TU WIEN Informatics

Heuristic Optimization of Integrated Energy Systems

An Extension to the IESopt Framework

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Artificial Intelligence

by

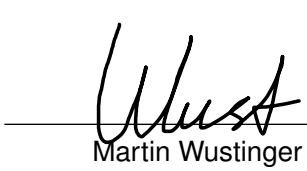
Martin Wustinger, B.Sc.
Registration Number 12122402

to the Faculty of Informatics

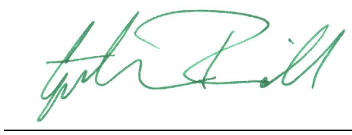
at the TU Wien

Advisor: Ao.Univ.-Prof. DI. Dr.techn. Günther Raidl
Assistance: Mag. DI. Dr.techn. Matthias Prandtstetter
DIin. Ulrike Ritzinger, PhD
DI. Hannes Koller

Vienna, August 26, 2026



Martin Wustinger



Günther Raidl

TU WIEN Informatics

Heuristische Optimierung von Integrierten Energiesystemen

Eine Erweiterung des IESopt Frameworks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Artificial Intelligence

eingereicht von

Martin Wustinger, B.Sc.
Matrikelnummer 12122402

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.-Prof. DI. Dr.techn. Günther Raidl
Mitwirkung: Mag. DI. Dr.techn. Matthias Prandtstetter
DIin. Ulrike Ritzinger, PhD
DI. Hannes Koller

Wien, 26. August 2026

Martin Wustinger

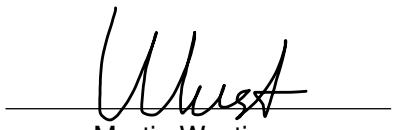
Günther Raidl

Declaration of Authorship

Martin Wustinger, B.Sc.

I hereby declare that I have written this thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work – including tables, maps and figures – which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, August 26, 2026



Martin Wustinger

Acknowledgements

I would like to take this opportunity to thank everyone who supported me in writing this thesis and accompanied me throughout the entire process.

First and foremost, I would like to thank my supervisors, Matthias Prandtstetter at the AIT Austrian Institute of Technology and Günther Raidl at TU Wien. Without your support, it would not have been possible for me to write this thesis in its current form. Matthias, thank you for always answering my questions concerning various topics in the field of energy systems and for helping me find a topic that is not only relevant to the AIT Austrian Institute of Technology but also personally interesting to me. I would also like to thank Günther for continuing to support me after my successful Bachelor's thesis and the Bachelor of Honors program, and for making it possible for me, from the TU Wien side, to write my Master's thesis outside TU Wien.

Next, I would like to thank my direct project team. Ulrike and Hannes were not only involved in numerous discussions and contributed their ideas and suggestions, but also actively supported me in developing both the heuristics and the framework itself. Thank you for your help, the many discussions, and the pleasant collaboration.

Finally, I would like to thank everyone in the IES department for creating such a pleasant working environment. To everyone from the department who may read this thesis. You are all wonderful people, and I would be very happy to do something together again in the future, whether in a professional or private setting.

Danksagung

Ich möchte mich hiermit bei allen Personen bedanken, die mich beim Verfassen dieser Arbeit unterstützt und während der gesamten Zeit begleitet haben.

Beginnen möchte ich mit meinen Betreuern Matthias Prandtstetter am AIT Austrian Institute of Technology und Günther Raidl an der TU Wien. Ohne eure Unterstützung wäre es mir überhaupt nicht möglich gewesen, diese Arbeit in dieser Form zu verfassen. Matthias hat mich bei Fragen zu verschiedenen Themen im Bereich der Energiesysteme stets unterstützt und sich dafür eingesetzt, ein Thema zu finden, das nicht nur für das AIT Austrian Institute of Technology relevant, sondern auch für mich persönlich interessant ist. Bei Günther möchte ich mich insbesondere dafür bedanken, dass du mich auch nach meiner erfolgreichen Bachelorarbeit und dem Bachelor-of-Honors-Programm weiterhin tatkräftig unterstützt hast und es mir seitens der TU Wien ermöglicht hast, meine Masterarbeit außerhalb der TU Wien zu verfassen.

Als Nächstes möchte ich mich bei meinem direkten Projektteam bedanken. Ulrike und Hannes waren nicht nur bei zahlreichen Besprechungen dabei und haben ihre Ideen und Vorschläge eingebracht, sondern haben mich auch bei der Entwicklung der Heuristiken und des Frameworks selbst tatkräftig unterstützt. Vielen Dank für eure Hilfe, die zahlreichen Diskussionen und die angenehme Zusammenarbeit.

Zuletzt möchte ich mich noch für das angenehme Arbeitsumfeld in der IES-Abteilung bedanken. An alle, die diese Arbeit aus der Abteilung möglicherweise lesen. Ihr seid allesamt sehr liebe Menschen, und ich würde mich freuen, in Zukunft wieder einmal etwas mit euch zu unternehmen – ganz egal, ob im beruflichen oder privaten Umfeld.

Abstract

While energy systems can be the origin of a wide variety of optimization problems, many of the underlying components are shared across different models. This observation motivated the Integrated Energy Systems unit at the AIT Austrian Institute of Technology to develop the IESopt modeling framework, which aims to simplify the modeling of such systems. However, this approach also has limitations. Since IESopt ultimately relies on a mathematical optimization solver to optimize the constructed model, the computational effort required to obtain high-quality solutions can increase substantially with model complexity.

Heuristic optimization is therefore investigated in this thesis as a means of addressing these limitations and generating high-quality solutions for even very complex instances within reasonable computational times. To this end, we propose a heuristic framework that, following the philosophy of IESopt, aims to support developers in designing heuristic methods for specific optimization problems. We further investigate both the development effort required to construct such heuristics and the solution quality that can be achieved using the proposed framework. This evaluation is based on two case studies derived from research projects conducted by the AIT Austrian Institute of Technology.

Our results demonstrate that heuristic optimization can produce highly competitive solutions, outperforming the exclusive MILP-based methods in IESopt on most of the investigated instances. We therefore argue that the development of heuristic methods should at least be considered for more complex research projects.

Kurzfassung

Während Energiesysteme in der Praxis eine Vielzahl an unterschiedlichen Optimierungsproblemen aufweisen können, bestehen sie grundsätzlich häufig aus ähnlichen Komponenten. Diese Beobachtung motivierte die Abteilung Integrated Energy Systems des AIT Austrian Institute of Technology zur Entwicklung des Modellierungsframeworks IESopt, das die Modellierung solcher Systeme vereinfachen soll. Dieser Ansatz hat jedoch auch einige Einschränkungen. Da IESopt letztlich auf einen mathematischen Optimierungssolver zurückgreift, um das erstellte Modell zu optimieren, kann der erforderliche Rechenaufwand für die Ermittlung hochwertiger Lösungen mit zunehmender Modellkomplexität erheblich ansteigen.

Heuristische Optimierung wird daher in dieser Arbeit als Möglichkeit untersucht, diese Einschränkungen zu umgehen und auch für sehr komplexe Instanzen innerhalb angemessener Rechenzeiten hochwertige Lösungen zu erzeugen. Dazu schlagen wir ein heuristisches Framework vor, das sich an der Philosophie von IESopt orientiert und Entwickler bei der Erstellung heuristischer Methoden für spezifische Optimierungsprobleme unterstützen soll. Darüber hinaus untersuchen wir sowohl den Entwicklungsaufwand für die Erstellung solcher Heuristiken als auch die mit dem vorgeschlagenen Framework erreichbare Lösungsqualität. Die Evaluierung basiert auf zwei Fallstudien aus Forschungsprojekten des AIT Austrian Institute of Technology.

Unsere Ergebnisse zeigen, dass die heuristische Optimierung sehr konkurrenzfähige Lösungen erzeugen kann und bei den meisten untersuchten Instanzen die alleinige Verwendung von IESopt übertrifft. Wir kommen daher zu dem Schluss, dass die Entwicklung heuristischer Methoden zumindest für komplexere Forschungsprojekte in Betracht gezogen werden sollte.

Contents

CHAPTER 1

Introduction

The term *energy system* is broad and may refer to a small-scale household comprising, for example, a rooftop photovoltaic (PV) system and an electric vehicle, but it may equally describe a national scale electricity grid composed of multiple power plants and urban centers. In general, an energy system can be defined as a collection of infrastructure and technologies used to produce, convert, store, distribute, and consume energy. Here, energy may refer to different forms of energy or energy carriers. An *energy carrier* in this context is a physical medium that enables energy to be stored, transported, and delivered for subsequent use, such as electricity, heat, biogas, hydrogen, or fuels. The term *Integrated Energy System* (IES) refers to an energy system that integrates multiple energy carriers and their associated technologies. For example, a household equipped with a hot water boiler that draws electricity from the grid and converts it into heat to satisfy a hot water demand.

The Integrated Energy Systems unit at the AIT Austrian Institute of Technology specializes in the modeling, analysis, and optimization of such IES across a variety of research projects. Since these systems typically comprise similar components, the unit developed the IESopt framework [SM24] (short for Integrated Energy System Optimization) to avoid repeatedly defining commonly occurring components, such as energy producers, consumers, conversion technologies, and the connections between them. A more detailed introduction to the IESopt framework is provided in Chapter 3. In general, IESopt can be viewed as a modular modeling toolkit composed of generic components that can be assembled to represent a wide variety of energy systems. The framework is primarily targeted at researchers with limited experience in optimization and software engineering, enabling them to model and analyze such systems without requiring extensive background knowledge. Similar to many other frameworks, as discussed in Section 3.3, IESopt translates the defined model into either a Linear Program (LP) or a Mixed-Integer Linear Program (MILP). While an LP consists exclusively of continuous decision variables and can therefore generally be solved efficiently in practice using methods such as the simplex

method [Dre57] or the interior point method [Kar84], MILPs also include discrete decision variables, which make the solving process in general NP-hard [GJ79].

This approach therefore performs well for smaller applications, such as a single household with only a few components, but can encounter scalability challenges when applied to large-scale or highly detailed energy system models. A representative example is the planning and construction of a new power plant in Austria. Projects of this type may involve long-term planning horizons and require substantial spatial and temporal detail. Consequently, even seemingly simple components can result in a substantial number of decision variables and constraints when scaled across many households, locations, and time steps. The resulting MILP can therefore become computationally demanding to solve, limiting the practical scalability of the modeling approach.

In such cases, heuristic optimization, as outlined in Chapter 2, could provide a promising alternative. This is particularly relevant because the considered problems often rely on estimates of future conditions, such as weather forecasts or day-ahead electricity price predictions. Consequently, obtaining a provably optimal solution may be less important than obtaining a high-quality solution efficiently. This is especially relevant in applications that require repeated reoptimization, such as potential commercial applications, or when evaluating a large number of scenarios. Despite these potential benefits, the AIT Austrian Institute of Technology has so far not incorporated heuristic optimization into its widely used IESopt framework.

The main motivation for this thesis is addressing this gap. The objective is to answer two fundamental questions that arise when considering whether the additional effort required to develop heuristic optimization strategies is justified alongside an established IESopt model. Namely, what solution quality can be expected when applying heuristic optimization to IES models, and what computational and development effort is required to obtain such solutions? Answering these questions is intended to provide practical guidance for assessing whether, and to what extent, project resources should be allocated to the development of heuristic optimization methods.

To answer these questions, we first propose, in Chapter 4, a heuristic framework designed to support the development of heuristic optimization methods for problems for which an IESopt model or a similar mathematical model is already available. The main advantage of leveraging an existing model is that parts of the solution evaluation and feasibility checking can be delegated to the model, while the heuristic focuses on more complex tasks, such as constructing values for discrete decision variables. The framework consists of a set of general operations that must be implemented for each problem individually. On top of these problem-specific operations, it provides several metaheuristic and hybrid optimization approaches that are problem-independent and can be applied to any problem for which the required operations have been implemented. This allows developers to employ more sophisticated and potentially more effective, including even nonlinear, metaheuristics without having to implement and adapt each method separately for every individual problem, thereby reducing development effort.

To meaningfully evaluate the proposed framework and assess the performance gains that can be achieved through heuristic optimization in the context of IES optimization, we conduct two case studies in Chapters 5 and 6. Both case studies originate from research projects conducted by the Integrated Energy Systems unit at the AIT Austrian Institute of Technology. Chapter 5 presents the HoWaFlex problem, which considers the heating and domestic hot water demands of multiple households within an energy community sharing a PV system. The objective is to determine when each household's boiler should be activated in order to make efficient use of the available PV generation while minimizing the overall energy costs.

The second case study, presented in Chapter 6, considers the operation of a battery energy storage system (BESS) for a single household. The objective is again to minimize the total energy costs, but the optimization decisions concern when to charge and discharge the battery in order to satisfy the household's electricity demand. As in the HoWaFlex case, a PV system provides a source of low-cost and renewable electricity that can be used to reduce the household's reliance on grid electricity.

The contributions of this thesis can be summarized as follows:

- Development and implementation of a hybrid heuristic framework that builds upon the established IESopt framework to enable the efficient generation of high-quality solutions while leveraging existing mathematical optimization models.
- Evaluation of the proposed framework and the potential benefits of heuristic optimization in the context of two representative IES case studies.

The remainder of this thesis is organized as follows. Chapter 2 introduces the fundamentals of heuristic optimization and provides the necessary background for readers unfamiliar with the subject to follow the discussion and analysis throughout the thesis. Chapter 3 presents the current modeling and optimization practices employed by the IES unit at the AIT Austrian Institute of Technology and provides a more detailed introduction to the IESopt framework [SM24]. Chapter 4 introduces the proposed heuristic framework. It describes the general operations that must be implemented for each problem individually and introduces the hybrid heuristic approaches provided by the framework, including their implementation. Chapter 5 presents the HoWaFlex case study, while Chapter 6 addresses the BESS case study. Both case studies represent practical applications of the proposed framework and are used to evaluate its performance and the potential benefits of heuristic optimization. Finally, Chapter 7 summarizes the main findings of the thesis, provides concluding remarks, and outlines future work.

CHAPTER 2

Heuristic Preliminaries

This chapter introduces the fundamental concepts and terminology of heuristic optimization required to follow the methods and discussions throughout this thesis.

In general, optimization problems are characterized by a set of *decision variables* that describe the choices available within the model. A specific assignment of values to these variables constitutes a *solution*. Furthermore, optimization problems are typically subject to constraints that restrict the admissible values of these decision variables. Solutions satisfying all constraints are referred to as *feasible*, whereas those violating one or more constraints are considered *infeasible*. The set of all feasible solutions forms the *solution space*, within which the objective is to identify the solution that optimizes a given *objective function*. Such a solution is referred to as the *global optimum*.

The term *heuristic* refers to a problem-solving method that aims to compute high-quality solutions to a given optimization problem within a reasonable amount of time, without guaranteeing global optimality. Consequently, heuristic optimization is typically employed when exact solution methods become computationally infeasible or are unable to produce sufficiently good solutions within the available time.

Heuristics are typically classified into *construction heuristics* and *improvement heuristics*. Construction heuristics generate a feasible solution from scratch by successively assigning values to the decision variables until a complete solution has been constructed. This process is often performed in a greedy manner, where each decision is selected according to a problem-specific criterion that appears most promising at the current stage of the construction process. Since these decisions are made without considering their long-term consequences, they do not necessarily lead to a globally optimal solution. This characteristic gives rise to the term *greedy heuristic* or *greedy algorithm*.

Improvement heuristics, in contrast, start from an existing feasible solution and iteratively explore its so-called *neighborhood* in an attempt to improve the objective value [GP19]. A neighborhood consists of all solutions that can be reached by applying one or more

predefined modifications, commonly referred to as *moves*. The exact definition of a neighborhood and its associated moves is problem-specific and determines which solutions are considered adjacent in the solution space. At each iteration, a neighboring solution is selected according to a predefined search strategy. Common strategies include *best-improvement*, in which the entire neighborhood is explored before selecting the best candidate, and *first-improvement*, in which the search terminates as soon as an improving neighbor is found. Since these methods repeatedly explore a local region of the solution space, they are collectively referred to as *local search* methods. A solution is called a *local optimum* if no neighboring solution provides a better objective value.

Following the previous definitions, we can now introduce the concept of *metaheuristics*. Gendreau and Potvin [GP19] describe metaheuristics as solution methods that orchestrate the interaction between local improvement procedures and higher-level strategies in order to create a search process capable of escaping local optima and performing a robust exploration of the solution space. Unlike problem-specific heuristics, metaheuristics are generally designed as problem-independent frameworks that can be adapted to different optimization problems by defining suitable solution representations, objective functions, and problem-specific operators. Metaheuristics can therefore be viewed as general search strategies that combine different mechanisms to balance exploration of new regions of the solution space with exploitation of promising solutions [GP19].

Furthermore, *hybrid metaheuristics* combine metaheuristic techniques with other optimization paradigms, such as mathematical programming, dynamic programming, or constraint programming. Following the definition of Blum and Raidl [BR16], these approaches exploit the complementary strengths of different optimization methods by integrating heuristic and exact solution techniques. This concept is particularly relevant to this thesis, as the framework proposed in Chapter 4 extends the IESopt modeling framework, which relies on mathematical optimization, by incorporating heuristic methods to improve the solution process for computationally challenging problems.

2.1 Overview of Selected Metaheuristics

While the works of Gendreau and Potvin [GP19] and Blum and Raidl [BR16] provide comprehensive descriptions of a wide range of metaheuristic algorithms that could be applied to integrated energy system optimization, the following sections focus specifically on the metaheuristics that will be presented as part of the framework in Chapter 4. A detailed description of these hybrid metaheuristics and their implementation within the framework is provided in Section 4.2. The purpose of this section is instead to provide the reader with a brief introduction to the employed methods, ensuring familiarity when they are referenced throughout the thesis.

