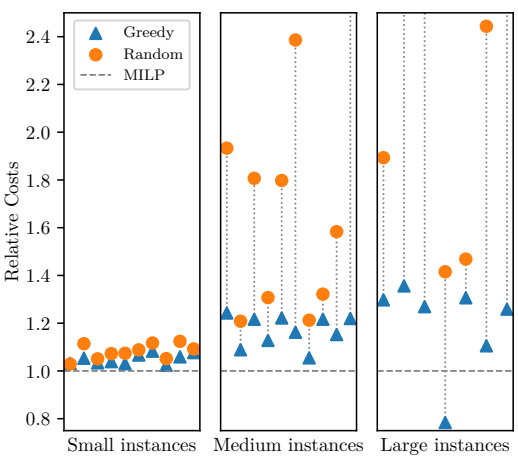
Each experiment in this set is conducted on its own test set of 30 different, randomly chosen instances with ten instances per size category. We exclude problem instances that are proven to be infeasible by the MILP from being chosen for the test sets. For every experiment, the running time of VNS is limited to five minutes on small, to ten minutes on medium, and to 30 minutes on large instances. In comparison, Gurobi on the MILP were given a time limit of 24 hours for each instance. Time limits for VNS are enforced after the completion of the current shaking method or step in the local search phase.

In this section, "average" values always refer to the arithmetic mean. The cost of a method A relative to a method B on an instance always refers to the cost of A's solution on that instance divided by the cost of B's solution. We say that the relative cost difference of A to B equals x percent on an instance, if A's solution costs $|x|$ percent *more* than B's solution on that instance (or less, if x is negative).

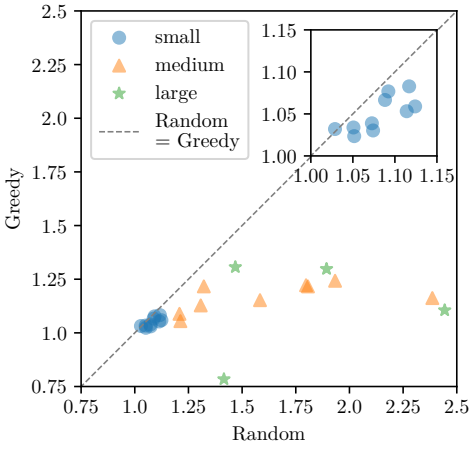
6.2.1 Initial solution

In the first step, we compare VNS with a random initial solution to VNS with an initial solution constructed by the greedy heuristic proposed in Section 5.4. Both variants are run once for each of the 30 instances in the test set. For every size category, the variants are run with a Best Improvement policy on one half of the instances and with a First Improvement policy on the other half, as presented in Algorithm 5.2 and Algorithm 5.3. Which policy is used for which instance is determined randomly but stays the same for both VNS variants. We set $k_{\max}$ to ten for all runs.

For three large instances of the test set, the MILP was not able to find a feasible solution within the given time limit (but did not prove their infeasibility either). For one of those instances, the two VNS variants were not capable of doing so either. On all other 29 instances, the variant using the greedy construction heuristic found a feasible solution, while the variant with the random initial solution was only successful on 24 instances, failing on five large and one medium instance. Additionally, for all instances except one small one, the final solution found by the variant starting with the heuristically constructed solution was less expensive. Considering only the instances both variants reached feasible solutions on, the solutions found when starting with a random solution cost on average 29% more than when starting with the heuristic's solution. The average cost of both variants relative to the MILP increased with increasing instance size, as can be seen in Figures 6.1a and 6.1b. However, even when comparing the two variants, the larger the instances, the more expensive the final solution when starting with a random solution compared to the other variant. While only costing 3% more on average on small instances, the cost difference rose to 38% on medium and 62% on large instances. Therefore, the larger the instances get, the more advantageous it is to use the greedy initialization over the random initialization.



(a) The x-values stand for instances (in no particular order), the y-values for the costs of the two VNS variants relative to the MILP per instance. Missing points within the figure for an instance indicate infeasible solutions.



(b) Costs relative to MILP of VNS solutions with greedy and random initialization. Each point represents one instance. The figure only shows the costs for those instances for which both VNS variants reach feasibility.

Figure 6.1: Costs of solutions found by VNS with greedy and random initialization relative to MILP. For one small instance (leftmost instance in the left subfigure), the solution of the random variant is better than that of the greedy one. On one large instance, VNS with greedy initialization achieves a better solution than the MILP.

As expected, the initial solution was better when heuristically constructed than randomly drawn for every instance, even for instances for which both initialization strategies did not result in a feasible solution. While every random initial solution was infeasible, the greedy heuristic was able to provide a feasible initial solution for half of the instances. Its ability in doing so diminished significantly with larger instances, with the greedy heuristic successfully computing a feasible solution for nine out of ten of the small instances, half of the medium ones and for only one large instance. However, even the infeasible initial solutions constructed by the greedy heuristic were closer to feasibility than their randomized counterparts in terms of a smaller penalty term as introduced in Section 5.3.

This resulted in a headstart of the variant starting with the heuristically constructed solution over the other, which can also be seen in Figure 6.2. In less than four seconds, VNS with greedy initialization was able to reach solutions as good as the other variant's final solution for half of the small instances. For the other four instances on which it outperformed the other variant, it took at worst 142 seconds to do so. On the medium instances, VNS with greedy initialization outperformed the other variant at latest after 72 seconds and in less than eight seconds on 80% of the test set. On the large instances, it took less than 45 seconds for 60% and less than 130 seconds for 90% of instances, with only one instance taking more than 20 minutes.

As shown by this experiment, greedy initialization overall leads to more feasible and better solutions, initial and final. Within the given time limit, the greedy initialization provides a headstart for VNS, hence leading to better solutions in less time. In the following we therefore always use the greedy heuristic over randomization for initialization.

6.2.2 Local search policy

After choosing the initialization method, we now turn to the local search policy in the next experiment. At choice are the Best Improvement and the First Improvement policy as

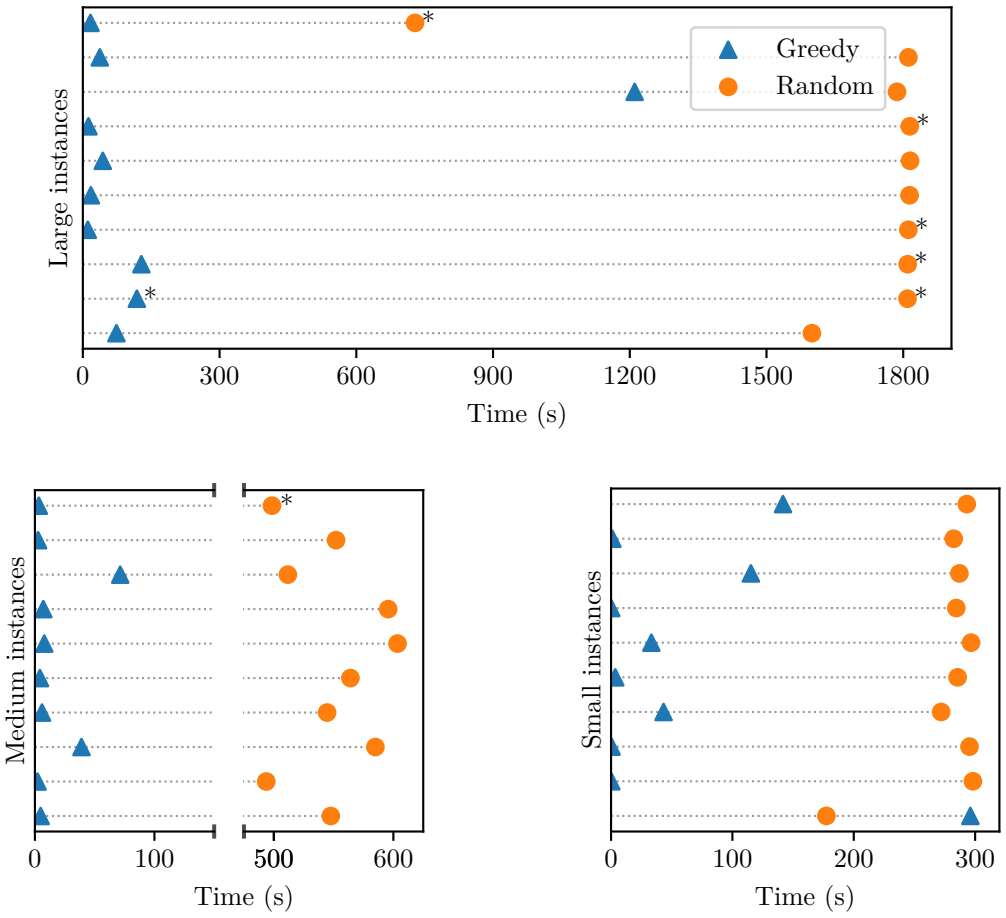


Figure 6.2: Comparison of time needed to reach a solution at least as good as the worse one of the two variants' final solutions, i.e., for all but the bottom small instance, the random variant's final solution. Infeasible final solutions of a variant on an instance are marked with an asterisk. Note the different time limits when comparing the subfigures.

presented in Section 5.1. In this experiment, VNS is run twice on the same new test set, once using a Best Improvement policy and once a First Improvement. For the initialization, we use the greedy heuristic, following the results of the first experiments. We repeat the experiment five times, each time with a different $k_{\max} \in \{2, 4, 6, 8, 10\}$.

As in the first experiment, the MILP was not able to find a feasible solution in the given time for three large instances in the test set. Both VNS variants were able to reach feasibility for all instances and all $k_{\max}$ except one medium instance. For that instance, both variants' final solutions were infeasible for all $k_{\max}$ due to overloaded components. The relative cost differences when comparing the final solutions of the two variants are relatively small, as can be seen in Table 6.3, making Best Improvement only marginally better than First Improvement. However, since the Best Improvement variant still led to better solutions for the majority of instances over all size categories and $k_{\max}$, as shown in Table 6.2, we use it as the local search policy moving forward.

6.2.3 Number of neighborhood structures

The final parameter that has to be chosen is the maximum number of neighborhood structures $k_{\max}$. To narrow its range down in a first step, we compare the performance for the different $k_{\max}$ of VNS with Best Improvement from the previous experiment.

		Instances			
$k_{\max}$	method	Small	Medium	Large	All
2	`BestImp`	70.00%	77.78%	60.00%	68.97%
	`FirstImp`	30.00%	22.22%	30.00%	27.59%
4	`BestImp`	70.00%	77.78%	60.00%	68.97%
	`FirstImp`	30.00%	22.22%	30.00%	27.59%
6	`BestImp`	80.00%	100.00%	70.00%	82.76%
	`FirstImp`	20.00%	0.00%	20.00%	17.24%
8	`BestImp`	80.00%	66.67%	70.00%	72.41%
	`FirstImp`	20.00%	33.33%	20.00%	24.14%
10	`BestImp`	60.00%	66.67%	70.00%	65.52%
	`FirstImp`	40.00%	33.33%	20.00%	31.03%

Table 6.2: Comparison of solution quality of Best and First Improvement for different $k_{\max}$. Rows with `BestImp` denote percentage of instances for which the Best Improvement policy achieves better solutions than First Improvement (rows with `FirstImp` vice versa).

	Instances			
$k_{\max}$	Small	Medium	Large	All
2	-1.10%	1.70%	2.38%	0.97%
4	-0.63%	1.31%	2.12%	0.92%
6	0.56%	2.62%	4.70%	2.63%
8	0.78%	1.74%	1.85%	1.45%
10	0.05%	1.91%	3.00%	1.64%

Table 6.3: Average relative cost difference of First Improvement to Best Improvement. The table does not include instances for which neither policy resulted in a feasible solution.

There are different measures for determining a good choice of $k_{\max}$. One way of comparison is the total amount of instances for which a $k_{\max}$ results in the best solution. As Table 6.4 shows, considering all instances except the one for which VNS could not find a feasible solution, $k_{\max} = 2$ delivers the best solution in the majority of cases.

Another measure are the average costs of the final solution, as shown by Figure 6.3. VNS with $k_{\max} = 6$ results in the lowest average relative costs for small and large instances and on all compared instances overall. While it might mostly not result in the best solution, in the average case, it reaches good solutions with costs close to the MILP's.

Finally, one could determine the best $k_{\max}$ by a ranked choice voting system, considering not only the $k_{\max}$ which results in the best solution for an instance, but also the second

	Instances			
$k_{\max}$	Small	Medium	Large	All
2	10.00%	55.56%	30.00%	31.03%
4	40.00%	22.22%	20.00%	27.59%
6	20.00%	11.11%	30.00%	20.69%
8	10.00%	0.00%	10.00%	6.90%
10	20.00%	11.11%	10.00%	13.79%

Table 6.4: Percentage of instances for which each $k_{\max}$ results in the best solution for VNS with Best Improvement. Colored entries signify the best value for a size category.

$k_{\max}$	score
2	71
4	71
6	57
8	43
10	48

Table 6.5: For every instance, each $k_{\max}$ scores points according to its reached solution quality. The colored entries contain the best total scores.

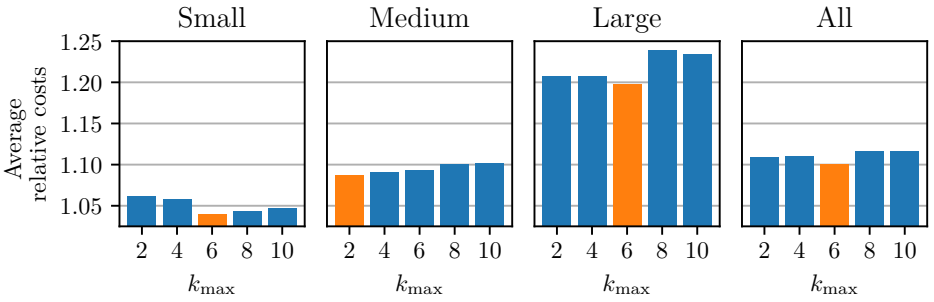


Figure 6.3: Average costs of VNS with Best Improvement relative to MILP for different $k_{\max} \in \{2, 4, 6, 8, 10\}$. The orange bars indicate the lowest average relative costs for a category. Only instances both MILP and VNS could solve to feasibility are included.

best and so forth. Inspired by Borda count [BCE+16], given $n = 5$ different $k_{\max}$, for each instance, we award the $k_{\max}$ leading to the best result with $n - 1$ points, the second best with $n - 2$ points and so forth until the worst $k_{\max}$, which receives zero points. Multiple $k_{\max}$ can share a place if they reach solutions with the same objective value. Infeasible solutions always lead to zero points. The added scores are listed in Table 6.5, the result was a tie of $k_{\max} = 2$ and $k_{\max} = 4$, with $k_{\max} = 6$ in the third place.