2.1.1 Large Neighborhood Search

The concepts underlying the Large Neighborhood Search (LNS) metaheuristic were first introduced by Shaw [Sha98]. In contrast to conventional local search methods, which typically explore a predefined neighborhood consisting of small modifications to the current solution, LNS aims to explore larger and more complex neighborhoods. This is most often achieved by partially destroying the current solution and subsequently reconstructing the removed components using a repair procedure. The destroy operation typically involves the random or heuristic-guided selection and removal of solution components. The resulting partial solution is then completed by the repair procedure, which commonly employs a construction heuristic or a MILP solver. By allowing larger parts of the solution to be reconsidered simultaneously, LNS is able to escape local optima more effectively than traditional local search approaches.

Although LNS is most commonly associated with applications in routing and scheduling problems, as summarized by Pisinger and Ropke [PR19], it has also been applied to various problems in the energy sector. For example, Yang et al. [YLJ25] proposed an approach for solving the Unit Commitment Problem (UCP), a fundamental optimization problem in power system operation, by combining graph neural networks with an LNS procedure. Similarly, Aliyan et al. [AAL+26] employed a variant of LNS, namely Adaptive Large Neighborhood Search (ALNS), to efficiently cluster prosumers into energy communities based on their electricity demand and photovoltaic generation profiles.

2.1.2 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure (GRASP) was first introduced by Feo and Resende [FR89] in 1989. It combines a greedy randomized construction heuristic with a local search procedure and uses controlled restarts to explore different regions of the solution space. During each GRASP iteration, an initial solution is first constructed and subsequently refined using the local search algorithm. Once the local search terminates, for example because a local optimum has been reached or another predefined stopping criterion is satisfied, a new initial solution is generated and the process is repeated.

To ensure that successive local searches begin from sufficiently different starting points, the greedy construction procedure is randomized. In this thesis, this is achieved through the use of *restricted candidate lists* (RCLs). Rather than always selecting the locally best candidate at a given stage of the algorithm, an RCL containing the most promising candidates according to a problem-specific greedy criterion is first constructed. One candidate is then selected uniformly at random from this list. This introduces controlled randomness into the construction process while preserving the overall greedy nature of the heuristic. The construction of the RCL is implementation-specific and may, for example, be based on a fixed number k of candidates or on a threshold relative to the priority of the best candidate, with all candidates exceeding this threshold included in the RCL. The latter approach ensures that all candidates in the RCL are sufficiently

competitive with the greedy choice. If the entire decision space should remain eligible for selection, alternative strategies such as weighted random sampling or tournament selection may also be employed. For a broader overview of construction randomization techniques, we refer to the surveys by Grasas et al. [GJF+17] and Ferone et al. [FGFJ19].

GRASP has been successfully applied to a variety of optimization problems within the energy sector. For example, Moya et al. [MdSLAVB15] employed GRASP to schedule flexible household loads in a smart home environment with the objective of minimizing electricity procurement costs. Another application was presented by Serrano et al. [SRL23], who combined GRASP with Tabu Search to optimize the operation of electrical distribution networks.

2.1.3 Construct Merge Solve & Adapt

First introduced by Blum et al. [BPLIL16], the Construct, Merge, Solve & Adapt (CMSA) procedure is a hybrid metaheuristic that combines randomized greedy construction heuristics with mathematical programming. The central idea is to iteratively identify a substantially reduced sub-instance of the original optimization problem that is likely to contain high-quality solutions while remaining sufficiently small to be solved efficiently by an exact optimization method, such as a MILP solver.

To this end, CMSA repeatedly generates multiple independent solutions using a randomized greedy construction heuristic. The solution components contained in these solutions are merged into a common solution component pool, which defines the reduced sub-instance. Rather than considering the complete search space, the mathematical optimization model is restricted to selecting only those solution components contained in the current pool, thereby significantly reducing the size of the optimization problem.

After solving the restricted optimization problem, the solution component pool is updated using an aging mechanism. Components that are not selected in the optimal solution of the restricted problem gradually increase in age and are eventually removed from the pool, while newly generated greedy solutions introduce additional components. This iterative process of constructing, merging, solving, and adapting is repeated until a predefined stopping criterion is satisfied.

While mathematical programming is widely employed for energy system optimization, as discussed in Section 3.3, we did not identify any work applying the CMSA procedure to integrated energy system optimization. This is somewhat surprising, as CMSA has been successfully applied to a variety of large-scale combinatorial optimization problems. Notable examples include the work of Akbay et al. [AKB23], who employed CMSA to solve the Electric Vehicle Routing Problem with Time Windows, Simultaneous Pickup and Delivery, and Partial Vehicle Charging, as well as the work of Rosati et al. [RKB+23], who applied CMSA to the Bus Driver Scheduling Problem with complex break constraints.

CHAPTER 3

State of the Art

In this chapter, we present how the Integrated Energy Systems unit at the AIT Austrian Institute of Technology models and optimizes the integrated energy systems arising from its research projects. In particular, Section 3.1 introduces the IESopt framework, a modeling framework designed to simplify the development and optimization of integrated energy systems for researchers. Section 3.2 discusses the practical advantages and limitations of the framework, while Section 3.3 presents alternative frameworks that serve a similar purpose.

3.1 IESopt – Integrated Energy System Optimization

As discussed in the introduction of this thesis, modeling an energy system from the ground up can be a tedious task, as each application comes with its own specifications and unique challenges. Nevertheless, the underlying structure of many energy system models is remarkably similar, with fundamental components recurring across different applications. In general, an energy system consists of energy sources, also referred to as producers, and energy sinks, also referred to as consumers. Producers and consumers are typically connected by appropriate infrastructure that enables the flow of energy between them. This observation motivated Strömer and Maggauer [SM24] to develop the Integrated Energy System Optimization (IESopt) framework, a modular framework for high-performance energy system optimization that allows developers to focus on problem-specific modeling aspects rather than repeatedly implementing common components.

The central idea behind IESopt is to provide a no-code approach to energy system modeling, enabling researchers with little or no expertise in optimization or software engineering to develop energy system models and analyze their results effectively. This is achieved through the use of general components that serve as abstract building blocks for the various elements of an energy system model. Their specific use and interactions are defined in a configuration file written in YAML, which can additionally reference

external data sources, such as CSV files, to facilitate data management. For optimization, IESopt relies on an implementation in the Julia programming language [BEKS17] using the JuMP modeling language [LDDG+23]. The framework translates the components and their connections specified in the configuration file into the decision variables and constraints of a JuMP optimization model. The resulting model is subsequently solved using a mathematical optimization solver, such as the open-source solver HiGHS [HH18] or commercial solvers including Gurobi [Gur26] and CPLEX [IBM26].

3.1.1 Modeling in IESopt

IESopt distinguishes between five so-called *Core Components*, which serve as the fundamental building blocks for modeling a wide range of energy systems. During the optimization process, the framework translates these component into corresponding mathematical formulations.

Throughout the following definitions, the term *carrier* refers to an *energy carrier*, i.e., a form in which energy is stored or transported, such as electricity, heat, or hydrogen. An IESopt model may contain multiple energy carriers simultaneously. For example, a residential energy system may use electricity to satisfy a heating demand. In such a model, electricity is supplied through an electrical grid and subsequently converted into heat by a heating device, thereby coupling the electricity and heat carriers.

Another important concept is that of a *snapshot*, which represents a specific point in time for which all time-dependent quantities are known. Typical examples include the available PV generation, the household electricity demand, or the heating demand. Since energy systems evolve continuously over time, their behavior must be discretized to enable meaningful mathematical optimization. This discretization is typically performed using fixed time intervals, resulting in a finite set of snapshots. The set of all snapshots is denoted by $\mathcal{T}$, with each individual snapshot represented by $t \in \mathcal{T}$. Throughout this thesis, the terms *snapshot* and *timestep* are used interchangeably. While both refer to a point in time within the discretized planning horizon, the latter more explicitly emphasizes the temporal dependency of the optimization model.

IESopt defines to following five Core Components:

`Unit` The abstract concept of energy conversion, i.e., transforming one carrier into another, can be extended to represent a variety of technologies and functionalities. For example, as outlined in the original paper [SM24], an electrolyzer can be viewed as converting electricity and water into hydrogen, heat, and oxygen. An even simpler example is a coal-fired power plant, which converts coal into heat and electricity. A `Unit` can be used to model exactly such conversion processes. Furthermore, it allows for a variety of configurations, including availability, installed capacity, operating modes, costs, and varying efficiencies.

`Node` In its basic configuration, a `Node` serves as a connection point for one or more other components and is associated with a specific carrier. For all connected components,

it encapsulates the *nodal balance equations*, which ensure that, for each snapshot, the total supply (feed-in) at the `Node` is equal to the total demand (withdrawal). However, a `Node` can also be configured to represent more complex behavior by becoming stateful. In this case, inter-temporal constraints are introduced that link consecutive states, allowing the nodal balance to shift energy between snapshots. This can be interpreted as an abstract representation of energy storage.

`Connection` Represents an energy exchange between two `Node` components. It can be configured to be bidirectional, incorporate transmission losses, and may be restricted by fixed or variable bounds.

`Profile` So far, none of the previously introduced components is capable of creating or destroying energy within the model. Instead, energy can only be converted by a `Unit` or transported between two `Node` components using a `Connection`. In its simplest configuration, a `Profile` component injects a predefined amount of energy into, or withdraws it from a `Node` component it is connected to, at each snapshot. This enables the modeling of external energy sources, such as natural gas imports or photovoltaic generation, as well as energy demands, including the heating demand of a household or the electricity consumption of a factory. In practice, `Profiles` are typically provided as external time-series data, for example in CSV files, rather than being specified directly in the YAML configuration.

`Decision` All other Core Components operate primarily based on a user-defined configuration and are limited in their interaction with each other. To enable the modeling of optimal designs for future energy systems and to incorporate investment decisions, the `Decision` component is introduced. It features a variety of bound and cost constraints and can be used in place of many parameters of the other Core Components.

Using these five building blocks, it is possible to model a wide variety of energy systems. In addition, IESopt supports so-called *Templates*, which combine multiple Core Components into reusable, higher-level components that can subsequently be used like regular components within the configuration file. For problem formulations requiring additional constraints or custom functionality, IESopt also provides an extension mechanism through Julia-based add-ons. These allow users to directly access the underlying JuMP model and modify its decision variables and constraints both during and after the model construction process.

3.2 Limitations of IESopt

There is no doubt that IESopt enables the rapid and efficient creation of energy system models and their subsequent optimization. However, since the modeling process ultimately translates the specified system into either a LP or a MILP, the computational performance of IESopt is inherently determined by the performance of the underlying optimization

solver. While LPs can typically be solved very efficiently, the computational complexity and time required to obtain an optimal solution to a MILP may vary significantly depending on the problem instance. This can be particularly problematic in applications where operating conditions change frequently and the optimization problem must be solved repeatedly, or in simulation-based analyses such as the Monte Carlo method [MRR+53], where even a moderate increase of solve time of a single optimization can lead to a substantial increase in the overall computational effort.

It should therefore come with no surprise that the choice of optimization solver has a significant influence on the computational efficiency of IESopt. However, this choice is not always as straightforward in practical applications as it is in an academic setting. While most commercial solvers provide free or cheaper academic licenses for students and researchers, industrial environments, such as those at the AIT Austrian Institute of Technology, are often subject to budgetary constraints or licensing restrictions that may preclude the use of state-of-the-art commercial optimization software. Consequently, our evaluation in the following chapters presents results obtained not only with the commercial solver Gurobi [Gur26], but also with the open-source solver HiGHS [HH18]. This comparison provides insight into the performance of the proposed framework under both commercially licensed and freely available optimization software.

Nevertheless, even state-of-the-art optimization solvers may reach their computational limits, as solving MILPs remains NP-hard in general. Consequently, it is often necessary to trade solution quality for computational efficiency in order to obtain high-quality solutions within acceptable time limits or, for particularly challenging instances, to obtain feasible solutions at all.

One possible approach to addressing these challenges within the IESopt framework is to simplify the underlying optimization model. Since energy systems are typically optimized over a fixed planning horizon, the temporal resolution is often one of the primary factors determining the overall problem size, as it directly affects the number of snapshots that must be considered. Increasing the duration of individual timesteps therefore reduces the number of optimization variables and constraints, which can substantially decrease the computational effort required to solve the model.

In practice, however, reducing the temporal resolution is often not feasible. For example, increasing the timestep length from 15 min to 60 min is unsuitable when variable electricity tariffs are considered, as the Austrian day-ahead market provides electricity prices at 15 min intervals. A coarser temporal resolution would therefore fail to capture these price fluctuations accurately. A related approach is the use of representative days or weeks, in which only a subset of the planning horizon is optimized and the resulting insights are extrapolated to longer periods, such as months or entire years.

Alternatively, the mathematical formulation itself can be simplified by removing less critical modeling details or by reformulating certain constraints and decision variables to obtain a more computationally tractable model. While such simplifications can significantly improve computational performance, they inevitably reduce the fidelity of

the optimization model. Excessive abstraction may therefore compromise the validity of the obtained results and ultimately undermine the original optimization objective.

A fundamentally different approach is the use of heuristic optimization methods. Rather than simplifying the mathematical model, heuristics aim to efficiently compute high-quality, although generally non-optimal, solutions to the considered computationally challenging optimization problems. Such methods have long been established for solving NP-hard problems and are widely used whenever exact optimization becomes computationally prohibitive. Despite their success in many application domains, heuristic optimization methods have so far received only limited attention in the context of IESopt.

3.3 Alternative Energy System Modeling Frameworks

To further motivate the use of heuristic optimization methods in the context of IESopt, we examine the capabilities of other energy system modeling frameworks and investigate whether the limitations discussed in Section 3.2 also apply to them. Unfortunately, these limitations are largely a consequence of the underlying optimization paradigm and are therefore not specific to IESopt. Most modern energy system modeling frameworks rely on a similar approach. The energy system is represented using generic components, which are translated into a mathematical optimization formulation and subsequently solved using established optimization algorithms. Consequently, while these frameworks greatly simplify the modeling process, their computational scalability remains fundamentally tied to the capabilities of the underlying optimization solver.

IESopt is therefore not the only open-source framework developed for the modeling and optimization of energy systems. Numerous alternative frameworks have been proposed, including *EnergyModelsX* by Hellemo et al. [HBH+24], *MacroEnergy.jl* by Macdonald et al. [MPB+25], and *PowNet 2.0* by Bunnak et al. [BEP+25]. Despite differences in implementation language and application focus, these frameworks all follow the same underlying principle as IESopt. Energy systems are represented using predefined components and models, which are subsequently translated into mathematical optimization formulations and solved using established optimization solvers. In particular, EnergyModelsX and MacroEnergy.jl, similar to IESopt, employ a component-based modeling approach built on JuMP, while PowNet 2.0 is implemented in Python and focuses specifically on power grid operation. Nevertheless, all three frameworks ultimately rely on conventional mathematical optimization methods, and therefore share the same advantages and limitations with IESopt.

A slightly older comparison of open-source energy system modeling frameworks is provided by Candas et al. [CMB+22], who evaluate five more frameworks with respect to their modeling capabilities, component complexity, and intended application domains. Note for the purposes of this thesis, all five of the considered frameworks also predominantly rely on mathematical optimization formulations as the primary solution approach.

In summary, existing energy system modeling frameworks provide powerful abstractions

for formulating and solving complex optimization problems. However, they generally rely on exact mathematical optimization methods and therefore inherit the computational challenges associated with large-scale LP and MILP formulations. To the best of our knowledge, none of the reviewed frameworks incorporates heuristic optimization as an integrated solution strategy. This observation further motivates the development of the heuristic extension proposed in this thesis.

Note that the focus on IESopt in this thesis is motivated by the authors' previous experience with the framework and its relevance to the Integrated Energy Systems unit at the AIT Austrian Institute of Technology. However, due to the shared underlying principles of the frameworks discussed in this section, the concepts and considerations developed in this thesis are not limited to IESopt and can also be applied to other frameworks following a similar modeling and optimization approach.

CHAPTER 4

Heuristic Framework for IESopt

Motivated by the lack of heuristic optimization methods that integrate with the IESopt framework to overcome the limitations of conventional mathematical optimization solvers, we propose a generic heuristic framework that complements the existing IESopt framework while following its underlying design principles. The proposed framework is intended to assist researchers in the field of integrated energy system optimization by enabling them not only to model their optimization problems efficiently but also to obtain high-quality solutions, even for computationally challenging instances. At the same time, it is designed to remain intuitive to use without requiring extensive expertise in heuristic optimization.

Initially, our primary objective was to preserve the fundamental no-code philosophy of IESopt as far as possible. We therefore investigated whether a heuristic algorithm could be constructed solely from the information contained in the IESopt configuration file. However, this quickly revealed a fundamental trade-off between generalizability and problem specificity. Since every element of an energy system is ultimately represented by one or more Core Components in the IESopt model, knowledge of the specific component type, like `Unit` or `Node`, provides only limited structural information that can be exploited by a heuristic algorithm. Consequently, a heuristic operating primarily on the translated decision variables and constraints would, in many respects, differ only marginally from the preprocessing and built-in heuristic mechanisms already employed by modern MILP solvers.

The next idea was to enrich the components defined in the configuration file with additional tags that convey domain-specific information to the heuristic algorithm. For example, a component could be identified as a hot water boiler, indicating that it continuously loses heat to its surroundings. Based on this information, the heuristic could infer that maintaining a lower storage level may be beneficial to reduce thermal losses. Similarly, decision variables could be associated with `Profile` components, enabling rules such as charging a storage system whenever renewable generation is abundant or electricity

prices are low. In this way, the heuristic could derive its decisions from a collection of problem-specific rules.

However, this approach proved to be less straightforward than it initially appeared. As discussed in Section 3.1.1, IESopt allows multiple Core Components to be combined into reusable Templates, which define new, more complex components. These user-defined components would likely require their own tags and corresponding heuristic rules. Consequently, the extensibility of such an approach would be limited, as each newly introduced component type could require additional heuristic logic to be implemented. The situation is further complicated by the use of add-ons, which can directly modify the underlying JuMP model without their functionality being explicitly represented in the configuration file. As a result, every new add-on could also necessitate the development of corresponding tags and heuristic rules, further increasing the complexity and maintenance effort of the framework.

4.1 A Modular Approach based on the Strategy Pattern

After preliminary experiments with the approaches discussed in the introduction of this chapter, we ultimately adopted a more straightforward and practical solution. The Strategy Pattern, first introduced by Gamma et al. [GHJV95], is a well-established design pattern in object-oriented software engineering that encapsulates algorithms or functionality into interchangeable strategies. These strategies share a common interface and can therefore be selected or replaced at runtime without affecting the surrounding application.

In the proposed framework, this concept is applied by identifying the problem-specific building blocks required by more complex heuristic algorithms and encapsulating them as individual strategies. Examples of such components include construction heuristics, neighborhood functions for local search procedures, or problem-specific objective evaluations. This separation enables the implementation of metaheuristics that depend only on these abstract strategy interfaces and are therefore independent of any particular energy system model. Consequently, the same metaheuristic implementation can be reused across different optimization problems without modification.

Conversely, when a new energy system optimization problem is introduced, it is sufficient to implement the required strategy components in order to immediately leverage the complete library of existing metaheuristics. This modular design promotes code reuse, simplifies the implementation of new optimization problems, and facilitates the extension of the framework with additional heuristic methods.

4.1.1 Generic Structures

Due to the high computational performance of the Julia programming language [BEKS17], as well as the implementation of IESopt in Julia and the resulting seamless compatibility between both frameworks, Julia was selected as the implementation language for the proposed framework. However, because Julia is not a class-based object-oriented language

and instead relies on multiple dispatch, the Strategy Pattern, originally developed in the context of object-oriented software engineering, cannot be adopted directly in its classical form. Consequently, its underlying principles must be adapted to Julia's programming paradigm by employing generic structures and multiple dispatch instead of classes and inheritance.

To enable the definition of metaheuristics that are independent of specific problem formulations, we introduce abstract placeholder structures, which can later be replaced by concrete problem-specific implementations. We define the following key concepts:

`GenericInstance` This structure serves as a placeholder for a problem instance. A problem instance is defined as a concrete specification of a problem, typically storing all required parameters, constants and input data. An instance is considered immutable throughout the optimization process, as modifying its contents would result in a different problem instance.

`GenericSolution` This structure serves as a placeholder for a solution of a given problem instance. It stores the solution representation as well as all information required to evaluate and manipulate the solution. In contrast to the problem instance, which represents the static component of the optimization process, the solution represents the decision variables and may therefore be modified throughout the optimization procedure.

Since some metaheuristics require access to individual parts of a solution, we define a *solution component* as an atomic unit of the solution representation. Such a component typically corresponds to a single decision made by the optimization algorithm, for example, the activation of a hot water boiler at a specific timestep. Because solution components are immutable and usually consist of an identifying key together with only a small number of associated decision variables, no dedicated data type is introduced for them. Instead, the native Julia `NamedTuple` type is used to represent individual solution components. This also makes it easy to define different kinds of solution components whenever a problem requires them.

`GenericConfiguration` While the previous two structures represent the static problem definition and the dynamic solution representation, respectively, the configuration stores the fixed parameters controlling the optimization process itself. These parameters include problem-specific hyperparameters used during the generation, evaluation, or improvement of solutions, but are independent of a specific problem instance. Consequently, a single configuration can be applied to multiple instances of the same problem class. However, different metaheuristics may require different configurations for the underlying problem-specific operations.

`GenericState` To support optimization methods that require access to information about the current stage of the optimization process, we introduce a state representation. The state stores dynamic meta-information generated during an optimization

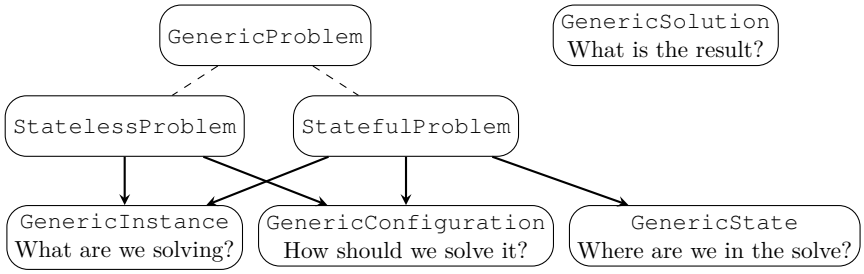


Figure 4.1: Overview of the generic structures in the heuristic framework and their relationships. Dashed lines indicate *is-a* relationships, while arrows indicate that the connected structures are contained within the respective problem structure.

run, such as the current iteration of the algorithm or intermediate data required by specific methods. It also enables the reuse of problem-specific operations across different phases of the optimization process. For example, the same construction procedure may be used both during initial greedy solution generation and in later refinement phases, while behaving slightly differently depending on the current optimization state.

`GenericProblem` We define a problem as the composition of a `GenericInstance`, a `GenericConfiguration`, and, optionally, a `GenericState`. Depending on whether a state representation is required for the implementation, we distinguish between a `StatelessProblem` and a `StatefulProblem`. Since all sources of randomness, such as for example the used random seed, are expected to be controlled through the included configuration, solving a given problem instance with a fixed configuration therefore deterministically produces a corresponding `GenericSolution`. Figure 4.1 presents an overview of all included structures.

In this context, it is important to note that the representation of an energy system model within the framework may differ from its corresponding formulation in IESopt. While the IESopt formulation contains all information required to define and solve the exact optimization problem, the heuristic implementation may require additional problem-specific variables and auxiliary information to efficiently construct and evaluate solutions. Conversely, information present in the original formulation may be omitted if it is not required by the heuristic or would introduce unnecessary computational overhead. Furthermore, since IESopt fundamentally relies on linear programming, the models it represents are linear. This restriction does not necessarily apply to the heuristic implementation, which may employ nonlinear representations or operations if they facilitate the construction or evaluation of solutions.

4.1.2 Problem-Specific Operations

After defining the generic structures, we can now introduce the operations that must be implemented for a specific problem in order to make use of the proposed framework. The

concrete implementation of each operation for a given problem definition corresponds to a strategy that is subsequently used by the metaheuristics described in Section 4.2. The framework defines the following core operations:

`generate_solution` The first operation defined by the framework is the generation of an initial solution. This typically involves the implementation of a greedy construction heuristic capable of returning a `GenericSolution` for a given `GenericProblem`. As the practical implementations presented as part of the case studies in Chapters 5 and 6 demonstrate, this is generally the most challenging operation to implement. At the same time, it is also the stage at which domain-specific knowledge can be incorporated into the heuristic without restrictions, as there is no requirement to adhere to the structure of the IESopt configuration file. An important aspect of this operation is the incorporation of randomization. Most metaheuristics presented in Section 4.2 require a diverse set of high-quality initial solutions, which can be achieved by introducing controlled randomization into the construction process. The hyperparameters used to control this randomization are typically stored in the respective `GenericConfiguration`.

`next_neighbor` This operation takes a `GenericProblem` and a `GenericSolution` as input and returns a neighboring `GenericSolution`. It is primarily used in algorithms relying on local search procedures, such as the LNS and GRASP metaheuristic introduced in Sections 4.2.1 and 4.2.2 respectively. If required, a `GenericState` can be used to facilitate structured neighborhood exploration.

`solve` This is the final primary operation and represents the framework's interface to IESopt. As discussed previously, the heuristic does not necessarily have to account for every aspect of the optimization problem. Instead, one of the key advantages of using an LP or MILP formulation in the background is that it can be employed to complete and validate a heuristic solution. For example, a practical strategy is to let the heuristic determine the values of the discrete decision variables and subsequently fix these variables within the IESopt model through the `solve` operation. The resulting LP or MILP can then be solved to determine the remaining continuous variables while simultaneously ensuring that all constraints are satisfied. Consequently, this operation primarily consists of mapping a set of solution components to the corresponding variables of the IESopt model before invoking the underlying solver.

`components` This is the first of two translation operations. As discussed previously, some metaheuristics require access not only to complete solutions but also to their individual solution components. For example, the CMSA metaheuristic presented in Section 4.2.3 operates on individual solution components. This operation therefore is intended to return a vector of solution components extracted from a given `GenericSolution`.

`solution` This operation is the counterpart to `components`. It takes a set of solution components as input and reconstructs the corresponding `GenericSolution`.

4.1.3 Using the Underlying MILP for Evaluation

A key advantage of building upon a framework such as IESopt is that the underlying MILP formulation can be used to compute all implied and continuous variables required for a complete solution and to evaluate the exact objective value. Consequently, it is often a viable strategy to reduce the complexity of the `GenericProblem` implementation by allowing the `generate_solution` and `next_neighbor` operations to operate on a reduced search space, while relying on the `solve` operation to reconstruct previously omitted variables and enforce neglected constraints using the underlying IESopt model. This approach is especially demonstrated by the BESS case study presented in Chapter 6.

The primary advantage of such an approach is the resulting simplicity of the implementation. Since the developer neither has to validate the heuristic's output nor account for every constraint present in the IESopt model, implementing such a heuristic becomes significantly easier and faster than developing one that performs complete feasibility checking and objective evaluation independently.

Additionally, computing the exact objective value is often unnecessary in practice, as demonstrated by the HoWaFlex case study in Chapter 5. Typical energy system applications focus on evaluating decisions with respect to their future impact and therefore rely on imperfect forecasts of future conditions. As a result, a good estimation of the objective value often suffices, since many of the parameters used in its computation are themselves estimates, or because the overarching optimization process ultimately depends on the resulting decisions rather than the exact objective value.

However, it should be noted that invoking the solver is typically the most expensive component in terms of total runtime, as illustrated by the case studies presented in Chapters 5 and 6. Consequently, classical local search procedures may become considerably less efficient, since evaluating a move using simplified computations, such as delta evaluation, is typically substantially faster than fixing all variables in the underlying MILP formulation and invoking the solver to evaluate the resulting solution. Depending on the intended use case, it may therefore be preferable to perform feasibility checking and objective evaluation directly within the heuristic. Considerations for implementing such an approach are outlined in Section 4.3.1.

4.2 Library of Meta-Heuristics

In this section, we take a closer look at the metaheuristics currently available within the proposed framework. As outlined in Section 4.1.2, these methods are implemented independently of the underlying optimization problem and operate exclusively on the previously defined problem-specific operations. Furthermore the descriptions presented

Algorithm 4.1: Large Neighborhood Search (LNS) procedure.

Input: A concrete `GenericProblem` p and `GenericSolution` s, time limit T^{lim}, iteration limit I^{lim}
Output: A corresponding `GenericSolution`

Initialize $s^{\text{bsf}} \leftarrow s$ and $z^{\text{bsf}} \leftarrow \texttt{solve}(p, s)$;
Initialize $i \leftarrow 0$;
while T^{lim} is not reached **and** $i < I^{\text{lim}}$ **do**
    $s^{\text{neig}} \leftarrow \texttt{next\_neighbor}(p, s^{\text{bsf}})$;
    Compute $z^{\text{neig}} \leftarrow \texttt{solve}(p, s^{\text{neig}})$;
    if $z^{\text{neig}} < z^{\text{bsf}}$ **then**
        Set $s^{\text{bsf}} \leftarrow s^{\text{neig}}$ and $z^{\text{bsf}} \leftarrow z^{\text{neig}}$;
        Set $i \leftarrow 0$;
    else
        Set $i \leftarrow i + 1$;
    end
end
return s^{bsf};

in this section focus purely on how they are implemented in our framework, for a more general description of the metaheuristics and some related work, we refer to Chapter 2.

It is important to note that the primary objective of this thesis is to establish the foundation of a heuristic framework rather than to provide an exhaustive collection of metaheuristic implementations. Furthermore, not only the expected solution quality is an important measure but also the complexity of the implementation, or rather how complex the required operations are that need to be implemented specifically for a new problem. Consequently, the metaheuristics currently included in the framework were selected primarily based on their suitability for the case studies presented in Chapters 5 and 6 and their inherit complexity. Possible extensions of the framework and additional metaheuristics that could be incorporated in the future are discussed in Section 4.3.

4.2.1 Large Neighborhood Search

The selection of this metaheuristic was largely motivated by the way candidate solutions are evaluated in our case studies. As outlined in Section 4.1.3, using the underlying MILP model for solution evaluation considerably simplifies the implementation, but also limits the efficiency of certain classes of metaheuristics. In particular, this evaluation approach deliberately omits the logic required for efficient delta evaluation, as outlined in Section 4.3.1, and instead relies on repeatedly solving the MILP model for each candidate neighbor. Consequently, ensuring that only neighbors with a high probability of yielding an improved or promising solution are evaluated can substantially reduce unnecessary computational effort. The pseudocode of the implemented LNS algorithm is presented in Algorithm 4.1.

Algorithm 4.2: Greedy Randomized Adaptive Search Procedure (GRASP).

Input: A concrete `GenericProblem` p, local search procedure LS, time limit T^{lim}, iteration limit I^{lim}
Output: A corresponding `GenericSolution`

```
Initialize s^bsf ← ∅ and z^bsf ← +∞;
while T^lim is not reached do
    Generate s ← generate_solution(p);
    Refine s using LS until either T^lim or I^lim is reached;
    Compute z ← solve(p, s);
    if z < z^bsf then
        Set s^bsf ← s and z^bsf ← z;
    end
end
return s^bsf;
```

Furthermore, note that the LNS implementation provided by our framework follows a first-improvement search strategy. More specifically, it repeatedly generates promising neighboring candidate solutions and accepts the first neighbor that improves upon the current solution. This process is repeated until either the time limit T^{lim} is reached or no improving solution has been found for I^{lim} consecutive iterations.

The strategy for generating promising neighboring solutions is problem-specific and must be implemented in the `next_neighbor` operation.

4.2.2 Greedy Randomized Adaptive Search Procedure

The primary motivation for integrating GRASP into the framework was the simplicity of its basic formulation and its clear decomposition into distinct algorithmic components. As discussed in Section 2.1.2, unlike a single local search procedure, GRASP repeatedly restarts the search from newly constructed greedy solutions whenever the current local search converges to a local optimum. This iterative restart mechanism enables the exploration of different regions of the solution space and thereby reduces the risk of becoming trapped in poor local optima. The pseudocode for our implementation of the GRASP algorithm is presented in Algorithm 4.2

In the proposed framework, this corresponds to generating a `GenericSolution` using the `generate_solution` operation, followed by applying a local search procedure to improve the constructed solution. This procedure could, for example, be the LNS presented in Section 4.2.1. The improvement process is repeated until either the overall time limit T^{lim} is reached or a predefined number of consecutive local search iterations fails to improve the incumbent solution. This termination criterion is again denoted by I^{lim}. After terminating the local search, the metaheuristics restarts from another `GenericSolution`.

Algorithm 4.3: Construct Merge Solve & Adapt (CMSA) procedure.

Input: A concrete `GenericProblem` p, maximum age $a^{\max}$, maximum pool size $C^{\lim}$, number of solutions per iteration S^{iter}, time limit $T^{\lim}$
Output: A corresponding `GenericSolution`

Initialize $s^{\text{bsf}} \leftarrow \emptyset$ and $z^{\text{bsf}} \leftarrow +\infty$;
Initialize solution component pool $\mathcal{C} \leftarrow \emptyset$;
while $T^{\lim}$ is not reached **do**
  for $i \leftarrow 1$ **to** S^{iter} **and** $|\mathcal{C}| < C^{\lim}$ **do**
    Generate $s \leftarrow$ `generate_solution`(p);
    for $c \in$ `components`(p, s)**and** $|\mathcal{C}| < C^{\lim}$ **do**
      if $c \notin \mathcal{C}$ **then**
        Add c to $\mathcal{C}$ and set $a_c \leftarrow 1$;
      end
    end
  end
  Compute $s^{\text{opt}}, z^{\text{opt}} \leftarrow$ `solve`$(p, \mathcal{C})$;
  if $z^{\text{opt}} < z^{\text{bsf}}$ **then**
    Set $s^{\text{bsf}} \leftarrow s^{\text{opt}}$ and $z^{\text{bsf}} \leftarrow z^{\text{opt}}$;
  end
  for $c \in \mathcal{C}$ **do**
    if $c \in$ `components`(p, s^{opt}) **then**
      Set $a_c \leftarrow 1$;
    else
      Set $a_c \leftarrow a_c + 1$;
    end
  end
  $\mathcal{C} \leftarrow \{c \in \mathcal{C} \mid a_c \leq a^{\max}\}$
end
return s^{bsf};

In order to meaningfully restart the search from a new and substantially different solution, GRASP requires the greedy construction procedure to be randomized. While many randomization strategies exist, this thesis primarily employs RCLs, as outlined in Section 2.1.2. Instead of selecting the single best decision at each step of the greedy procedure, the k most promising candidates are collected into an RCL, from which the next decision is selected uniformly at random.

4.2.3 Construct Merge Solve & Adapt

Given that a MILP formulation is already available through the IESopt framework, implementing a Construct, Merge, Solve & Adapt (CMSA) metaheuristic is a natural choice. The pseudocode of the implemented CMSA algorithm is presented in Algorithm 4.3.

As outlined in Section 2.1.3, the procedure is fundamentally divided into four distinct phases, which also give the algorithm its name. Similar to the GRASP procedure described in Section 4.2.2, the *Construct* phase relies on the randomized construction procedure implemented in `generate_solution` to generate diverse candidate solutions. The solution components of these candidate solutions are then extracted using `components` and merged into a common pool during the *Merge* phase. This pool represents the aforementioned reduced sub-instance and contains by construction only solution components that have appeared in at least one generated solution, which are therefore expected to be part of high-quality solutions.