As different measures suggest different choices for $k_{\max}$, we ran VNS with Best Improvement on a new test set in a third experiment. As 2, 4 and 6 appear to be the most promising candidates from the previous experiment, we ran VNS on each instance with $k_{\max}$ ranging from 2 to 7. Except for one medium instance, VNS reached feasible solutions for all $k_{\max}$ on all instances the MILP could solve. On that medium instance, only $k_{\max} = 6$ and $k_{\max} = 3$ resulted in feasible solutions, the others resulted in infeasibility due to overloaded components. VNS was also able to solve two out of three large instances and one medium instance to feasibility the MILP could not.

While $k_{\max} = 5$ resulted in the best solution for most of the small instances, $k_{\max} = 6$ was the best choice for the other size categories, as well as when considering all instances (Table 6.6). When considering the average relative costs, $k_{\max} = 6$ again appears to be a good choice. While it is not the best choice when only considering small or large instances, it reaches the lowest average relative costs overall, as visualized by Figure 6.4.

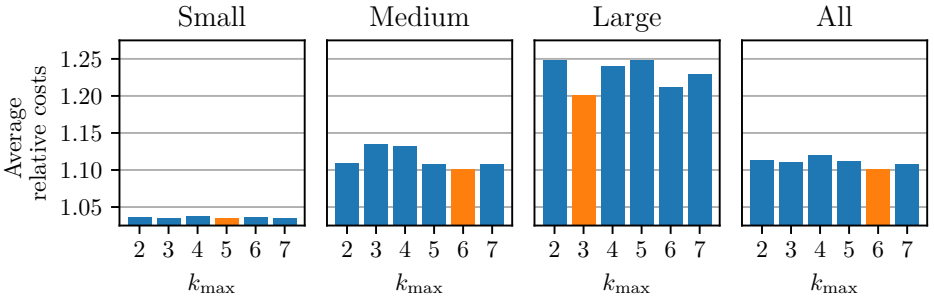


Figure 6.4: Average costs of VNS with Best Improvement relative to MILP for different $k_{\max}$ in the range of 2 to 7. The orange bars indicate the lowest average relative costs for a category. Only instances on which all $k_{\max}$ and the MILP resulted in feasible solutions are included.

	Instances			
$k_{\max}$	Small	Medium	Large	All
2	20.00%	22.22%	11.11%	17.86%
3	10.00%	0.00%	33.33%	14.29%
4	10.00%	22.22%	0.00%	10.71%
5	40.00%	0.00%	0.00%	14.29%
6	0.00%	33.33%	55.56%	28.57%
7	20.00%	22.22%	0.00%	14.29%

Table 6.6: Percentage of instances for which each $k_{\max}$ results in best solution for VNS. Colored entries signify the best value for a size category.

$k_{\max}$	score
2	77
3	64
4	61
5	68
6	76
7	68

Table 6.7: For every instance, each $k_{\max}$ scores points according to its reached solution quality. The colored entry contains the best total score.

Repeating the ranked choice voting with Borda count resulted in $k_{\max} = 2$ as the winner, but this time with $k_{\max} = 6$ as a close second (see Table 6.7).

Again, there does not seem to be a $k_{\max}$ that is clearly superior to the others in every compared aspect. However, each of our measurements suggests $k_{\max} = 6$ as a relatively good number of neighborhood structures. We therefore always choose $k_{\max} = 6$ in the following. As there does not seem to be clear relationship between instance size and best $k_{\max}$, we use the same $k_{\max}$ for every size category moving forward.

6.3 Comparison of VNS and Heuristic

After determining a good parameter setting for VNS, we compare it to the heuristic method proposed by Stampa [Sta22]. On a randomly selected test set of 35 instances per size category (and which have not yet been used for the other experiments), we run VNS five times with different random seeds on each instance. As before, the time limits for VNS are five minutes on small instances, ten on medium and 30 on large ones. The heuristic was given a time limit of one hour for each instance.

In the following, "average" values for a single instance always refer to the median of the five results for that instance. Averages over multiple instances, e.g. for an entire size category, always refer to the arithmetic mean over those median values unless stated otherwise. A run refers to a single execution of VNS on one particular instance. Averages over all runs of multiple instances also refer to the arithmetic mean.

The MILP was not able to find a feasible solution for nine out of the 35 large instances. Out of these nine instances, three could also not be solved by VNS and the heuristic. On three large and one medium instance, only the heuristic was not able to find a feasible solution, failing at finding a solution that did not violate an inverter's lower bound. When VNS reached a feasible solution for an instance, it did so for all runs on that instance. For all large and small instances that were solved to feasibility by both VNS and the heuristic, the former method achieved better solutions for all runs than the latter. On the medium instances, VNS reached better solutions in 98% of all runs, only being outperformed by the heuristic for one out of the five runs on three different instances.

Comparison with respect to runtime

First, for every instance, we compare the runtime needed by VNS to reach a solution at least as good as the heuristic's (Figure 6.5). For the three large instance and the one medium instance only the heuristic failed at, we use the time it took for VNS to reach a feasible solution.