In the *Solve* phase, a MILP model is constructed from the current pool of solution components. This is achieved by taking the original MILP formulation and fixing every decision variable, whose associated solution component assumes only a single value within the pool, to that value. All remaining variables are either left unrestricted or restricted to the values represented by the corresponding solution components in the pool. After solving this reduced MILP, the pool is updated in the *Adapt* phase. For this purpose, an age value a_c is maintained for every solution component c. Whenever a solution component is contained in the pool but does not appear in the solution returned by the MILP solver, its age is increased. Components whose age exceeds a predefined threshold are removed from the pool and are therefore eliminated from the reduced search space. This threshold is defined by the hyperparameter $a^{\max}$.

These four phases are repeated iteratively until the overall time limit $T^{\lim}$ is reached. Additionally, to better control the size of the solution component pool during the execution of the algorithm, we introduce two further hyperparameters. First, the number of solutions generated using `generate_solution` during each CMSA iteration is specified by S^{iter}. Second, to directly limit the size of the reduced sub-instance, we introduce the hyperparameter $C^{\lim}$, which specifies the maximum number of solution components that may be stored in the pool simultaneously. This extension is motivated by the need to ensure that the resulting reduced MILP instances remain tractable within the prescribed time limit $T^{\lim}$. This becomes particularly important for large or computationally challenging problem instances, for which even the reduced MILP may otherwise become prohibitively difficult to solve.

Unlike the GRASP implementation in section 4.2.2, CMSA does not rely on the `next_neighbor` operation. Consequently, implementing this metaheuristic within the framework only requires implementations of the `generate_solution`, `solve`, and `components` operations.

4.3 Extending the Framework

All metaheuristics introduced in Section 4.2 are already implemented within the framework and are evaluated as part of the benchmarks presented in Chapters 5 and 6. Consequently, this section focuses on illustrating how classical metaheuristics that are not part of the

proposed first prototype can nevertheless be implemented using the components provided by the framework.

4.3.1 Local Search using Delta Evaluation

In some applications, it may be beneficial to incorporate solution evaluation directly into the heuristic rather than relying on the underlying MILP model to compute the objective value as illustrated in Section 4.1.3. While this approach requires additional development effort, it can significantly improve the overall runtime. This is particularly relevant when the intended application involves repeatedly evaluating similar candidate solutions, as is commonly the case in classical local search procedures.

In local search, *delta evaluation* refers to evaluating the impact of a move operation rather than recomputing the objective value of each newly generated neighboring solution from scratch. Instead of fully evaluating a neighboring candidate solution, the objective value is computed based on the current solution and the modifications introduced by the applied move. Since the required computations depend on the underlying optimization problem, delta evaluation must be implemented individually for each problem. As the metaheuristics provided by the framework invoke the `solve` operation whenever the objective value of a solution is required, the simplest way of integrating delta evaluation is to store the objective value directly within the concrete implementation of `GenericSolution`. Julia's built-in multiple dispatch can then be used to provide a specialized implementation of the `solve` operation that accepts the implementation of `GenericSolution` as input and simply returns the stored objective value, thereby enabeling the seamless integration of delta evaluation into metaheuristics such as LNS and GRASP.

Furthermore, the efficient evaluation of neighboring candidate solutions enables the exploration of substantially larger portions, or even the entirety, of a neighborhood. In this context, a structured strategy for traversing the neighborhood becomes necessary. Within the framework, this can be achieved through an explicit implementation of `GenericState`. Such a state can, for example, maintain the current position within the neighborhood exploration process, thereby allowing the systematic traversal of large neighborhoods without repeatedly generating or evaluating previously considered candidate solutions.

4.3.2 Multiple Neighborhood Functions

Methods such as the Variable Neighborhood Search (VNS), introduced by Mladenović and Hansen [MH97], and Adaptive Large Neighborhood Search (ALNS), introduced by Ropke and Pisinger [RP06], simultaneously rely on multiple neighborhood functions rather than a single neighborhood definition. By employing different neighborhoods, these metaheuristics aim to exploit their combined strengths and achieve a balance between escaping local optima and further improving the current solution.

As with any neighborhood definition, these neighborhood functions must be implemented individually for each specific problem. However, since the framework invokes the `next_neighbor` operation whenever a neighboring candidate solution is requested, support for multiple neighborhoods can be incorporated naturally. In particular, the implementation of `next_neighbor` can dispatch between the available neighborhood functions, while an implementation of `GenericState` manages the current neighborhood selection and therefore determines which neighborhood function is invoked.

4.3.3 Multithreaded Optimization

Lastly, we briefly discuss the topic of parallel optimization. In the case studies presented in Chapters 5 and 6, the implemented metaheuristics were executed using a single thread, while only the underlying MILP solvers were allowed to utilize multiple threads. For more computationally demanding metaheuristics, however, it may be advantageous to parallelize the heuristic itself.

The most suitable parallelization strategy depends on the amount of communication required between individual processes or threads. However, if the processes or threads operate largely independently, a straightforward implementation would be to provide each process or thread with its own instance of `GenericState`, allowing it to maintain its own state, while at the same time, the instances of `GenericInstance` and `GenericConfiguration` can be shared across all processes or threads, as they represent immutable problem data and configuration parameters.

Although parallelization is not implemented in the current version of the framework, the GRASP and CMSA algorithms could be parallelized with relatively little modification. In GRASP, the individual randomized construction and subsequent refinement processes are independent of one another and can therefore be executed in parallel. Similarly, the construction of the component pool in CMSA could be parallelized, although the primary performance bottleneck is typically the solution of the resulting restricted subinstance which can independently be parallelized by increasing the number of threads available ot the MILP solver. For a comprehensive discussion of parallel metaheuristics, we refer to the work of Talbi [Tal09].

CHAPTER 5

Case Study: HoWaFlex2Market

The first case study used for evaluation of the proposed framework originates from a research project called HoWaFlex2Market (or HoWaFlex in short) carried out at AIT Austrian Institute of Technology with partners. The projects acronym HoWaFlex2Market stands for *hot water flexibility to electricity markets*. General information about the project is available on the AIT Austrian Institute of Technology website[1].

While the HoWaFlex project primarily focuses on research and on comparing different approaches under a variety of scenarios, this thesis reuses one specific problem variant investigated within the project. More precisely, the considered optimization problem aims to minimize the total energy costs of an energy community consisting of multiple households that share a photovoltaic (PV) system. A schematic representation of such an energy community is shown in Figure 5.1.

Each household is equipped with an individual hot water boiler that supplies domestic hot water (DHW), space heating, or both. The optimization problem aims to determine the optimal control strategy for the boilers across the entire energy community by allocating the available PV generation among the households while accounting for variable electricity tariffs and household electricity demand profiles over the planning horizon.

5.1 Problem Description

The problem is defined over a fixed planning horizon, which is discretized into individual timesteps $t = 1, \ldots, T$, with the total number of timesteps denoted by T. The duration of each timestep is given by Δ, expressed as a fraction of one hour. For instance, for a timestep length of 15 minutes, $\Delta = \frac{1}{4}$.

Since each household in the community is equipped with its own hot water boiler, a household index $h = 1, \ldots, H$ is introduced to distinguish between them, with H

[1] https://www.ait.ac.at/en/research-topics/flexibility-business-models/projects/howaflex2market

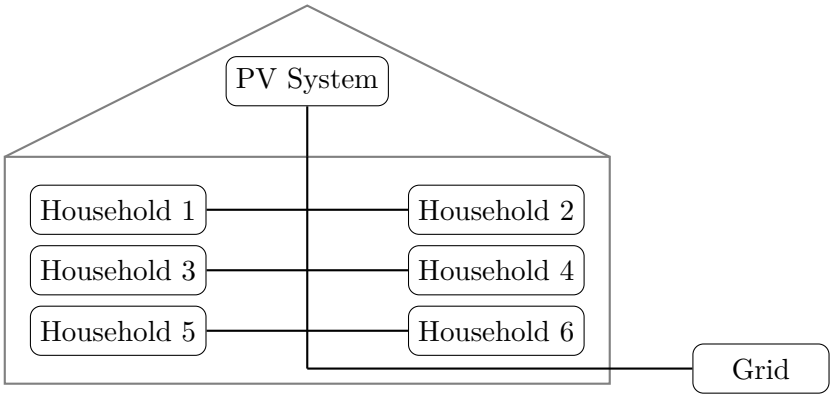


Figure 5.1: Schematic representation of a HoWaFlex energy community.

denoting the total number of households. Each boiler is conceptually divided into a heating component and a storage (tank) component, where the former is responsible for heating the water stored in the latter.

To simplify the modeling, the thermal state of the tank is represented in units of kWh, corresponding to the amount of energy required to heat the stored water from its reference intake temperature. The state of charge of the tank, hereafter referred to as stored energy, is initialized at E_h^{init} and is constrained to remain within lower and upper bounds E_h^{min} and E_h^{max}, respectively.

Heat losses due to imperfect insulation are modeled using a linear approximation consisting of a proportional loss coefficient α_h and a constant loss term β_h for each household h. The unit of β_h is kW, denoted throughout the remainder of this thesis by $[\beta_h] = \text{kW}$. In addition, each household exhibits a time-dependent hot water demand $D_{h,t}$ at every timestep t, where $[D_{h,t}] = \text{kW}$. For modeling purposes, this demand aggregates both domestic hot water consumption and hot water used for space heating into a single demand profile. While these two demands are typically supplied by separate water systems in practice, technologies capable of providing both from a single boiler system also exist. Alternatively, if both demand types are to be included in the optimization while being supplied by separate boilers, the additional boilers can simply be modeled as separate household entities with independent demand profiles.

To control energy injection into the tank, a binary control variable $x_{h,t}$ is introduced, where $x_{h,t} = 1$ indicates that the heating element of household h is active at timestep t, and $x_{h,t} = 0$ indicates that it is inactive. The heater operates at a rated power P_h^{heat}, where $[P_h^{\text{heat}}] = \text{kW}$, and with efficiency η_h, which captures the fraction of electrical power effectively converted into thermal energy stored in the tank. Although heating systems may exhibit transient dynamics, such as ramp-up behavior or switching spikes during activation, these effects are neglected in the present model, as their duration is negligible compared to the typical 15-minute timestep resolution and therefore has an insignificant impact on the overall energy balance.

Under these assumptions, the stored energy $e_{h,t}$ evolves according to

$$e_{h,t+1} = e_{h,t} \cdot (1 - \alpha_h)^{\Delta} - \beta_h \cdot \Delta - D_{h,t} \cdot \Delta + x_{h,t} \cdot P_h^{\text{heat}} \cdot \eta_h \cdot \Delta, \tag{5.1}$$

for all households $h = 1, \ldots, H$ and all timesteps $t = 1, \ldots, T$.

Power supply in this problem is modeled via two sources. First, electricity can be purchased from the grid. The corresponding costs consist of a fixed grid usage fee C^{grid}, where $[C^{\text{grid}}] = €/\text{kWh}$, and a supplier-specific electricity tariff $C_{h,t}^{\text{sup}}$, where $[C_{h,t}^{\text{sup}}] = €/\text{kWh}$, for each household h. Depending on the selected tariff, the supplier price may either be fixed over the planning horizon or vary over time according to the day-ahead electricity market. Second, the community is equipped with a PV system that generates power P_t^{pv} at each timestep t, where $[P_t^{\text{pv}}] = \text{kW}$, bounded between zero and its rated nominal peak capacity $P_{\text{nom}}^{\text{pv}}$. Excess PV generation that is not consumed locally can be exported back to the grid at a feed-in tariff $C_t^{\text{feed-in}}$, where $[C_t^{\text{feed-in}}] = €/\text{kWh}$.

To capture the cost allocation at the level of individual heater operation, the PV and grid power consumed by household h at timestep t are denoted by $p_{h,t}^{\text{pv}}$ and $p_{h,t}^{\text{grid}}$, respectively, where $[p_{h,t}^{\text{pv}}] = \text{kW}$ and $[p_{h,t}^{\text{grid}}] = \text{kW}$. The total cost $c_{h,t}$ associated with operating the heater in timestep t, where $[c_{h,t}] = €/\text{kWh}$, can then be expressed as

$$c_{h,t} = \frac{p_{h,t}^{\text{pv}}}{P_h^{\text{heat}}} \cdot C_t^{\text{feed-in}} + \frac{p_{h,t}^{\text{grid}}}{P_h^{\text{heat}}} \cdot (C^{\text{grid}} + C_{h,t}^{\text{sup}}). \tag{5.2}$$

Finally, several modeling extensions and alternative objective formulations can be considered. The primary objective in this formulation is cost minimization. This implies that PV energy is allocated preferably to households facing higher electricity tariffs. Depending on the application, this behavior may be undesirable. An alternative is to enforce a more equitable distribution of PV energy across households.

Preliminary experiments have explored the inclusion of such fairness constraints. However, they introduced additional feasibility issues. In particular, when PV generation is abundant relative to demand, strict proportional allocation may require assigning energy to households that cannot physically absorb it due to limited consumption capacity. In these experiments, this issue was addressed by introducing non-flexible loads (NFLs), representing baseline consumption from devices such as refrigerators or air-conditioning units. These loads ensure a minimum level of demand across all households and thereby improve the feasibility of equitable energy allocation.

Another open consideration concerns Legionella prevention in domestic hot water systems. Legionella bacteria can proliferate in stagnant hot water systems and pose health risks. Periodic heating to sufficiently high temperatures is commonly assumed to mitigate this risk and could be incorporated into the model as a constraint requiring the tank to reach a specified temperature threshold within a given maximum interval. However, given the technological assumptions in the later case studies, this requirement is treated as

optional. Moreover, the long-term effectiveness of such thermal disinfection remains subject to uncertainty, as Legionella bacteria may persist in pipework or potentially adapt to repeated thermal treatments, as discussed by Molina Gómez et al. [MGBPCR22].

5.1.1 MILP Formulation

Sets of Indices

$\mathcal{H} = \{1, \dots, H\}$	set of households
$\mathcal{T} = \{1, \dots, T\}$	set of timesteps

Model Parameters

Δ : duration of one timestep,	$[\Delta] = \text{h}$
P_h^{heat} : rated electrical power of the heater of household h,	$[P_h^{\text{heat}}] = \text{kW}$
η_h : heating efficiency of household h,	$[\eta_h] = 1$
α_h : proportional thermal loss coefficient of household h,	$[\alpha_h] = 1$
β_h : constant thermal loss coefficient of household h,	$[\beta_h] = \text{kW}$
$D_{h,t}$: thermal energy demand of household h at timestep t,	$[D_{h,t}] = \text{kW}$
$E_h^{\min}$: minimum storage capacity,	$[E_h^{\min}] = \text{kWh}$
$E_h^{\max}$: maximum storage capacity,	$[E_h^{\max}] = \text{kWh}$
E_h^{init} : initial stored energy,	$[E_h^{\text{init}}] = \text{kWh}$
P_t^{pv} : available PV generation at timestep t,	$[P_t^{\text{pv}}] = \text{kW}$
C^{grid} : grid usage cost,	$[C^{\text{grid}}] = €/\text{kWh}$
$C_{h,t}^{\text{sup}}$: supplier electricity tariff,	$[C_{h,t}^{\text{sup}}] = €/\text{kWh}$
$C_t^{\text{feed-in}}$: PV feed-in tariff,	$[C_t^{\text{feed-in}}] = €/\text{kWh}$

Decision Variables

$x_{h,t} \in \{0, 1\}$	heater activation decision,	$[x_{h,t}] = 1$
$e_{h,t} \in \mathbb{R}_+$	stored thermal energy,	$[e_{h,t}] = \text{kWh}$
$p_{h,t}^{\text{grid}} \in \mathbb{R}_+$	grid power assigned to household h,	$[p_{h,t}^{\text{grid}}] = \text{kW}$
$p_{h,t}^{\text{pv}} \in \mathbb{R}_+$	PV power assigned to household h,	$[p_{h,t}^{\text{pv}}] = \text{kW}$
$p_t^{\text{feed-in}} \in \mathbb{R}_+$	PV power exported to the grid,	$[p_t^{\text{feed-in}}] = \text{kW}$

Objective Function and Constraints

$$\min \sum_{h \in \mathcal{H}} \sum_{t \in \mathcal{T}} p_{h,t}^{\text{grid}} \cdot \Delta \cdot (C^{\text{grid}} + C_{h,t}^{\text{sup}}) - \sum_{t \in \mathcal{T}} p_t^{\text{feed-in}} \cdot \Delta \cdot C_t^{\text{feed-in}} \tag{5.3}$$

$$\text{s.t. } e_{h,t+1} = e_{h,t}(1 - \alpha_h)^{\Delta} - \beta_h \Delta - D_{h,t} \Delta + x_{h,t} P_h^{\text{heat}} \eta_h \Delta \qquad \forall h \in \mathcal{H}, t \in \mathcal{T} \tag{5.4}$$

$$e_{h,1} = E_h^{\text{init}} \qquad \forall h \in \mathcal{H} \quad (5.5)$$

$$e_{h,T+1} \geq E_h^{\text{init}} \qquad \forall h \in \mathcal{H} \quad (5.6)$$

$$E_h^{\max} \geq e_{h,t} \geq E_h^{\min} \qquad \forall h \in \mathcal{H}, t \in \mathcal{T} \quad (5.7)$$

$$p_{h,t}^{\text{pv}} + p_{h,t}^{\text{grid}} = x_{h,t} P_h^{\text{heat}} \qquad \forall h \in \mathcal{H}, t \in \mathcal{T} \quad (5.8)$$

$$\sum_{h \in \mathcal{H}} p_{h,t}^{\text{pv}} + p_t^{\text{feed-in}} = P_t^{\text{pv}} \qquad \forall t \in \mathcal{T} \quad (5.9)$$

$$p_{h,t}^{\text{pv}}, p_{h,t}^{\text{grid}} \geq 0 \qquad \forall h \in \mathcal{H}, t \in \mathcal{T} \quad (5.10)$$

$$p_t^{\text{feed-in}} \geq 0 \qquad \forall t \in \mathcal{T} \quad (5.11)$$

$$x_{h,t} \in \{0, 1\} \qquad \forall h \in \mathcal{H}, t \in \mathcal{T} \quad (5.12)$$

(5.3) **Objective function**: minimizes the total electricity costs of the households while accounting for revenues from feeding electricity back to the grid.

(5.4) **Stored energy**: governs the evolution of the energy stored in each household's tank, accounting for heat losses, demand, and energy supplied by the heating element.

(5.5),(5.6) **Initial and final energy**: fixes the initial tank energy (E_h^{init}) and enforces the cyclic constraint that the final stored energy is at least the initial level.

(5.7) **Capacity bounds**: ensures that the stored energy remains within the minimum ($E_h^{\min}$) and maximum capacity ($E_h^{\max}$) of each tank.

(5.8) **Heater power balance**: ensures that the power supplied to the heating element (P_h^{heat}) is provided by either PV generation or the grid.

(5.9) **PV power balance**: distributes the available PV generation (P_t^{pv}) among the households or feeds the remaining electricity into the grid.

(5.10),(5.11) **Non-negativity constraints**: ensure that electricity flows are non-negative.

(5.12) **Binary control**: ensures that the heating element is either active or inactive at each timestep.

5.2 Framework Implementation

This section describes the implementation of the heuristic framework proposed in Chapter 4 for the HoWaFlex case study, defining the problem-specific implementations of the generic structures and problem-specific operations.

5.2.1 Implementation of the Generic Structures

We define the following problem-specific implementations of the generic structures introduced in Section 4.1.1.

`HoWaFlexInstance` Specific implementation of `GenericInstance`. This structure stores the *Model Parameters* defined in Section 5.1.1. In addition, it precomputes two quantities used by the greedy algorithm described in Section 5.2.2 to prioritize heater activations. The first quantity is the average grid cost C_h^{avg}, where $[C_h^{\text{avg}}] =$ €/kWh, computed as

$$C_h^{\text{avg}} = C^{\text{grid}} + \frac{1}{T}\sum_{t=1}^{T} C_{h,t}^{\text{sup}}, \tag{5.13}$$

which represents the average cost of drawing electricity from the grid for household h.

The second precomputed quantity is the required energy factor $\rho_{h,d}$, which is recursively computed as

$$\rho_{h,d+1} = \frac{\rho_{h,d}}{(1-\alpha_h)^{\Delta}} + \frac{\beta_h \Delta}{(E_h^{\max} - E_h^{\min})(1-\alpha_h)^{\Delta}}, \tag{5.14}$$

with $\rho_{h,0} = 1$. It represents the amount of thermal energy that needs to be stored, to ensure that 1 kWh of thermal energy remains available after d timesteps while compensating for thermal losses.

These precomputed values are subsequently used by the greedy algorithm presented in Section 5.2.2 to efficiently estimate the priority of possible heater activations without repeatedly performing the corresponding calculations during the construction procedure.

`HoWaFlexSolution` Specific implementation of `GenericSolution`. This structure stores the *Decision Variables* defined in Section 5.1.1. Furthermore, in order to efficiently calculate the set of potential boiler activations, it maintains two additional vectors representing the current bounds of the stored thermal energy within the respective tanks.

First, to ensure that the thermal energy stored in the tank never falls below its prescribed lower bound, we introduce the current feasible minimum energy level of the tank $e_{h,t}^{\min}$, where $[e_{h,t}^{\min}] =$ kWh, for each timestep t. This vector is used by the greedy algorithm presented in Section 5.2.2 to restrict the set of feasible heater activations throughout the search.

In most cases, $e_{h,t}^{\min}$ is equal to $E_h^{\min}$. However, due to the cyclic constraint requiring the energy level at the end of the planning horizon to be at least E_h^{init}, as well as potentially large peaks in hot water demand, $e_{h,t}^{\min}$ may vary over time.

Note that, in the current implementation, this vector remains unchanged throughout the execution of the algorithm and could therefore also be stored as part of the

`HoWaFlexInstance`. Nevertheless, to maintain a consistent design we group all vectors related to the stored thermal energy within the `HoWaFlexSolution`, which is why it is introduced here.

Second, the vector storing the current feasible maximum energy level of the tank $e^{\max}_{h,t}$, where $[e^{\max}_{h,t}] = \text{kWh}$, is used to efficiently determine the feasibility of heater activations. Initially, the vector is initialized as $e^{\max}_{h,t} = E^{\max}_h$ for all timesteps $t = 1, \ldots, T$. During the execution of the algorithm, it is updated according to

$$e^{\max}_{h,t-1} = \frac{e^{\max}_{h,t} + \beta_h \cdot \Delta + D_{h,t} \cdot \Delta - x_{h,t} \cdot P^{\text{heat}}_h \cdot \eta_h \cdot \Delta}{(1 - \alpha_h)^{\Delta}}. \tag{5.15}$$

and represents the maximum thermal energy that may be stored at each timestep in order to avoid violating the storage capacity constraint. Note, that this computation represents the inverse operation of the update of the stored thermal energy described in Equation 5.1.

While activating the heater at timestep t' increases the stored thermal energy $e_{h,t}$ within the tank for all subsequent timesteps $t > t'$, it simultaneously decreases the feasible maximum energy level $e^{\max}_{h,t}$ for all preceding timesteps $t < t'$. This is because, in order to avoid violating the storage capacity constraint at timestep t', the thermal energy stored before the activation must be kept below $E^{\max}_h$, as the fixed boiler activation at timestep t' will contribute additional energy that would otherwise cause the storage capacity to be exceeded.

Consequently, when we want to determine the feasibility of a heater activation at timestep t we can check whether the resulting $e_{h,t+1}$ would exceed the stored $e^{\max}_{h,t+1}$ and if that is the case eliminate that timestep t from the search.

Additionally, a priority queue Q^{best} storing the currently most favorable heater activation for each household is maintained as part of the `HoWaFlexSolution`. It is used by the greedy algorithm presented in Section 5.2.2 to efficiently determine the next heater activation to be selected.

`HoWaFlexConfiguration` Specific implementation of `GenericConfiguration`. This structure stores the following hyperparameters, which are used to control the algorithms presented in Section 5.2.2 and 5.2.3:

W^{size} Defines the size of the temporal window used to select heater activations.

k^{rcl} Defines the maximum size of the RCL used to introduce randomization into the heuristic.

r^{destroy} Defines how much of a solution is destroyed during the `next_neighbor` operation.

$M^{\text{solver}} \in \{\texttt{fix}, \texttt{start}\}$ Specifies whether the MILP model during the `solve` operation is restricted by fixing the selected decisions (`fix`) or whether these decisions are provided as initial values (`start`) when computing the objective value.

`HoWaFlexState` Specific implementation of `GenericState`. This structure stores an iteration counter tracking the current execution of `generate_solution` and a flag indicating whether the current call originates from the `generate_solution` or `next_neighbor` operation. Since both operations rely on the same underlying methods for the greedy construction of a solution and the repair of a partially destroyed solution, this information allows the metaheuristics presented in Section 4.2 to distinguish between the two contexts. In particular, it ensures that the first invocation of `generate_solution` starts from the purely greedy solution before introducing randomized decisions, while repairs performed by `next_neighbor` always select the most favorable decisions.

`HoWaFlexProblem` Specific implementation of `StatefulProblem`.

5.2.2 Implementation of `generate_solution`

The construction heuristic operates according to a greedy principle and aims to find a heater control sequence that not only satisfies the hot water demand for each household but is also as cost-efficient as possible. This is archived by first, filtering all possible heater activations for each household and then evaluating them and identifying the most favorable one. As will be discussed in more detail later, we consider the most favorable activation to be the one that results in the highest cost savings compared to the average activation cost C_h^{avg} of the respective household introduced in Equation 5.13. The pseudocode of the final algorithm is presented in Algorithm 5.1.

The construction procedure starts from an empty boiler control sequence, where no heater activations are initially scheduled. This typically causes the initially stored thermal energy E_h^{init} to be depleted quickly for each household h due to hot water demand $D_{h,t}$ and thermal losses represented by α_h and β_h. Formally, this *critical timestep*, at which the demand can no longer be satisfied, can be identified for each household h by comparing $e_{h,t}$ with $e_{h,t}^{\min}$ for all timesteps t. This timestep is denoted by t_h^{crit} and is computed as

$$t_h^{\text{crit}} = \min \left\{ t \in T \mid e_{h,t} < e_{h,t}^{\min} \right\}. \tag{5.16}$$

Note that satisfying the demand at timestep t_h^{crit} requires at least one additional heater activation at some timestep $t \leq t_h^{\text{crit}}$. Consequently, the search for a suitable activation for household h can be restricted to the interval $1, \ldots, t_h^{\text{crit}}$.

In theory, this interval could also be restricted from the opposite direction by determining the earliest timestep at which a heater activation would provide sufficient thermal energy at t_h^{crit}. However, preliminary tests indicated that this intuitive restriction offers little practical benefit, while introducing additional computational overhead. Therefore, with computational efficiency in mind, we employ a comparatively simpler and stricter approach. We introduce the *activation window*, defined as the interval

$$\left[\max \left(t_h^{\text{crit}} - W^{\text{size}}, 1 \right), t_h^{\text{crit}} \right], \tag{5.17}$$

Algorithm 5.1: Implementation of `generate_solution` for the HoWaFlex problem

Input: `HoWaFlexProblem` p, restricted candidate list size k^{rcl}, activation window size W^{size}
Output: `HoWaFlexSolution` s

Initialize an empty `HoWaFlexSolution` s;
for $h \in \mathcal{H}$ **do**
  `SetBestActivation(`h`);`
end
while s is not feasible **do**
  Pop the highest priority household h and timestep t_h^{best} from Q^{best};
  `ActivateBoiler(`h, t_h^{best}`);`
end
return s

Function `ActivateBoiler(`h, t`)`**:**
  Activate the heater by setting $x_{h,t} \leftarrow 1$;
  Reduce the available PV power $P_t^{\text{pv}} \leftarrow P_t^{\text{pv}} - P_{h,t}^{\text{pv}}$;
  Update $e_{h,t'}$ for all $t' \in \{t, \dots, T\}$;
  Update $e_{h,t'}^{\max}$ for all $t' \in \{t, \dots, 1\}$;
  if $\exists t : e_{h,t} < e_{h,t}^{\min}$ **then**
    `SetBestActivation(`h`);`
  end
  for $h' \in Q^{\text{best}}$ *where* $t_{h'}^{\text{best}} = t_h^{\text{best}}$ **do**
    `SetBestActivation(`h'`);`
  end
end

Function `SetBestActivation(`h`)`**:**
  Set critical timestep $t_h^{\text{crit}} \leftarrow \min\{t \in T \mid e_{h,t} < e_{h,t}^{\min}\}$;
  Compute the start of the activation window $t^{\text{start}} \leftarrow \max(t_h^{\text{act}} - W^{\text{size}}, 1)$;
  Set candidates $C \leftarrow \{t \mid t^{\text{start}} \leq t \leq t_h^{\text{crit}},\ x_{h,t} \text{ is a feasible activation}\}$;
  Compute $\pi_{h,t}$ for all $t \in C$ and set the resulting $p_{h,t}^{\text{grid}}$ and $P_{h,t}^{\text{pv}}$;
  Sort C in descending order of $\pi_{h,t}$;
  Select t_h^{best} uniformly at random from the first $\min(k^{\text{rcl}}, |C|)$ elements of C;
  Set t_h^{best} as the best activation of household h in Q^{best} with priority $\pi_{h,t_h^{\text{best}}}$
end

which only contains timesteps in the close vicinity of t_h^{crit}. The size of this window is controlled by the hyperparameter W^{size}. The underlying intuition is that activating the heater long before the demand violation occurs is generally undesirable, even if this would

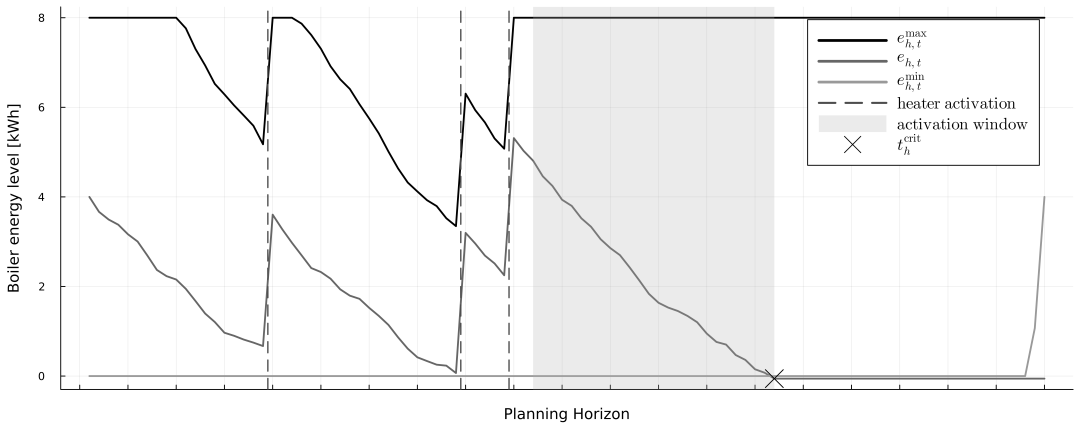


Figure 5.2: Visualization of the stored thermal energy $e_{h,t}$ together with the corresponding lower and upper energy bounds $e^{\min}_{h,t}$ and $e^{\max}_{h,t}$ for a sample household h over the planning horizon.

provide sufficient stored thermal energy to satisfy the demand D_{h,t^{crit}_h}. Although earlier timesteps may occasionally offer slightly lower energy prices, activating the heater too early increases the stored energy level over an extended period and consequently results in severe thermal losses.

With the relevant activation horizons for all households established, we can now consider another important property of the binary decision variables $x_{h,t}$. These variables exhibit a certain degree of independence, as fixing a single variable $x_{h,t}$, i.e., activating or deactivating the heater of household h at timestep t, only affects variables associated with either the same household h or the same timestep t.

For variables associated with the same household h, activating or deactivating the heater at timestep t' affects both future and past timesteps. Activating the heater increases the stored thermal energy at subsequent timesteps, therefore the energy levels $e_{h,t}$ must be updated for all timesteps t with $t > t'$ according to Equation 5.1. Conversely, the additional energy introduced at timestep t' reduces the available storage capacity in preceding timesteps. Therefore, the current feasible maximum energy level $e^{\max}_{h,t}$ must be updated backwards for all timesteps t with $t < t'$ according to Equation 5.15.

These updates do not need to be propagated across the entire energy vectors. For the stored thermal energy $e_{h,t}$, the forward propagation can be terminated once the effect of the activation is no longer sufficient to change the feasibility of the corresponding minimum energy constraint, i.e., once the updated energy level falls below $e^{\min}_{h,t}$ and the hot water demand $D_{h,t}$ at timestep t can no longer be satisfied. Similarly, the backward propagation of the current feasible maximum energy levels $e^{\max}_{h,t}$ can be stopped as soon as a stricter upper bound is encountered, which is typically the maximum storage capacity $E^{\max}_h$. A visualization of the resulting energy vectors for a sample household and their evolution over the planning horizon is provided in Figure 5.2.

For variables sharing the same timestep t, activating the heater of household h requires electrical power, which is supplied either by the PV system or the grid. If the required power is partially supplied by the PV system, less PV generation remains available for the other households at timestep t. Consequently, their grid supply and the resulting costs $c_{h,t}$, introduced in Equation 5.2, may change.

The priority $\pi_{h,t}$, according to which the most favorable feasible heater activation is selected, is computed as

$$\pi_{h,t} = c_{h,t} \cdot \left(1 + \rho_{h,t_h^{\text{crit}}-t} \cdot 10^{-3}\right) - C_h^{\text{avg}}. \tag{5.18}$$

It combines the activation cost $c_{h,t}$ of household h at timestep t, introduced in Equation 5.2, the average grid cost C_h^{avg}, introduced in Equation 5.13, and the required energy factor $\rho_{h,d}$, introduced in Equation 5.14. The inclusion of the required energy factor $\rho_{h,d}$ was originally intended to account for thermal losses between the considered activation timestep t and the critical timestep t_h^{crit}. Specifically, it penalizes activations that occur far before the critical timestep by increasing the priority value based on the additional energy required to compensate for these losses, rather than relying solely on the activation window to exclude such candidates. However, after preliminary hyperparameter tuning, this term was found to primarily serve as a tie-breaking mechanism. The scaling factor 10^{-3} was selected during the same tuning process.

The average grid cost C_h^{avg} is subtracted from the priority value to account for differences in energy supplier tariffs between households. This transforms the priority measure into a rough estimate of the expected cost savings in €/kWh resulting from activating the heater of household h at timestep t.

Although all costs are expressed in €/kWh and therefore do not directly account for differences in heater power ratings, the available PV generation implicitly affects the cost calculation. In particular, if only a small amount of PV power remains available at a timestep, heaters with higher power requirements consume a larger share of grid electricity and may therefore achieve lower cost savings.

Randomization of the construction process is achieved by applying Restricted Candidate Lists (RCLs) at the individual household level. For each household, the priority $\pi_{h,t}$ is computed for all timesteps within the current activation window, and one of the k^{rcl} most favorable activation candidates is selected uniformly at random. The selected activation is subsequently stored in the priority queue Q^{best}.