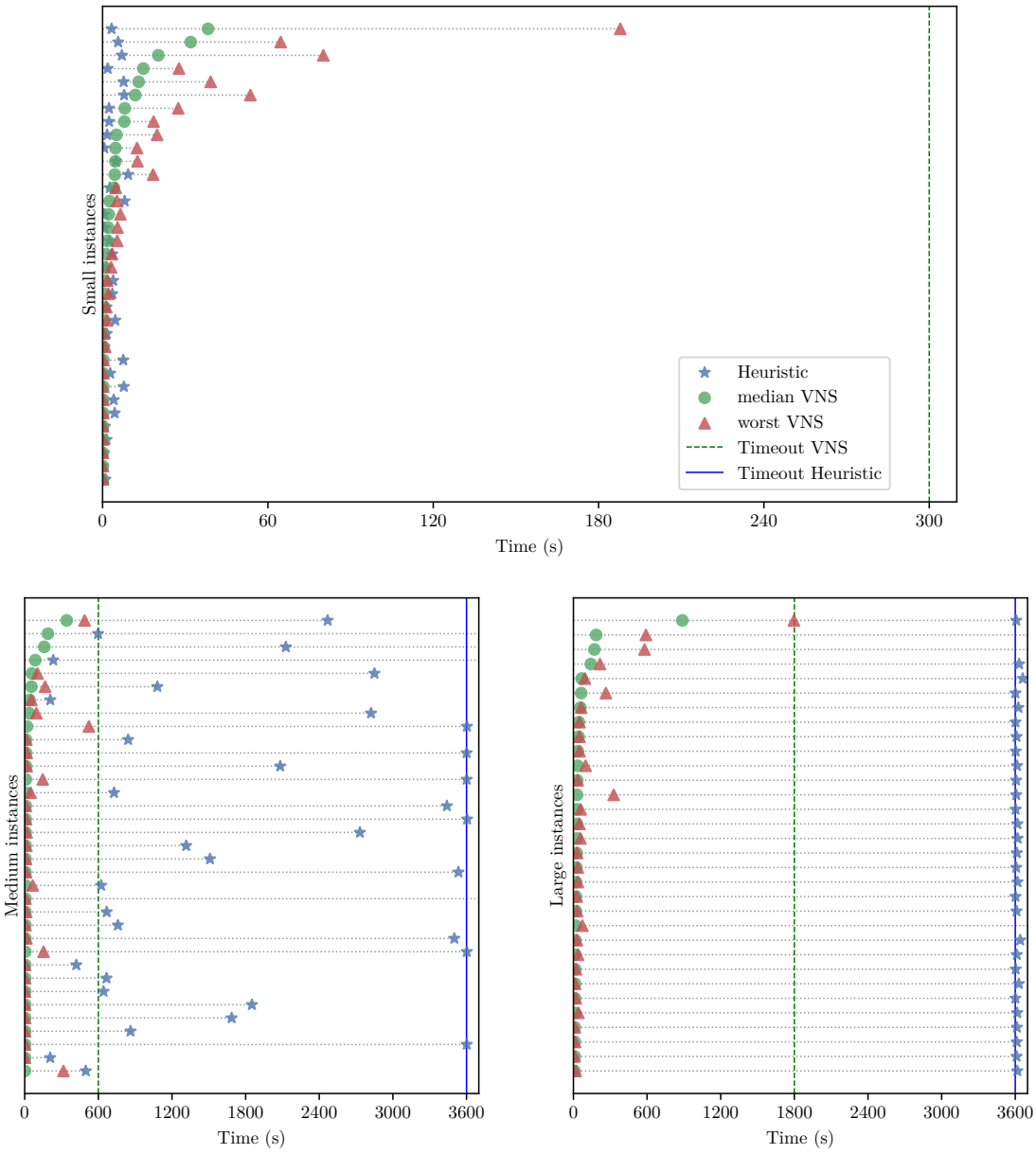


Figure 6.5: Comparison of time needed by VNS to reach the heuristic's final solution quality per instance. For each instance on the y-axis the worst and average runtime of VNS as well as the heuristic's runtime is depicted on the x-axis. Only instances VNS found a feasible solution for are included. The instances are sorted by the median runtime of VNS in descending order from top to bottom. Points for the heuristic are missing for the one medium and three large instances it could not solve to feasibility. Points for the worst VNS runtime are missing for the three medium on which VNS could not outperform the heuristic on one run. Note the different time limits when comparing the subfigures.

For the small instances, the initial solution provided by the greedy construction heuristic, which takes on average 0.16 seconds to complete, is on average already at least as good as the heuristic's final solution in 26% of all runs. The heuristic terminates on average after 3.41 seconds for the given small instances. The average runtime of VNS to reach a solution as good as the heuristic's is slightly more (5.28 seconds on average over all runs). VNS reaches a solution as good as the heuristic's in less or equal time in 71% of runs. In the other runs, the heuristic terminates earlier, however, VNS still outperforms the heuristic

later on, in the worst case 184.62 seconds and on average 13.73 seconds after the heuristic terminates. For the medium instances, the initial solution of VNS, which takes on average 2.91 seconds to compute, is already at least as good as the heuristic's solution in 19% of runs. For the medium instances, the heuristic's average runtime increases significantly to 1824.6 seconds, with it not completing after its maximum runtime of one hour for 20% of instances (counting in the one instance the heuristic did not solve). In the aforementioned 98% of runs in which VNS outperforms the heuristic on the medium instances, VNS reaches a solution as good as the heuristic's after 38.60 seconds on average.

The heuristic was not able to terminate before reaching its runtime limit of one hour on any of the 29 large instances it reached feasibility for. VNS reached a solution at least as good as the heuristic's on average after 77.61 seconds. In 23% of runs it already did so with the initial solution, which took on average 12.03 seconds to compute.

On the small instances, the average runtime of the heuristic to terminate and of VNS to reach the former's solution quality are very close. However, on the medium instances the average runtime of VNS increases almost eightfold, while the average runtime of the heuristic rises to about 500 times its time on the small instances. For the large instances, the heuristic's runtime reaches the time limit of one hour on all instances, while VNS' average runtime to outperform the heuristic is still little more than one minute.

In summary, VNS leads to better solutions in less time than the heuristic on most instances. On some small instances, VNS takes slightly longer to outperform the heuristic's solution, but still does so later on. The heuristic's runtime is also more negatively affected by increasing instance size than the VNS.

Comparison with respect to solution quality

The heuristic's performance is more negatively affected by increasing instance size than VNS, not only in terms of runtime but also solution quality, as shown in the following.

We compare the costs of the solutions found by VNS and the heuristic to the MILP (fig. 6.6). Table 6.8a shows the average costs of the solutions computed by VNS and the heuristic relative to the MILP's costs per size category. While they are on average worse than the MILP's solutions, VNS' solutions have a smaller average cost ratio than the heuristic's for all size categories. Additionally, the heuristic's average cost ratio increases more with larger instances than the VNS'. The average cost ratio still increases for both, but for VNS to a lesser extent than for the heuristic. Additionally, the ratios for the different instances are far more spread for the heuristic than for VNS. Therefore, the VNS appears to deliver not only more stable results across the different size categories, but also within them.

Similar observations can be made when grouping the instances also within the size categories, e.g. by amount of edges and vertices in the solar farm graph (Figure 6.7). For both methods, there is a relationship between amount of edges or vertices and relative cost, as both generally achieve worse relative costs for instances with more edges and/or vertices. However, the heuristic clearly struggles more with an increasing amount of either parameter.

Besides comparing the costs of both methods to the MILP's solution, we also compare the costs of VNS to the heuristic's directly (Table 6.8b). This also allows for the inclusion of the six large instances that could not be solved by the MILP, but by the two other methods. For small instances, the average ratio is slightly less than one, meaning that VNS and the heuristic find solutions of almost the same quality. However, VNS is still always better than the heuristic, as even the worst solution on a small instance is still less expensive than its heuristic counterpart. For medium instances, VNS sometimes results in worse solutions, but not on more than 10% of medium instances. Additionally, the worse solutions never cost more than 111% of the heuristic's solution's cost. On average, the solutions reached

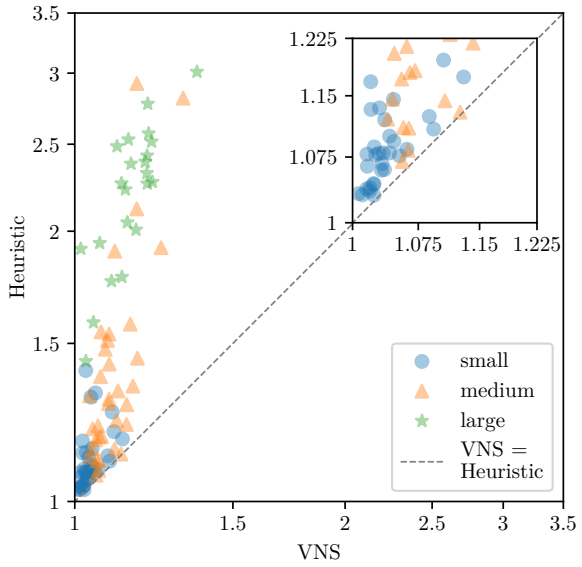


Figure 6.6: Cost ratio (relative to MILP) of VNS and heuristic per instance. Only instances that were solved to feasibility by all methods are included. The cost ratio of VNS for an instance is the median cost ratio over all five runs on that instance. Both axes of the main window and the close up use a logarithmic scale.