Applying the RCL at the household level significantly reduces the size of the priority queue and avoids repeatedly updating a large number of candidate activations. This is particularly beneficial because a heater activation typically shifts the corresponding activation window, causing many previously considered activation candidates to become invalid and requiring the queue to be updated. The main difference compared to an RCL containing all feasible activations occurs when multiple households have candidates with similar priorities, as the selected activation may not belong to the global k^{rcl}

best candidates. However, this situation only arises when several households exhibit comparable priorities, and preliminary experiments indicate that this approximation has a negligible impact on solution quality.

Finally, note that this procedure is a greedy heuristic and therefore does not produce a proven optimal solution. This can be illustrated by a simple counterexample. Consider an instance with $T = 10$ and a single household h. Due to variable electricity prices assume that for all $t \geq 6$ the costs $c_{h,t}$ is very high, effectively restricting all economically feasible activations to the interval $[1, 5]$. By choosing suitable demand and heat loss parameters, an instance can be constructed in which the boiler must be activated at least twice, while the storage capacity constraint $e_{h,t}^{\max}$ enforces a minimum separation of three timesteps between consecutive activations.

The optimal solution therefore activates the heater at timesteps 1 and 5. An activation at any intermediate timestep $t \in 2, 3, 4$ prevents both an earlier and a later activation within the interval $[1, 5]$, leaving only a single feasible activation in this low-cost region. Now suppose that one of the timesteps $t \in 2, 3, 4$ has a slightly lower cost than timesteps 1 and 5. The greedy heuristic will select this seemingly most attractive activation first. As a consequence, no additional feasible activation remains within the interval $[1, 5]$, forcing the algorithm to schedule the second activation at some timestep $t \geq 6$. Since these timesteps incur substantially higher costs, the resulting solution is strictly suboptimal.

5.2.3 Implementation of `next_neighbor`

We compute neighboring solutions for this problem using a destroy/repair operation. In this context, destroying a solution means removing heater activations from the control sequence, while repairing corresponds to reinserting activations until the sequence becomes feasible again. The pseudocode of the final implementation of the `next_neighbor` operation is presented in Algorithm 5.2.

Selecting activations uniformly at random is unlikely to encourage the repair procedure to explore a different region of the search space. Instead, it would typically reconstruct a solution similar to the original one, as most surrounding activations remain unchanged. We therefore remove activations that are likely to dependent on each other, namely activations that either occur at the same timestep t or belong to the same household h and are consecutive in time. The extent of the destroy operation is controlled by the hyperparameter r^{destroy}, which specifies the fraction of the solution to be removed.

To select such a set of related activations, we first sample an integer n^{house} from the normal distribution $\mathcal{N}(r^{\text{destroy}} \cdot H, 1)$ and clamp it to the interval $[1, H]$. This value specifies the number of households for which a contiguous part of the control sequence is removed. The number of consecutive activations removed for each selected household, denoted by n^{time}, is then computed as

$$n^{\text{time}} = T \cdot \frac{r^{\text{destroy}} \cdot H}{n^{\text{house}}}, \tag{5.19}$$

Algorithm 5.2: Implementation of `next_neighbor` for the HoWaFlex problem. The operations `SetBestActivation` and `ActivateBoiler` are defined in Algorithm 5.1

Input: `HoWaFlexProblem` p, `HoWaFlexSolution` s, destroy rate r^{destroy}
Output: Neighboring `HoWaFlexSolution` s'

Initialize s' as a copy of s;
`// destroy procedure`
Sample integer $n^{\text{house}} \sim \mathcal{N}(r^{\text{destroy}} \cdot H, 1)$ and clamp to interval $[1, H]$;
Compute $n^{\text{time}} \leftarrow T \cdot r^{\text{destroy}} \cdot \frac{H}{n^{\text{house}}}$ and clamp to $[1, T]$;
Select t^{first} uniformly at random from $\{1, \ldots, T - n^{\text{time}}\}$;
for each household h in a uniformly random subset of $\mathcal{H}$ of size n^{house} **do**
    for $t = t^{\text{first}}$ **to** $t^{\text{first}} + n^{\text{time}} - 1$ **do**
        `DeactivateBoiler(`h, t`)`;
    end
end
`// repair procedure, similar to Algorithm 5.1`
while s' is not feasible **do**
    if Q^{best} is not empty **then**
        Pop the highest priority household h and timestep t_h^{best} from Q^{best};
        `ActivateBoiler(`h, t_h^{best}`)`;
    else
        Find the first timestep $t > t_h^{\text{best}}$ for which $x_{h,t} = 1$ and set it as t^{blocking};
        `DeactivateBoiler(`h, t^{blocking}`)`;
    end
end
return s';

Function `DeactivateBoiler(`h, t`)`**:**
    Deactivate the heater by setting $x_{h,t} \leftarrow 0$;
    Increase the available PV power $P_t^{\text{pv}} \leftarrow P_t^{\text{pv}} + P_{h,t}^{\text{pv}}$;
    Update $e_{h,t'}$ for all $t' \in \{t, \ldots, T\}$;
    Update $e_{h,t'}^{\max}$ for all $t' \in \{t, \ldots, 1\}$;
    for $h' \in Q^{\text{best}}$ **do**
        `SetBestActivation(`h'`)`;
    end
end

and subsequently clamped to the interval $[1, T]$. The resulting region of size $n^{\text{house}} \times n^{\text{time}}$ therefore corresponds approximately to a fraction r^{destroy} of the complete search space of size $H \times T$. The affected households and the first timestep starting from which consecutive activations are removed are selected uniformly at random.

Compared to fixing the number of households to $\lceil r^{\text{destroy}} \cdot H \rceil$, randomizing n^{house} allows multiple households to be affected at once even when $r^{\text{destroy}} \cdot H \leq 1$. Finally, note that the destroy operation removes a fixed number of activations rather than explicitly enforcing infeasibility. Consequently, it is theoretically possible for the solution to remain feasible after the destroy step, although this is unlikely in practice.

The repair process uses the greedy procedure described in the Algorithm 5.1. However, since the destroyed time window is not necessarily located at the end of the planning horizon, the newly generated control sequence may not transition smoothly back to the unchanged part of the original solution. To address this issue, we remove additional activations at the beginning of the remaining sequence and continue the repair process for the newly occurring activation horizon if necessary. Consequently, the repair procedure is not strictly limited to the originally selected time window and may, in rare cases, modify a slightly larger portion of the solution than specified by the destruction rate r^{destroy}.

5.2.4 Implementation of `solve`, `candidates` and `solution`

In the HoWaFlex problem, a solution component consists solely of a household h, a timestep t, and a heater decision x_t. The implementation of `solve` therefore first identifies the heater variables for which both an on and an off decision are present and then fixes those for which only a single value is specified in the IESopt model. The remaining heater decisions are left to the optimization solver. All other decision variables, corresponding to the various power flows, can be inferred from the heater decisions and are therefore are therefore also left to the solver. The restricted MILP model is then solved, and the resulting solution is returned as a `HoWaFlexSolution`.

Similarly, the problem-specific implementations of `candidates` and `solution` operate exclusively on the x_t variables and either extract the heater decisions as a vector or return a corresponding `HoWaFlexSolution` respectively.

5.3 Generation of Benchmark Instances

In order to accurately evaluate the presented algorithms, we conducted experiments on two different types of instances. The first set of instances is based on data collected from the Campus Domus student dormitory, an actual student housing community located in St. Pölten, the capital of Lower Austria. The data was collected as part of the HoWaFlex2Market research project conducted by the AIT Austrian Institute of Technology. Since each apartment in this community is equipped with the same type of boiler and the energy supplier is fixed by the project partners, the variability between individual households is somewhat limited. The households mainly differ in their hot water consumption profiles, which motivated the generation of the more generic instances with greater structural diversity.

The second set of instances is intended to represent generic energy communities, such as multi-family residential buildings, with a variety of apartment sizes and energy suppliers

across Austria. While the data is, whenever possible, collected from real-world sources, the resulting communities do not correspond to existing physical settlements.

5.3.1 Campus Domus

The first set of instances used to evaluate the HoWaFlex problem is based on real-world measurements collected at the Campus Domus[2] student dormitory in St. Pölten. The dormitory consists of 86 shared apartments, each accommodating up to three students and equipped with an individual electric hot water boiler. Unfortunately, neither the exact number of occupants per apartment nor the actual occupancy patterns are known. In particular, it is unclear whether the registered residents are present throughout the entire measurement period, as students may be absent for extended periods, for example by returning home during weekends or holidays. However, the project partners confirmed that the provided measurements originate exclusively from boilers installed in apartments that were rented during the respective measurement periods. Consequently, depending on the selected time range, measurements from approximately 60 boilers are available. Note that the boilers at Campus Domus are used exclusively for DHW preparation.

The dataset comprises measurements from four distinct periods: eight weeks from November to December 2025, one week in March 2026, three weeks from March to April 2026 and 15 weeks from April to July. In addition to the PV generation profile, P_t^{pv}, the dataset contains measurements from two types of sensors installed in each boiler. The first records the electrical input power, corresponding to the heater power P_h^{heat} whenever the heating element is active. The second measures the thermal energy stored in the boiler at each measurement timestamp. Based on these measurements, the boiler loss parameters α and β, as well as the corresponding domestic hot water demand profile $D_{h,t}$, were estimated for each apartment. The timestep length is set to 15 min ($\Delta = \frac{1}{4}$) to align with the temporal resolution of the day-ahead market.

The heater power P_h^{heat} is set to 2.4 kW, as specified by the project partner, with a boiler efficiency η_h estimated from the data and is slightly below one for all boilers. The maximum capacity $E_h^{\max}$ was not directly provided and is set to 3.3999 kWh for the purposes of this thesis. This value is motivated by the operating strategy currently employed by the project partner. In their rule-based control system, the boiler is activated whenever PV power is available, provided that the stored energy level is currently below 2.8 kWh, as significantly exceeding this level is expected to increase lime deposits. Since a boiler activation can increase the stored energy by at most

$$P_h^{\mathrm{heat}} \cdot \eta_h \cdot \Delta \leq 2.4\,\mathrm{kW} \cdot 1 \cdot \frac{1}{4} = 0.6\,\mathrm{kWh}, \tag{5.20}$$

setting $E_h^{\max} = 3.3999\,\mathrm{kWh}$ allows a boiler activation only when the currently stored energy level before an activation is at most 2.7999 kWh. This therefore closely mimics the operational constraint implemented by the project partner.

[2] https://www.campus-domus.at/

Table 5.1: Energy tariffs considered for the Campus Domus benchmark instances. The tariff type identifier denotes fixed (F) and variable (V) pricing models. The eFRIENDS tariff consists of a fixed surcharge in addition to the day-ahead market price C_t^{da}, whereas the grünstrom 12 tariff remains fixed over the entire planning horizon.

Type	Supplier	Tariff [€/kWh]
V	eFRIENDS – der bessere FLEX15	$C_t^{\mathrm{da}} + 0.01$
	Feed-in: eFRIENDS – der bessere FLEX15	$C_t^{\mathrm{da}} - 0.01$
F	grünstrom 12	0.187
	Feed-in: eFRIENDS – der bessere FLOAT	0.049397

Table 5.2: Grid costs considered for the Campus Domus benchmark instances.

Year	Grid cost [€/kWh]
2025	0.0820
2026	0.0879

The minimum capacity $E_h^{\min}$ is set to 0 kWh, corresponding to the lower bound observed in the provided measurements. Consequently, Equation 5.23 cannot be used to derive the underlying boiler temperature for these instances, as the intake water temperature T^{I} is unknown. Furthermore, in contrast to the approach described in Section 5.3.2, the sensor measurements are likely calibrated with respect to the minimum tank temperature $T^{\min}$ rather than the intake water temperature T^{I}.

Since the optimization strategy currently employed by our project partner does not account for the electricity price at which the apartments purchase energy from the grid, we evaluate the Campus Domus instances under two different pricing scenarios. Although the project partner did not provide information about the exact electricity supplier, they indicated that all apartments are typically supplied under the same tariff. Consequently, we consider both a variable tariff linked to the day-ahead electricity market and a fixed-price tariff to assess the impact of electricity pricing on the optimization results. The considered energy tariffs are listed in Table 5.1. The tariff information was obtained on June 18, 2026, using the E-Control Tarifkalkulator[3].

The applicable grid costs depend on the year in which the measurements were recorded. The corresponding values were obtained from the Austrian Bundeskanzleramt[4]. The resulting grid costs are summarized in Table 5.2.

This results in a total of eight instances being evaluated in the benchmark, consisting of

[3] https://www.e-control.at/tarifkalkulator

[4] https://ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2024_II_370/BGBLA_2024_II_370.pdf
https://ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2025_II_305/BGBLA_2025_II_305.pdf

Table 5.3: Time periods investigated in the Campus Domus benchmark instances. The table provides the measurement start time, the number of timesteps T, and the number of households H for each instance.

	Nov–Dec 2025	Mar 2026	Apr 2026	Apr–Jul 2026
Start date	Nov. 5, 2025	Mar. 23, 2026	Mar. 29, 2026	Apr. 22, 2026
Start time	10:00	07:15	13:30	02:00
Timesteps T	5,432	672	2,305	7,227
Households H	60	61	62	57

four different time periods, as specified in Table 5.3, and two different pricing models, as specified in Table 5.1.

5.3.2 Generic Communities

The instances described in this section were designed to incorporate as many factors of the problem as possible in order to challenge the tested algorithms. The instances are set in the year 2025 since this is the most recent year, at the time of writing this thesis, for which a complete set of data is available. Furthermore, to account for the 15-minute intervals at which varying energy prices are available, we set $\Delta = \frac{1}{4}$.

To account for varying weather conditions throughout Austria, we generate one instance for each of ten cities distributed across Austria. Since the data for all cities was obtained from the same sources, we do not discuss each city individually and instead describe the generation process in general.

PV Profile. The first dataset required for generating an instance is a time series representing the PV generation profile. The PV generation profiles were obtained from the data provided by Pfenninger and Staffell [PS16]. We assume a PV system with a nominal PV peak capacity of $P^{\text{pv}}_{\text{nom}} = 20\,\text{kWh}$ and use the default orientation, which corresponds to the presumably optimal configuration with a tilt of 35° and a south-facing orientation. The system loss was set to 0.1. Since the generated PV profiles are available at an hourly resolution, the energy generated during each hour was distributed uniformly across the corresponding timesteps.

Heating Demand. We incorporate both space heating and DHW. The data for space heating was also obtained from the work of Pfenninger and Staffell [PS16]. We again use the default parameters. Rather than providing values in a concrete unit, their model computes relative factors that must be normalized before use. We normalize the data by computing the mean of the annual totals across Austria and divide each time series by this mean. Multiplying the resulting factor by the expected average annual heating demand of a household yields the heating demand for the year 2025 while accounting for

local weather differences. We assume an average annual heating demand of 50 kWh/m^2, which is representative of a relatively recently constructed residential building[5].

Domestic Hot Water Demand. The DHW demand is based on the model developed by Chmielewska [Chm25]. While the model is primarily based on data from Poland rather than Austria, it uses real measurements from multi-family residential buildings and is therefore considered sufficiently suitable for the purposes of this work. In their work, they propose methods to compute the average daily domestic hot water consumption $\bar{V}_d$, where $[\bar{V}_d] = \mathrm{m}^2/\mathrm{d}$, as well as correction factors for different weekdays s_{d_j}, months s_{m_i}, and hours of the day s_{h_n}. Since we do not want to assume a specific number of rooms for each of the generic apartments, we used the average correction factors for apartments with two or more rooms, as our generated apartments were generally too large to realistically consist of only one room. The average daily domestic hot water consumption can therefore be computed from the apartment size A_R, where $[A_R] = \mathrm{m}^2$, as

$$\bar{V}_d = 0.00153 \cdot A_R + 0.011. \tag{5.21}$$

Furthermore, the expected daily consumption for a specific weekday i and month j is computed as

$$V_{d_{i,j}} = s_{d_i} \cdot s_{m_j} \cdot \bar{V}_d. \tag{5.22}$$

Finally, we do not want every household to have the same DHW consumption profile and also wanted to capture occasional peaks caused by activities such as taking a shower. Therefore, instead of directly using the expected daily consumption, we randomize $V_{d_{i,j}}$ within an interval of $\pm 5\,\%$ and distribute the resulting demand across the timesteps of that day. This is achieved by defining a minimum consumption of 0.2 kW per timestep, which roughly corresponds to washing one's hands, and a maximum consumption of 8 kW per timestep, which approximately corresponds to taking a shower. Timesteps are then selected randomly with probabilities proportional to the corresponding hourly correction factors. Consequently, timesteps with higher expected consumption are selected more frequently. Whenever a timestep is selected, a random amount of consumption within the defined bounds is assigned to it until the complete daily DHW demand $V_{d_{i,j}}$ has been distributed. While this approach reproduces the expected average demand over time, it does not account for the weather conditions of a particular day or location, nor does it consider the influence of public holidays.

Hot Water Boiler. The next step in the instance generation is the selection of the boilers, or more precisely, a storage tank and a heating element. To fulfill both the space heating and domestic hot water demand, we select so-called Hygiene Storage systems. These systems store only the hot water used for space heating, while DHW is heated on demand by pumping fresh tap water through a spiral heat exchanger inside the storage tank. This design not only allows both domestic hot water and space heating demand

[5] https://www.thermondo.de/info/rat/heizen/heizwaermebedarf-ermitteln/

Table 5.4: Household configurations considered for the Generic Communities benchmark instances.