	VNS	Heuristic
large	1.1684	2.2794
medium	1.1058	1.4331
small	1.0408	1.1135

(a) Average costs of VNS and heuristic relative to MILP. For all size categories, VNS achieves a smaller average cost ratio than the heuristic. Both methods' ratios increase for larger instances, but VNS' one does so to a lesser extent.

	avg	worst
large	0.5261	0.7308
medium	0.8135	1.1067
small	0.9391	0.9945

(b) Average costs of VNS relative to heuristic. The second column contains the worst cost ratio over all instances of that size category. VNS is only on the medium instances not always better than the heuristic. The larger the instances, the smaller are the relative costs on average.

Table 6.8: Comparison of average costs of VNS and heuristic per size category, on the left relative to the MILP, on the right relative to the heuristic. Only instances all methods could reach a feasible solution for are considered. "Average" again refers to the arithmetic mean (for VNS over the median over all five runs per instance).

by VNS on the medium instances cost about 19% less than the heuristic's. On the large instances, VNS was able to find solutions that were on average almost half the cost of the heuristic's and at least 26% less. With increasing instance size, VNS therefore becomes more favorable compared to the heuristic.

While this section focused on the comparison of VNS to the heuristic, we briefly want to compare VNS directly to the MILP as well (Figure 6.8). This allows us to include the three large and one medium instance the heuristic could not reach feasibility on. As mentioned above, VNS was on average not able to outperform the MILP on those instances that both methods could solve to feasibility. VNS reached better solutions than the MILP in at least one of five runs only on one medium and two large instances. Again, VNS finds comparatively worse solutions with increasing instance size. The solutions found by VNS cost on average 4% more than the MILP's solutions on the 35 small instances, 11% more on the 35 medium and 21% more on the 26 compared large instances. Additionally, the larger the instance size category, the larger the range of relative costs reached by VNS. Over all runs, VNS reached cost ratios ranging from 1.0047 to 1.2208 on small, 0.70518 to 1.4269 on medium, and 0.9699 to at worst even more than two on one run with 2.0341 on large instances. It should be noted that we set a time limit of 10 minutes on medium and 30 minutes on large instances for VNS, while Gurobi terminated on average after slightly

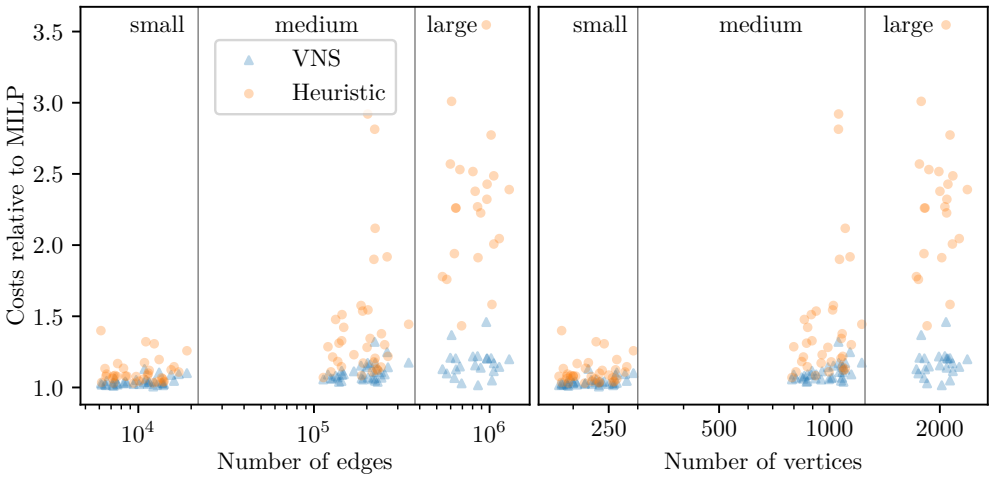


Figure 6.7: Cost of VNS and heuristic relative to MILP depending on number of edges and vertices (of graph of the solar farm instance). The x-axes use a logarithmic, the y-axes a linear scale. Only instances that were solved to feasibility by all methods are included. The cost ratio of VNS for an instance is the median cost ratio over all five runs on that instance.

more than 20 hours on the medium instances and 24 hours on the large ones. However, the MILP terminated on average after 34.62 seconds on the small instances, while VNS was not able to outperform the MILP on any of them after five minutes. Furthermore, VNS was able to solve six additional large instances to feasibility that the MILP could not. VNS reached better solutions than the heuristic on these instances, making its solutions the best available for them so far.

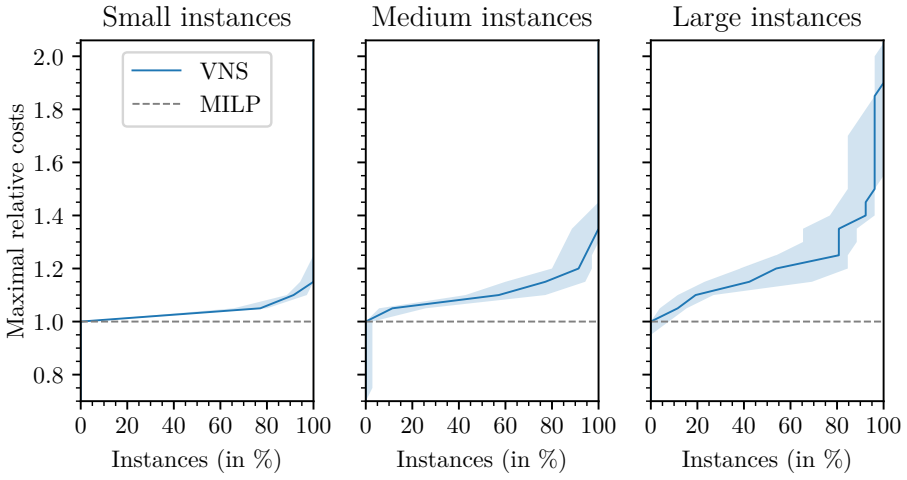


Figure 6.8: Percentage of instances per cost relative to the MILP that VNS solved to an equal or smaller cost ratio. Plotted lines show the median costs per instance, the borders of the colored areas are the worst (upper bound) and best (lower bound) costs. Only the instances that were solved to feasibility by Gurobi and VNS are included.

In summary, VNS reached feasible solutions on more instances than both the heuristic and the MILP. In terms of solution quality, VNS was less negatively affected by increasing instance size than the heuristic. Additionally, the average costs of VNS relative to the

MILP were more stable across different instances of the same size category than those of the heuristic. On most instances, VNS could not outperform the solutions found by the MILP. However, for all runs on six large instances and for at least one run on two large and one medium instance, VNS reached the best available solution so far.

6.4 Combining VNS and Heuristic

In Section 6.2, we observed that initializing VNS with our greedy heuristic resulted on average in better final solutions in less time than when starting from a random solution instead. Additionally, we noted that the greedy heuristic consistently provided better initial solutions than randomization. We concluded that the better initial solutions by the greedy heuristic gives VNS an advantage over the randomized variant, which contributed to the better solutions in the given time.

In Section 6.3, the heuristic's final solutions were better than our greedy heuristic's initial ones in 74% to 81% of runs for all size categories. After the comparison of greedy to random initialization, one might assume that better initial solutions for VNS lead to better final solutions overall. To test this hypothesis, we ran VNS again on the same instances used in the last experiment, this time using the heuristic's final solutions for these instances as VNS' initial ones. We excluded the six large instances and one medium instance of the test set that could not be solved by the heuristic. We again ran VNS five times on each instance with the same time limits as before.

On all runs on all instances, VNS was able to improve the initial solution provided by the heuristic. This can be seen in Figure 6.9, which compares the costs of the heuristic's solutions relative to the MILP's before and after applying VNS to it. It should be noted that the heuristic terminated on its own, not because of the time limit, on all small and on 28 of the 34 medium instances. This means that on those instances, VNS was able to further improve solutions the heuristic could not.

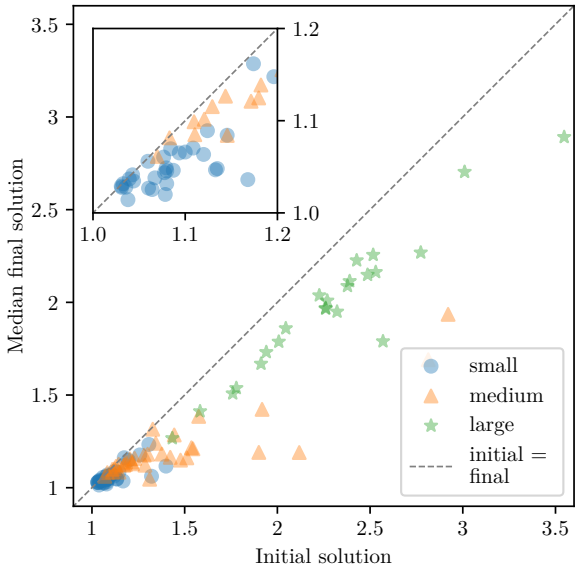


Figure 6.9: Average costs relative to MILP before (x-axis) and after (y-axis) running VNS on the heuristic's solutions per instance. For all instances, VNS was able to lower the initial costs. The larger the instance and the more expansive the initial solution compared to the MILP, the larger was the improvement. Only instances all methods found feasible solutions on are included.

As shown in Table 6.10, VNS was able to improve the heuristic's solutions to a lesser extent on small instances, reaching solutions that cost on average 96% of the initial cost. In comparison, VNS was able to lower the cost on average to 87% of the initial cost on medium and 88% on large instances. This result does make sense, as the heuristic's

	VNS	Heuristic	Combined
large	1.1684	2.2794	1.9718
medium	1.1058	1.4331	1.2052
small	1.0408	1.1135	1.0635

Table 6.9: Average costs relative to MILP per method and size category. The combination of VNS and heuristic reaches better ratios on average than the heuristic alone, but does not outreach VNS with greedy initialization.

	avg
large	0.8682
medium	0.8730
small	0.9600

Table 6.10: Average costs of VNS on heuristic's solutions relative to the costs of those solutions on their own. VNS was able to improve all the heuristic's solutions on average. The larger the instances, the larger the improvement.

solutions on small instance are already relatively close to the MILP's solutions, making further improvements rarer to find.

Comparing the median cost per instance, VNS' final solutions when starting from the heuristic's were worse than those of VNS with greedy initialization from Section 6.3 on all large, 94% of medium and 89% of small instances. As shown in Figure 6.10 and Table 6.9, this places the performance of the combination of VNS and the heuristic in terms of solution quality between the two methods in their "pure" forms. The increasing percentage for larger instances could be explained by the increasing cost difference between VNS and the heuristic for larger instances observed in the last experiment.

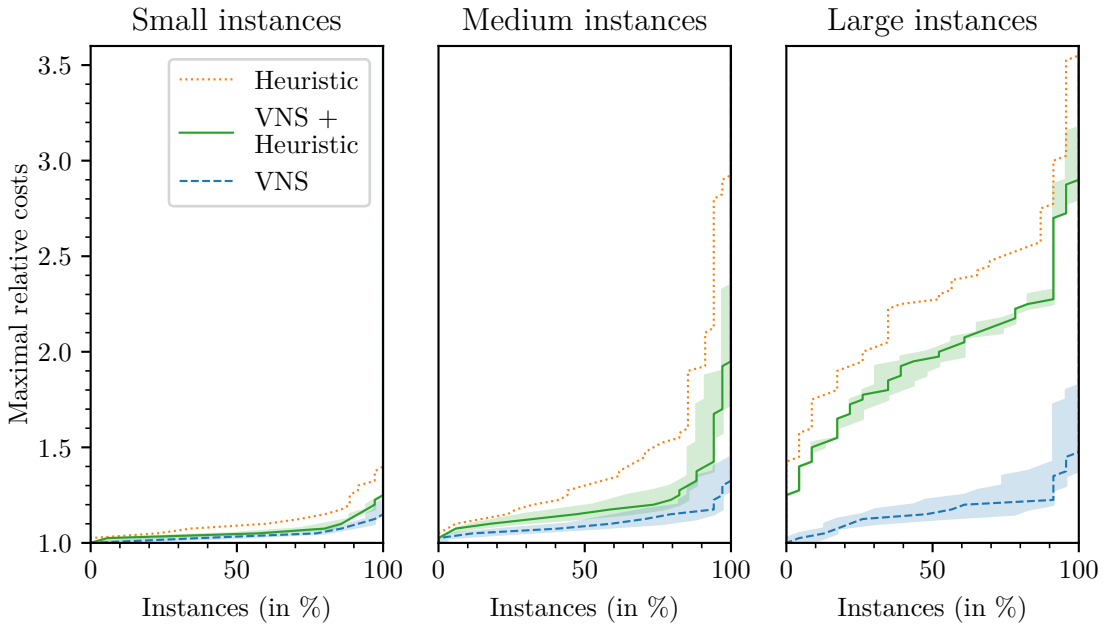


Figure 6.10: Percentage of instances (x-axis) each method reaches costs relative to the MILP at least as good as a given cost ratio (y-axis) for. Plotted lines show the median costs for each instance, the borders of the colored areas are the worst (upper bound) and best (lower bound) costs. Only the instances that were solved by the heuristic are included. The label VNS refers to VNS with greedy initialization in Section 6.3. The heuristic and VNS on the heuristic's solutions behave similarly for increasing instance size, as the latter directly depends on the former. VNS with greedy initialization is less affected by increasing instance sizes.