Parameter	Unit	Small	Medium	Large
Tank model		BCW300K	BCW500K80	BCW750F
Heater model		BVZ400	BVZ401	BVZ402
Apartment size	[m^2]	60	90	120
Tank volume	[L]	255	455	698
Tank diameter	[m]	0.67	0.80	1.01
Tank height	[m]	1.595	1.765	1.665
Constant heat loss	[W]	51	74	125
Heater power	[kW]	3.0	4.5	6.0
Heater efficiency factor	[1]	1.0	1.0	1.0

to be supplied from the same tank, but also eliminates problems related to legionella proliferation, since no domestic hot water is stored for extended periods of time. The tank- and heater-specific parameters were obtained from the following website[6]. In total, we define three different household configurations, each consisting of a storage tank, a matching heating element, and an apartment size suitable for such a system. The corresponding configurations are shown in Table 5.4.

Furthermore, we assume that the minimum tank temperature $T_h^{\min}$ is 42 °C and the maximum temperature $T_h^{\max}$ is 80 °C, which is the highest temperature supported by the selected heating elements. The initial temperature T_h^{init} was set to 45 °C We further assumed an intake water temperature T^{I} of 10 °C and an ambient room temperature T^{R} of 20 °C. Since all storage-related variables and constants in the HoWaFlex problem are expressed in kWh, we convert a given temperature $T_h^\star$, where $[T_h^\star] = °\text{C}$, into the corresponding stored energy level $E_h^\star$, where $[E_h^\star] = \text{kWh}$, using the following formula,

$$E_h^\star = V \cdot (T_h^\star - T^{\text{I}}) \cdot \frac{1.16}{1000}. \tag{5.23}$$

Here, V denotes the volume of the tank in L. The factor 1.16 approximates the volumetric heat capacity of water, expressed in $\text{W h}/(\text{L K})$. Dividing by 1,000 converts the resulting energy from Wh to kWh.

Energy Losses. Next, considerations were made regarding the heat losses of the storage tanks. Since we do not have access to an explicit heat loss model for the considered tanks, only the constant heat loss L_β, where $[L_\beta] = \text{W}$, provided by the manufacturer could be used for β_h, leaving $\alpha_h = 0$. However, assuming a constant heat loss is not only an unrealistic simplification but would also reduce the complexity of the optimization problem, as maintaining a lower storage temperature would no longer provide an implicit

[6] https://www.g2-energy-systems.de/speichersysteme/hygienespeicher-pufferspeicher/hygienespeicher-pufferspeicher-ohne-waermetauscher/300-l-hygienespeicher-ohne-waermetauscher-bcw300k.html

advantage through reduced thermal losses. To account for this effect we model the temperature-dependent heat loss using the standard heat transfer relation

$$\text{Heat Loss} = U \cdot A \cdot (T - T^{\mathrm{R}}), \tag{5.24}$$

where A denotes the surface area of the storage tank in m^2, T denotes the storage temperature in °C, T^{R} denotes the ambient reference temperature in °C, and U denotes the overall heat transfer coefficient of the tank insulation in $\mathrm{W}/(\mathrm{m}^2\,\mathrm{K})$. We assume the storage tanks to be cylindrical and compute their surface areas from the dimensions provided by the manufacturer.

The insulation U-value is estimated from the specified constant heat loss L_β by assuming that the reported heat loss was measured at a constant tank temperature of 65 °C, as specified in DIN EN IEC 60379:2023-11 [DIN23]. The U-value is therefore calculated as

$$U\text{−value} = \frac{L_\beta}{A \cdot (65 - 20) \cdot 1000}. \tag{5.25}$$

The resulting heat loss coefficient α is then computed as

$$\alpha = U\text{−value} \cdot A \cdot \frac{1000}{V \cdot 1.16}, \tag{5.26}$$

while the constant heat loss parameter β is set to 0 for all generated instances.

Pricing Model. The last dataset required for an instance of the HoWaFlex problem is the price at which a household can buy electricity from the grid. For this, we compare two variable tariffs depending on the spot market price and one fixed tariff. All tariffs were obtained on June 18, 2026 from the E-control Tarifkalkulator[7]. As only the energy used for heating and DHW is considered, the total annual electricity demand is unknown. Therefore, we use only the base energy rate of each tariff and exclude flat rates based on annual consumption. We use the prices provided in Table 5.5. Note that for the variable tariffs, the base energy rate is added to the day-ahead price C_t^{da}, where $[C_t^{\mathrm{da}}] = €/\mathrm{kWh}$, for which we use the prices provided for 2025 by the European Network of Transmission System Operators for Electricity (ENTSO-E) [Eur26].

Lastly, for the grid costs, we use the actual grid fees for the considered cities for the year 2025 as stated in Table 5.6, provided by Austrian Bundeskanzleramt[8].

Summing up, we consider three different household configurations, as specified in Table 5.4, three energy suppliers, as specified in Table 5.5, and ten different cities, as specified in Table 5.6. Consequently, we generate ten instances, one for each city, where each instance includes each configuration–supplier combination and therefore consists of exactly nine households. The consumption profiles are randomized both across households and across

[7] https://www.e-control.at/tarifkalkulator

[8] https://www.ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2024_II_370/BGBLA_2024_II_370.pdfsig

Table 5.5: Energy tariffs considered for the Generic Communities benchmark instances. The tariff type identifier denotes fixed (F) and variable (V) pricing models. The eFRIENDS and V-Strom-SPOT tariff consists of a fixed surcharge in addition to the day-ahead market price C_t^{da}, whereas the grünstrom 12 tariff remains fixed over the entire planning horizon. Since the PV system is shared among all households, we only consider a single feed-in tariff for all instances.

Type	Supplier	Tariff [€/kWh]
V	eFRIENDS - der bessere FLEX15	$C_t^{\text{da}} + 0.010$
	V-Strom-SPOT-H-KW4925	$C_t^{\text{da}} + 0.014$
	Feed-in: eFriends - der bessere FLEX15	$C_t^{\text{da}} - 0.010$
F	grünstrom 12	0.187

Table 5.6: Grid costs considered for the Generic Communities benchmark instances.

City	Grid costs [€/kWh]	City	Grid costs [€/kWh]
Bregenz	0.0540	Klagenfurt	0.0725
Deutsch-Wagram	0.0820	Linz	0.0528
Eisenstadt	0.0726	Salzburg	0.0747
Graz	0.0540	St. Pölten	0.0820
Innsbruck	0.0781	Wien	0.0740

instances to account for variability in demand patterns. Each instance covers the entire year 2025 with a timestep length of $\Delta = \frac{1}{4}$ hours. Therefore, each household has $\frac{24}{0.25} \cdot 365 = 35{,}040$ possible heater activations, resulting in a total of $\frac{24}{0.25} \cdot 365 \cdot 9 = 315{,}360$ decision variables per instance.

5.4 Experimental Setting

All computational experiments were conducted on the TU Wien HPC cluster of the Algorithms and Complexity Group[9] using a compute node equipped with a single AMD EPYC 9554P 64-Core processor. The implementation was written in Julia 1.12.5 [BEKS17] and executed in a single thread. Although parallelizing the heuristic implementation was investigated, it results in an increase in the overall runtime due to the associated overhead. In contrast, the employed MILP solvers, HiGHS 1.13.1 [HH18] and Gurobi 12.0.3 [Gur26], benefit from parallel execution and were therefore allowed to utilize up to eight threads.

Each problem instance is evaluated using all metaheuristics introduced in Chapter 4. In addition, the LP relaxation and the MILP formulation were solved independently to provide reference solutions. A time limit of 3,600 s was imposed on both the LP and

[9] https://www.ac.tuwien.ac.at/

MILP models, whereas all metaheuristics were limited to 1,000 s. The longer time limit for the exact optimization methods was chosen to obtain high-quality reference solutions and strong bounds against which the heuristic approaches could be evaluated, rather than relying solely on the LP relaxation.

Since each algorithm execution comprises both Julia code and calls to an external MILP solver, runtimes were measured as wall-clock time. For every combination of problem instance and randomized algorithm, a total of 30 independent runs were performed using different random seeds. This allows the results to be evaluated using statistically meaningful performance measures.

For the MILP solvers we relied primarily on the default parameters, with the only notable exception being the MIP gap being set to 10^{-5} which is slightly stricter than the default typically employed by the HiGHS [HH18] and Gurobi [Gur26] solvers, since we did not want that random noise created by rounding errors within the allowed gap to make an appearance in the final results.

Across all experiments, the construction heuristic used an activation window size of $W^{\text{size}} = \frac{24}{\Delta} = 96$, corresponding to the 96 timesteps in a single day. Consequently, to satisfy a given hot water demand, the algorithm was allowed to schedule a boiler activation at any timestep within the preceding 24 hours. Preliminary experiments showed that this configuration consistently outperformed larger activation windows spanning one week or one month and was therefore used for all benchmark instances.

Note that W^{size} remains unchanged across all experiments, even though the number of timesteps T varies between instances. Reducing W^{size} below 96 with $\Delta = \frac{1}{4}$ would result in an activation window shorter than one day, which is generally undesirable in practice. In particular, a shorter activation window may prevent the algorithm from scheduling heater activations during periods of PV generation to satisfy hot water demand at a later time. For example, PV power generated in the morning could no longer be used to satisfy hot water demand occurring later that night, despite it still being the same day.

For all metaheuristics using `next_neighbor`, the destroy rate was set to $r^{\text{destroy}} = 0.1$, meaning that approximately 10 % of the activations in the current solution were removed before the repair procedure. This comparatively large destroy rate was chosen because evaluating a candidate solution in the current implementation requires solving a JuMP model, which is considerably more computationally expensive than the destroy and repair operations themselves. Preliminary experiments with substantially smaller destroy rates, where only a few activations were modified per iteration, resulted in only marginal improvements between successive solutions. Consequently, the additional solver calls required to evaluate these small neighborhood changes outweighed their potential benefits.

For the GRASP algorithm, the maximum number of consecutive local search iterations without improvement was set to $I^{\max} = 10$. This value provided a suitable balance between intensifying the search around the current incumbent solution and restarting the search from a newly constructed solution. The RCL size was uniformly set to $k^{\text{rcl}} = 3$, as

preliminary experiments indicated that it introduced sufficient randomness while still favoring high-quality construction decisions.

For the CMSA algorithm, the number of solutions constructed in each iteration was set to $S^{\text{iter}} = 5$, while the maximum pool size was set to $C^{\text{lim}} = 1.05 \cdot T \cdot H$. Thus, the component pool contains at most approximately 5 % more components than a single feasible solution. This effectively restricts the reduced MILP model to approximately 5 % of the components in the original decision space. Furthermore, the maximum component age was set to $a^{\max} = 3$. However, this parameter rarely became active in practice, as the reduced MILP solver typically completed only one to three CMSA iterations within the given time limit. This was primarily due to the fact that the reduced MILP models, despite containing only approximately 5 % of the decision variables of a regular instance, remained computationally challenging for the employed mathematical optimization solvers.

Lastly, we investigate the performance of solving the MILP model when using the solution obtained by `generate_solution` as its starting solution. This allows the MILP solver to begin with a potentially strong incumbent solution rather than starting from scratch. This approach is enabled by setting $M^{\text{solver}} = \texttt{start}$ and is denoted as initialized MILP (IMILP) in the following evaluation.

5.5 Experimental Results

We conducted the experiments independently for the two benchmark datasets. The instances generated from data collected at the Campus Domus student dormitory are evaluated in Section 5.5.1, while the instances representing more generic energy communities are evaluated in Section 5.5.2.

5.5.1 Campus Domus

One major distinction between the Campus Domus instances and the generic community instances presented in Section 5.3.2 is the adopted pricing model. Since all households are supplied by the same electricity provider, they face identical electricity prices at every timestep. As a result, the heuristic could potentially be simplified, as it computes identical expected electricity costs for all households. This observation is particularly relevant for the fixed pricing model, where the optimization problem essentially reduces to deciding how many boilers to activate, and which ones, during periods of available PV generation and according to demand. These theoretical simplification further motivated the generation of the more complex generic communities presented in Section 5.3.2.

We begin by comparing the performance of the greedy construction heuristic (G) with that of solving the IESopt model using a conventional solver. The resulting average objective values over the 30 independent runs, are presented in Table 5.7. Unfortunately, HiGHS was unable to return any solutions to the IESopt models within the imposed time limit of 3,600 s, even for the smaller instances within this benchmark. Consequently, the corresponding results are omitted from the table. This is not only surprising but

Table 5.7: Comparison of the IESopt baseline and the greedy heuristic (G) over the Campus Domus benchmark instances. The *Price* column denotes the fixed (F) and variable (V) pricing model defined in Table 5.5. The objective value is denoted by $z^\star$, the relative gap is denoted by $\delta^\star$ and computed according to Equation 5.27, and the best bound reported by the MILP solver is denoted by b^{MILP}. The median of a metric across the 30 runs conducted for the greedy construction heuristic (G) is denoted by $\tilde{\cdot}$. Gurobi was used as optimization solver for the presented results.

Instance		**IESopt**				**Greedy**	
Timespan	Price	z^{LP} [€]	b^{MILP} [€]	z^{MILP} [€]	δ^{MILP} [%]	$\tilde{z}^{\text{G}}$ [€]	$\tilde{\delta}^{\text{G}}$ [%]
NovDez2025	F	2172.3	2176.2	2392.7	9.0	**2208.9**	1.5
NovDez2025	V	1481.3	1495.3	1818.8	17.8	**1550.7**	3.6
Mar2026	F	89.3	92.4	**104.3**	11.4	128.7	28.2
Mar2026	V	30.8	34.6	**42.5**	18.6	50.3	31.2
MarApr2026	F	-92.3	-86.0	138.3	162.2	**-11.0**	684.4
MarApr2026	V	-124.3	-114.9	77.9	247.5	**-68.2**	68.5
AprJul2026	F	-975.0	-966.2	461.4	309.4	**-873.0**	10.7
AprJul2026	V	-1305.8	-1274.5	-219.8	479.9	**-1118.7**	13.9

also highlights the need for heuristic approaches, in order to obtain practical solutions in commercial settings where budget constraints limit the use of commercial solvers such as Gurobi.

In Table 5.7, the columns z^{LP} and z^{MILP} denote the objective values obtained by solving the LP relaxation and MILP formulation obtained through the IESopt framework using Gurobi. The column b^{MILP} represents the best bound reported by the MILP solver after the time limit of 3,600 s was reached and is used as the reference value for the gap calculation. The median objective value over the 30 independent runs of the greedy construction heuristic for each instance is denoted by $\tilde{z}^{\text{G}}$. The relative gap $\delta^\star$, where $[\delta^\star] = \%$ is calculated as

$$\delta^\star = \frac{|z^\star - b^{\text{MILP}}|}{|z^\star|} \cdot 100, \tag{5.27}$$

where $z^\star$ denotes the objective value of the respective solution being evaluated. The median relative gap over the 30 independent runs is denoted by $\tilde{\delta}^{\text{G}}$. All reported values are rounded to one decimal place.

Note that the reported relative gaps are exceptionally large for the MarApr2026 instances. This is caused by the best MILP bound b^{MILP} being negative, resulting from the high amounts of energy exported to the grid and the corresponding feed-in revenue, while the objective values remain close to zero. As a result, the denominator in Equation 5.27 becomes smaller than the nominator, leading to disproportionately large relative gaps despite comparatively small absolute deviations. Consequently, for the MarApr2026 instance with the fixed pricing model, the greedy heuristic reports a substantially larger

Table 5.8: Comparison of the median relative gap to the MILP range $\tilde{\delta}^{\star}_{\text{range}}$ according to Equation 5.28, over the 30 runs conducted for each heuristic approach on the Campus Domus benchmark instances. The *Price* column denotes the fixed (F) and the variable (V) pricing model defined in Table 5.5. Gurobi was used as optimization solver for the presented results.

Instance		**IESopt**	**Heuristic Approaches**				
Timespan	Price	$\delta^{\text{MILP}}_{\text{range}}$ [%]	$\tilde{\delta}^{\text{G}}_{\text{range}}$ [%]	$\tilde{\delta}^{\text{LNS}}_{\text{range}}$ [%]	$\tilde{\delta}^{\text{GRASP}}_{\text{range}}$ [%]	$\tilde{\delta}^{\text{CMSA}}_{\text{range}}$ [%]	$\tilde{\delta}^{\text{IMILP}}_{\text{range}}$ [%]
NovDez2025	F	100.0	15.1	15.1	15.1	**13.6**	15.1
NovDez2025	V	100.0	17.1	17.1	17.1	17.1	**16.1**
Mar2026	F	**100.0**	305.7	305.3	305.7	305.7	152.2
Mar2026	V	**100.0**	198.4	197.3	198.4	198.4	110.9
MarApr2026	F	100.0	33.5	33.5	33.5	33.5	**33.3**
MarApr2026	V	100.0	24.2	24.2	24.2	24.2	**22.4**
AprJul2026	F	100.0	**6.5**	**6.5**	**6.5**	**6.5**	**6.5**
AprJul2026	V	100.0	14.8	14.8	14.8	14.8	**14.1**

mean relative gap $\tilde{\delta}^{\text{G}}$ than the corresponding $\tilde{\delta}^{\text{MILP}}$ obtained from the MILP model, even though its solution is in fact considerably closer to the MILP best bound than the solution obtained by the MILP model. We therefore encourage comparing the resulting objective values rather than the corresponding relative gaps.