This result might be surprising, when considering that on most instances, the heuristic's final solutions are better than the greedily constructed solutions as mentioned above. We therefore might expect the former to be a more advantageous starting point than the latter. One possible explanation is that better solutions are harder to escape from. If a solution is already very good, it could be more difficult to find further improvements in its close neighborhood. The greedy heuristic appears to offer a good middle ground between total randomization, which often results in infeasible solutions, that take a lot of time to repair, and already very optimized solutions, that are difficult to improve further. Another factor could be that the greedy construction heuristic prioritizes choosing the shortest possible edges, while the heuristic prioritizes packing components evenly in its construction phase. The first strategy might lead to initial solutions that are worse in themselves but closer to overall better solutions.

6.5 Summary and Discussion

On average, VNS was able to find better solutions in less time than the heuristic. VNS' solutions also worsen with increasing instance size, but to a lesser extent than the heuristic's. While VNS was on average not able to outperform the MILP in terms of solution quality, it was able to find solutions for more instances than the MILP within the given time limit. On average, VNS reached solutions that cost 4% more than the MILP's solutions on the small, 11% more on the medium and 21% more on the large instances. While we chose a time limit for our VNS that was significantly smaller than the MILP's, the MILP is clearly superior even in terms of running time on small instances. On those instances, the MILP found solutions in less than a minute on average that were better than what VNS could reach after five minutes.

The most successful configuration of VNS proved to be the initialization with the greedy heuristic, a Best Improvement policy for the local search and setting $k_{\max}$ to 6. The most crucial component for the success of our final VNS appears to be the choice of the initial solution. As seen in Section 6.2, starting from heuristically constructed solutions as opposed to random ones clearly improved the solutions' quality and resulted in more feasible solutions overall. The choice of the local search policy and $k_{\max}$ affected the quality of the found solutions to a lesser extent, with an exception to one instance, for which only some $k_{\max}$ resulted in feasibility.

One reason for the success of VNS over the heuristic could be that VNS allows temporary moves from feasible to infeasible solutions, namely as the result of the shaking phase. This enables VNS to break out of local minima. In comparison, the heuristic tries to construct a feasible solution and improve this solution while maintaining feasibility. Another advantage of our VNS is the usage of a dynamic penalty term for infeasible solutions. Even if a local improvement on an infeasible solution does not result in feasibility, it can still improve the current solution's quality in terms of cost or the penalty term. This allows VNS to incrementally move from infeasible towards feasible solutions and might make it less susceptible to getting stuck at both feasible and infeasible solutions. In Section 4.3, we briefly described the way the heuristic tries to improve its initial solution: for every vertex with an outgoing cable, the heuristic calculates the cost difference that would result from switching that cable to a different successor. It then chooses the change that results in the largest cost reduction while maintaining feasibility and repeats the process until no further improvement can be made. This is effectively the same as what our VNS does in its local search phase under a Best Improvement policy while using the recabling move introduced in Section 5.1. However, while the heuristic terminates if it can make no more moves to a better solution, i.e. if it found a local minimum, VNS can find further improvements through its shaking process, which we also briefly explained in Section 4.3. In Section 6.4,

we found practical instances for which this was the case. Additionally, while the version of Best Improvement formulated in Section 5.1 only considers one random recabling move per cable, the heuristic considers every possible new successor per component. This could be a reason why the heuristic's runtime increases so much and why the quality of the heuristic's solutions decreases more than VNS' solutions for larger instances.

7. Conclusion

In this thesis, we have proposed an application of the metaheuristic VNS to the Solar Farm Cable Layout Problem. We have formulated a greedy heuristic for constructing initial solutions for a given instance that proved to contribute to the success of our VNS. When comparing different configurations for VNS, in terms of the initial solution, local search policy and $k_{\max}$, we found a combination of greedy initialization, following a Best Improvement policy and using $k_{\max} = 6$ to be the most successful. We then compared the performance of this VNS version with an existing heuristic.

On average, VNS was able to find better solutions than that heuristic in less time. Both VNS' and the heuristic's solution quality decreased on larger instances. For example, our final VNS variant reached solutions that cost on average 4% more on small, 11% more on medium and 21% more on large instances than the MILP solutions. However, the heuristic was even more negatively affected by increasing instance size, making the VNS more advantageous the larger the instances get. On some instances that were not solved by the MILP (but not proven to be infeasible either), VNS could find feasible solutions better than the heuristic's, making those solutions the best ones available so far. Occasionally, VNS was even able to outperform existing MILP solutions, but on most instances it did not. It should be noted that VNS was given time limits of at most 30 minutes in comparison to the time limit of one day for the MILP. On the small instances however, the MILP was clearly superior both in terms of solution quality and runtime, as it found better solutions in less time than our VNS. While our VNS was for the most part not better than the MILP in terms of solution quality, it still achieves solid solutions in its given time overall. Additionally, it is the best solution method so far for some of the instances that the MILP fails at.

We defined a $k-$neighborhood of a given solution x as the set of all solutions that result out of the successive recabling of k components in x. Alternative neighborhood structures not explored in this thesis could be further researched and compared or combined, which might improve our VNS further. Possible operations for neighborhoods could be recabling of entire paths, adding and removing components or swapping of subtrees.

One disadvantage of our greedy heuristic is that it does not guarantee feasibility. While VNS with greedy initialization was able to reach feasibility on most instances when starting off from an infeasible solution, this was not the case for one large instance and one medium instance. We only tested the heuristic's *final* solutions as initial solutions for VNS. However, not only did this combination not outperform VNS with greedy initialization, its combined

runtime on large instances is very large, as the heuristic takes more than one hour for those. Another weakness of the heuristic that became apparent in its comparison to VNS is its improvement strategy. It is both time-consuming and susceptible to terminating with suboptimal local minima. One could therefore try out starting from the heuristic's initial solution, which are more often feasible than the greedy heuristic's, and replacing the heuristic's improvement steps with VNS.

We also concluded that better initial solutions in terms of costs do not generally result in better final solutions of VNS. Instead, the graph attributes of the initial solution appear to matter as well. We stated that one possible reason for the success of VNS with greedy initialization over VNS on the heuristic's final solutions could be their different strategies in connecting the components. The greedy heuristic prioritizes choosing cables of minimal length, which the heuristic does not consider. One could investigate this further by adapting the heuristic in such a way that it also prioritizes minimum lengths.

As mentioned in Chapter 2, numerous extensions of the basic VNS used in this thesis exist. VNS is designed as a local search method that deploys different neighborhood structures in order to escape local minima. Drawing neighbors completely at random as our VNS often does not lead to improvements. One of many possible avenues of improving VNS could therefore be using so-called *intensified shaking*, which tries to choose neighbors more promising than others.

We only applied our VNS to solar farms with fully connected layers so far. However, our formulation should be easily extendable to other solar farms as well, for example by storing a list of all allowed successors of each component to consider during the initialization and recabling. Furthermore, this thesis focused on taking a metaheuristic approach to an already formulated problem. To compare our VNS to existing methods, we adapted the model proposed by [Sta22]. However, as pointed out in detail in that work, this model could also be developed further. Some aspects the model misses were also briefly mentioned by us in Chapter 4, e.g. power losses due to cabling, an important factor in the real-life efficiency of solar farms. If alternative models are formulated in the future, our VNS might be extendable to those as well.

Bibliography

[BCE+16] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia. Introduction to Computational Social Choice. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, pages 1–20. Cambridge University Press, New York (NY), 2016.

[CFF20] Davide Cazzaro, Martina Fischetti, and Matteo Fischetti. Heuristic algorithms for the Wind Farm Cable Routing problem. *Applied Energy*, 278, 2020. Article No. 115617. doi:10.1016/j.apenergy.2020.115617.

[EABB18] Amro M. Elshurafa, Shahad R. Albardi, Simona Bigerna, and Carlo Andrea Bollino. Estimating the learning curve of solar PV balance–of–system for over 20 countries: Implications and policy recommendations. *Journal of Cleaner Production*, 196:122–134, 2018. doi:10.1016/j.jclepro.2018.06.016.

[FRA+14] Diogo R.M. Fernandes, Caroline Rocha, Daniel Aloise, Glaydston M. Ribeiro, Enilson M. Santos, and Allyson Silva. A simple and effective genetic algorithm for the two-stage capacitated facility location problem. *Computers & Industrial Engineering*, 75:200–208, 2014. doi:10.1016/j.cie.2014.05.023.

[GKS15] Bernard Gendron, Paul-Virak Khuong, and Frédéric Semet. Multilayer variable neighborhood search for two-level uncapacitated facility location problems with single assignment. *Networks*, 66(3):214–234, 2015. doi:10.1002/net.21626.

[GSW22] Sascha Gritzbach, Dominik Stampa, and Matthias Wolf. Solar farm cable layout optimization as a graph problem. *Energy Informatics*, 5, 2022. Article No. 25. doi:10.1186/s42162-022-00200-z.

[GUW+19] Sascha Gritzbach, Torsten Ueckerdt, Dorothea Wagner, Franziska Wegner, and Matthias Wolf. Engineering Negative Cycle Canceling for Wind Farm Cabling. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144, pages 55:1–55:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. Available at https://arxiv.org/abs/1908.02129, last accessed on October 29, 2022.

[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, pages 299–307. Association for Computing Machinery, 2020. doi:10.1145/3396851.3397754.

[IPC14] IPCC. Climate Change 2014: Mitigation of Climate Change. Contribution of Working Group III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change. Technical report, Intergovernmental Panel on Climate Change, 2014. Available at https://www.ipcc.ch/report/ar5/wg3/, last accessed on October 29, 2022.

[IRE21] IRENA. Renewable Power Generation Costs in 2020. Technical report, International Renewable Energy Agency, 2021. Available at `https://www.irena.org/publications/2021/Jun/Renewable-Power-Costs-in-2020`, last accessed on October 29, 2022.

[Jen20] Marc Jenne. Minimum-Cost Flow Algorithms for the Wind Farm Cabling Problem. Bachelor thesis, Karlsruhe Institute of Technology, April 2020.

[JH22] Yaman A. Jieb and Eklas Hossain. *Photovoltaic Systems: Fundamentals and Applications*. Springer International Publishing, Cham, 2022. `doi:10.1007/978-3-030-89780-2`.

[Lju21] Ivana Ljubić. Solving Steiner trees: Recent advances, challenges, and perspectives. *Networks*, 77(2):177–204, 2021. `doi:10.1002/net.22005`.

[LQC+21] Zhixing Luo, Hu Qin, Edwin T. C. Cheng, Qinghua Wu, and Andrew Lim. A Branch-and-Price-and-Cut Algorithm for the Cable-Routing Problem in Solar Power Plants. *INFORMS Journal on Computing*, 33(2):452–476, 2021. `doi:10.1287/ijoc.2020.0981`.

[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, pages 203–215. Association for Computing Machinery, 2017. `doi:10.1145/3077839.3077843`.

[Mar10] Miroslav Marić. An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem. *Computing and Informatics*, 29(2):183–201, 2010. Available at `https://www.cai.sk/ojs/index.php/cai/article/view/80`, last accessed on October 29, 2022.

[MH97] Nenad Mladenović and Pierre Hansen. Variable Neighborhood Search. *Computers & Operations Research*, 24(11):1097–1100, 1997. `doi:10.1016/S0305-0548(97)00031-2`.

[MHP08] Nanid Mladenović, Pierre Hansen, and José A. Moreno Pérez. Variable neighbourhood search: methods and applications. *4OR. A Quarterly Journal of Operations Research*, 6(4):319–360, 2008. `doi:10.1007/s10288-008-0089-1`.

[MRRP00] Simone L. Martins, Mauricio G.C. Resende, Celso C. Ribeiro, and Panos M. Pardalos. A Parallel Grasp for the Steiner Tree Problem in Graphs Using a Hybrid Local Search Strategy. *Journal of Global Optimization*, 17:267–283, 2000. `doi:10.1023/A:1026546708757`.

[MSDS14] Miroslav Marić, Zorica Stanimirović, Aleksandar Djenić, and Predrag Stanojević. Memetic Algorithm for Solving the Multilevel Uncapacitated Facility Location Problem. *INFORMATICA*, 25(3):439–466, 2014. `doi:10.15388/Informatica.2014.23`.

[NSM17] Susan Neill, Geoff Stapleton, and Christopher Martell. *Solar farms: The Earthscan Expert Guide to Design and Construction of Utility-scale Photovoltaic Systems*. Earthscan Expert Series. Routledge, London, New York, 2017.

[OACL18] Camilo Ortiz-Astorquiza, Ivan Contreras, and Gilbert Laporte. Multi-level facility location problems. *European Journal of Operational Research*, 267(3):791–805, 2018. `doi:10.1016/j.ejor.2017.10.019`.

[RUW02] Celso C. Ribeiro, Eduardo Uchoa, and Renato F. Werneck. A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs. *INFORMS Journal on Computing*, 14(3):228–246, 2002. `doi:10.1287/ijoc.14.3.228.116`.

[Sta22] Dominik Stampa. Theory and Algorithms of the Solar Farm Cable Layout Problem. Master's thesis, Karlsruhe Institute of Technology, February 2022.

[UW12] Eduardo Uchoa and Renato F. Werneck. Fast Local Search for the Steiner Problem in Graphs. *ACM Journal of Experimental Algorithmics*, 17(2), 2012. Article No. 2.2. doi:10.1145/2133803.2184448.

[VMB+20] Eero Vartiainen, Gaëtan Masson, Christian Breyer, David Moser, and Eduardo Román Medina. Impact of weighted average cost of capital, capital expenditure, and other parameters on future utility-scale PV levelised cost of electricity. *Progress in Photovoltaics: Research and Applications*, 28(6):439–453, 2020. doi:10.1002/pip.3189.

[WRS00] Austin S. C. Wade and Vic J. Rayward-Smith. Effective local search techniques for the steiner tree problem. In Ding-Zhu Du, James M. Smith, and Joachim H. Rubinstein, editors, *Advances in Steiner Trees*, volume 6 of *Combinatorial Optimization*, pages 255–281. Springer US, Boston (MA), 2000. doi:10.1007/978-1-4757-3171-2_12.