In Table 5.8 we compare the different metaheuristics implemented within the proposed framework. We refrain from reporting the median relative gaps $\tilde{\delta}$ introduced in Equation 5.27, as, as illustrated in Table 5.7, they may not adequately capture the relationship between the solutions obtained by the different heuristics. Instead, we report the gap relative to the MILP range $\delta^{\star}_{\text{range}}$, where $[\delta^{\star}_{\text{range}}] = \%$, defined as

$$\delta^{\star}_{\text{range}} = \frac{|z^{\star} - b^{\text{MILP}}|}{|z^{\text{MILP}} - b^{\text{MILP}}|} \cdot 100, \tag{5.28}$$

where $z^{\star}$ denotes the objective value of the respective solution. Similar to the relative gap $\delta^{\star}$, this metric indicates how close a solution is to the optimum. However, it is not sensitive to objective values around zero. Instead, it compares the absolute gap of the examined algorithm to the best MILP bound b^{MILP} with the absolute gap of the MILP solution. A value of approximately 100 % therefore indicates that the examined algorithm achieves a solution of comparable quality to the MILP solution with respect to the best MILP bound. Values below 100 % indicate solutions that are closer to the best MILP bound than the MILP solution itself, whereas values above 100 % quantify the additional deviation from the bound compared to the MILP solution. A comparison of the resulting objective values $z^{\star}$, corresponding to the cumulative total electricity costs incurred by the community to satisfy the demand of all included households over the planning horizon, is presented in Figure 5.4.

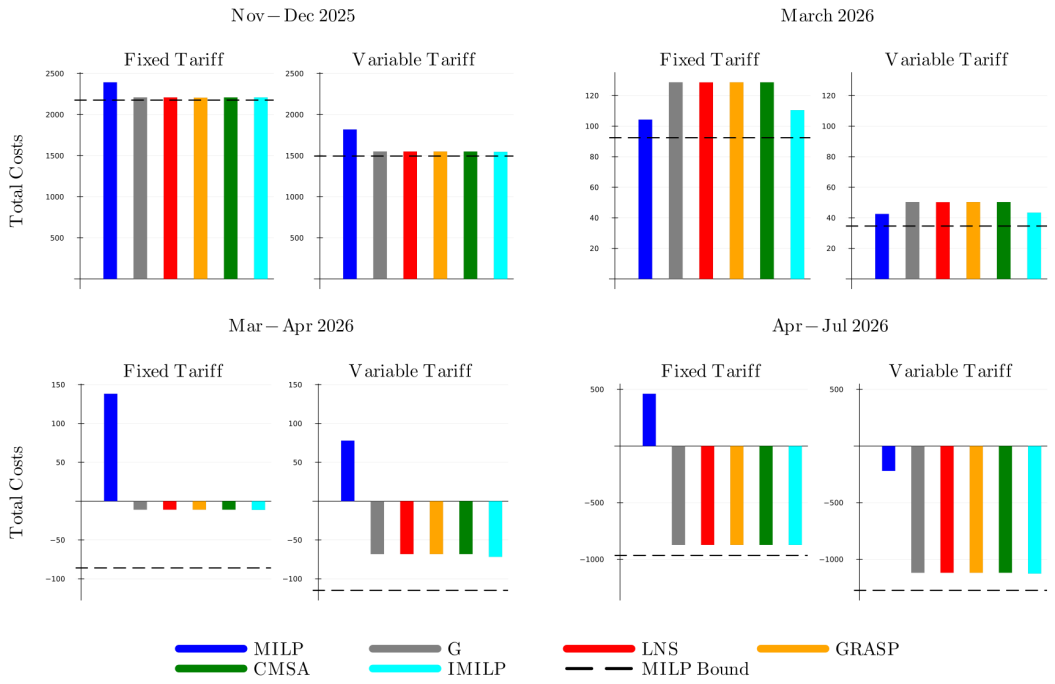


Figure 5.3: Comparison of the MILP objective value z^{MILP} and the median objective value $\tilde{z}^{\star}$, over the 30 independent runs of each heuristic approach on the Campus Domus benchmark instances. The objective value represents the cumulative total electricity cost incurred by the energy community over the planning horizon in €. The dashed line indicates the MILP bound b^{MILP}, which provides a lower bound on the optimal objective value and thus serves as a benchmark for assessing the quality of the obtained solutions.

Overall, only marginal improvements over the greedy construction heuristic were observed across the benchmark set. Furthermore, because all households are subject to the same pricing model, the optimization problem is considerably simplified for instances with a fixed electricity tariff. In this setting, the primary challenge is reduced to determining which boilers to activate based solely on consumption patterns and available PV generation. Consequently, the greedy construction heuristic is highly dependent on the tie-breaking mechanism used to select the next activation, as it considers only activations within the predefined activation window and does not account for future timesteps or potential future demand.

Next, we consider the measured wall-clock runtimes obtained from the experiments in this section, which are presented in Table 5.9. Especially in practical applications of the presented algorithms, runtime performance is a crucial aspect, as systems incorporating these algorithms may not have the computational budget to execute optimization procedures for several hours, but instead require solutions within seconds.

The MILP and IMILP approaches did not reach the global optimum within their respective

Table 5.9: Comparison of the median wall-clock runtimes $\tilde{t}^{\star}$ on the Campus Domus benchmark instances. The *Price* column denotes the fixed (F) and the variable (V) pricing model defined in Table 5.5. The median runtime is computed over 30 independent runs per algorithm. For G, the heuristic runtime is reported first, while the total runtime including solving the restricted MILP is given in parentheses.

Instance		**Gurobi**			**HiGHS**	
Timespan	Price	$\tilde{t}^{\text{MILP}}$ [s]	$\tilde{t}^{\text{LP}}$ [s]	$\tilde{t}^{\text{G}}$ [s]	$\tilde{t}^{\text{LP}}$ [s]	$\tilde{t}^{\text{G}}$ [s]
NovDez2025	F	3,611.73	28.87	0.25 (15.37)	70.85	0.25 (36.29)
NovDez2025	V	3,622.25	27.60	0.28 (16.35)	60.32	0.28 (39.43)
Mar2026	F	3,600.96	2.71	0.04 (1.31)	7.56	0.04 (3.22)
Mar2026	V	3,600.89	2.89	0.04 (1.19)	9.09	0.04 (3.20)
MarApr2026	F	3,603.58	13.43	0.10 (5.51)	26.16	0.10 (13.16)
MarApr2026	V	3,603.60	13.68	0.11 (5.56)	62.38	0.10 (13.29)
AprJul2026	F	3,619.80	42.16	0.28 (20.88)	99.08	0.27 (51.23)
AprJul2026	V	3,618.11	43.82	0.28 (19.13)	112.13	0.28 (51.85)

time limits and therefore terminated after their predefined time limits of 3,600 s and 1,000 s, respectively. Likewise, the metaheuristic approaches (LNS, GRASP, and CMSA) were executed until their specified runtime limit of 1,000 s was reached. In this context, it should be noted that some instances in this benchmark are substantially smaller than others. Consequently, the algorithms were able to perform a larger number of iterations within the same runtime limit. Since, all these algorithms were executed until their respective time limit is reached, the are omitted from Table 5.9.

Consequently, the runtime comparison focuses on the LP formulation and the greedy construction heuristic (G). For the latter, the runtime is divided into two distinct components. First, the heuristic runtime measures the time required solely to execute the Julia implementation of the heuristic and compute a feasible heater control sequence. Although this sequence could, in principle, already be applied in a real-world setting, it does not directly provide the exact objective value according to the IESopt model. It is important to note that the greedy construction heuristic itself is highly computationally efficient. This can primarily be attributed to the limited activation window size $W^{\text{size}} = 96$. Moreover, each boiler activation typically advances the activation horizon by several time steps, allowing the complete planning horizon to be traversed quickly. Consequently, the heuristic runtime remains short even for the larger instances.

In contrast, computing the objective value of a fixed boiler control sequence through the MILP formulation has a runtime comparable to solving the full LP formulation. Although the actual optimization process is relatively short due to the majority of decision variables already being fixed, the initialization of the model, variable fixing, and preprocessing steps introduce a considerable computational overhead.

Finally, we discuss some practical implications of the obtained results. The objective value of the HoWaFlex problem represents the total expected costs associated with heating and supplying DHW to a community over a given planning horizon. Since the hot water demand is assumed to be known for every timestep, the presented results should be interpreted with some caution, as such perfect foresight is generally not achievable in practice.

Nevertheless, conclusions can be drawn regarding the comparison between the variable and fixed pricing models. Considering the objective values $z^\star$ reported in Table 5.7, the variable pricing model achieves substantially lower costs than the fixed tariff under the assumption of perfect foresight. However, it should be noted that the reported objective values account only for energy costs and exclude additional external fees. Furthermore, although the Mar2026 instances cover a substantially shorter planning horizon, the expected total costs of the MarApr2026 and AprJul2026 instances are considerably lower. This can be attributed to the substantially higher PV generation during these periods, which even results in a net profit for some instances when fixed fees are disregarded.

Lastly, we discuss the implications of the observed runtimes and instance sizes. In practical applications, the optimization of boiler control sequences would typically not be performed over an excessively long planning horizon. Instead, a shorter horizon of several days could be optimized, with the optimization repeated whenever new information becomes available. Such information may include sudden changes in weather conditions, updated electricity prices from the spot market, or changes in household demand profiles, for example, when a resident takes a shower and thereby alters the expected DHW consumption.

For sufficiently small instances, the results presented in Table 5.8 suggest that solving the original MILP formulation provides the highest solution quality. However, depending on the application requirements, even computational times of several seconds may be unacceptable. In such cases, the greedy construction heuristic (G) represents a viable alternative, as it can generate high-quality solutions within substantially shorter runtimes, as demonstrated in Table 5.9.

Finally, it should be considered that modern boilers are capable of relatively frequent switching operations. Therefore, reducing the timestep duration, for example to one-minute intervals, could enable a more fine-grained utilization of available PV generation. However, decreasing the timestep size would also substantially increase the resulting problem complexity, potentially making heuristic approaches again increasingly attractive compared to exact optimization methods.

5.5.2 Generic Communities

The instances evaluated in this section are designed to represent public housing communities across different cities in Austria. They incorporate diverse household configurations and electricity tariffs and are designed to challenge the presented algorithms by capturing the various factors and characteristics that can arise in the HoWaFlex problem.

Table 5.10: Comparison of the IESopt baseline and the greedy heuristic (G) over the Generic Communities benchmark instances. The objective value is denoted by $z^\star$, the relative gap is denoted by $\delta^\star$ and computed according to Equation 5.27, and the best bound reported by the MILP solver is denoted by b^{MILP}. The median of a metric across the 30 runs conducted for the greedy construction heuristic (G) is denoted by $\tilde{\cdot}$. Gurobi was used as optimization solver for the presented results.

Instance	**IESopt**				**Greedy**	
	z^{LP}	b^{MILP}	z^{MILP}	δ^{MILP}	$\tilde{z}^{\text{G}}$	$\tilde{\delta}^{\text{G}}$
	[€]	[€]	[€]	[%]	[€]	[%]
Bregenz	9577.6	9634.3	11651.6	17.3	**9745.1**	1.1
Deutsch-Wagram	11349.1	11417.0	13480.0	15.3	**11545.9**	1.1
Eisenstadt	10603.5	10668.2	12593.0	15.3	**10806.6**	1.3
Graz	9435.6	9494.2	11484.0	17.3	**9624.9**	1.4
Innsbruck	10963.3	11029.3	12869.9	14.3	**11154.3**	1.1
Klagenfurt	10967.8	11030.7	13279.6	16.9	**11167.6**	1.2
Linz	9631.4	9689.5	11249.4	13.9	**9815.7**	1.3
Salzburg	11057.2	11120.5	13094.8	15.1	**11263.0**	1.3
St. Pölten	11584.4	11651.9	13681.9	14.8	**11775.8**	1.0
Wien	10795.3	10861.4	12999.3	16.4	**10983.8**	1.1

We begin again by comparing the basic greedy construction heuristic with the solution of the IESopt model. The resulting median objective values over the 30 independent runs are presented in Table 5.10. Since solving the models with the HiGHS solver again failed to produce feasible solutions within the specified time limit of 3,600 s, the corresponding results are omitted from the table.

The reported columns in Table 5.10 are identical to those in Table 5.10. We therefore refer to Section 5.5.1 for a description of the reported metrics and to Equation 5.27 for the definition of the relative gap $\delta^\star$.

As indicated by the difference between δ^{MILP} and $\tilde{\delta}^{\text{G}}$, the heuristic consistently outperforms direct optimization of the IESopt model using the MILP solver, achieving relative gaps that are more than a factor of 10 smaller across all instances.

Table 5.11 compares the different metaheuristics implemented within the proposed framework with the MILP baseline using the median relative gap $\tilde{\delta}^\star$ defined in Equation 5.27. The additional metaheuristics yield only marginal improvements over the greedy construction heuristic, if any. This indicates that, for the considered instances, most of the improvement is already achieved during the initial greedy construction, while subsequent metaheuristic refinement has only a limited impact on solution quality.

Figure 5.4 compares the resulting objective values $z^\star$, which represent the cumulative electricity costs incurred by the community to satisfy the demand of all included households over the planning horizon.

Table 5.11: Comparison of the median relative gap $\tilde{\delta}^{\star}$ according to Equation 5.27, over the 30 runs conducted for each heuristic approach on the Generic Communities benchmark instances. Gurobi was used as the optimization solver for the presented results.

Instance	**IESopt**	**Heuristic Approaches**				
	δ^{MILP}	$\tilde{\delta}^{\text{G}}$	$\tilde{\delta}^{\text{LNS}}$	$\tilde{\delta}^{\text{GRASP}}$	$\tilde{\delta}^{\text{CMSA}}$	$\tilde{\delta}^{\text{IMILP}}$
	[%]	[%]	[%]	[%]	[%]	[%]
Bregenz	17.3	**1.14**	1.14	1.14	1.14	1.14
Deutsch-Wagram	15.3	1.12	1.12	**1.11**	**1.11**	1.12
Eisenstadt	15.3	**1.28**	1.28	1.28	1.28	1.28
Graz	17.3	1.36	1.36	**1.35**	1.36	1.36
Innsbruck	14.3	**1.12**	1.12	1.12	1.12	1.12
Klagenfurt	16.9	1.23	1.23	**1.22**	1.23	1.23
Linz	13.9	1.29	**1.28**	1.29	1.29	1.29
Salzburg	15.1	**1.26**	1.26	1.26	1.26	1.26
St. Pölten	14.8	**1.05**	1.05	1.05	1.05	1.05
Wien	16.4	**1.11**	1.11	1.11	1.11	1.11

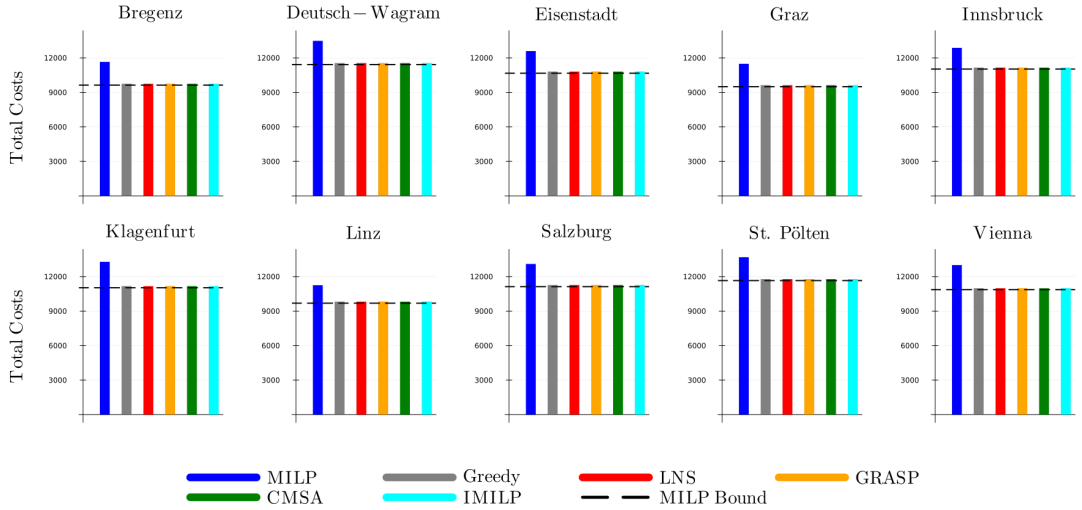


Figure 5.4: Comparison of the MILP objective value z^{MILP} and the median objective value $\tilde{z}^{\star}$, over the 30 independent runs of each heuristic approach on the Generic Communities benchmark instances. The objective value represents the cumulative total electricity cost incurred by the energy community over the planning horizon in euro. The dashed line indicates the MILP bound b^{MILP}, which provides a lower bound on the optimal objective value and thus serves as a benchmark for assessing the quality of the obtained solutions.

Lastly, in Table 5.12 we compare the measured wall-clock runtimes obtained from the experiments conducted in this section. Similarly to the benchmark presented in Section 5.5.1 the MILP and IMILP runs were again unable to obtain the optimal

Table 5.12: Comparison of the median wall-clock runtimes $\tilde{t}^{\star}$ on the Generic Communities benchmark instances. The median runtime $\tilde{t}^{\star}$ is computed over 30 independent runs per algorithm. For G, the heuristic runtime is reported first, while the total runtime including solving the restricted MILP is given in parentheses.

Instance	**Gurobi**			**HiGHS**	
	t^{MILP} [s]	$\tilde{t}^{\text{LP}}$ [s]	$\tilde{t}^{\text{G}}$ [s]	$\tilde{t}^{\text{LP}}$ [s]	$\tilde{t}^{\text{G}}$ [s]
Bregenz	3,612.03	25.20	0.65 (20.69)	72.92	0.65 (60.01)
Deutsch-Wagram	3,610.44	24.75	0.65 (19.91)	74.33	0.65 (58.95)
Eisenstadt	3,648.61	25.92	0.65 (22.68)	71.78	0.65 (59.41)
Graz	3,613.59	25.24	0.65 (22.09)	72.22	0.65 (60.98)
Innsbruck	3,651.66	24.68	0.64 (18.51)	75.61	0.64 (58.01)
Klagenfurt	3,652.07	27.19	0.66 (20.64)	73.25	0.66 (60.86)
Linz	3,617.13	26.76	0.66 (21.98)	71.89	0.67 (60.62)
Salzburg	3,612.70	24.77	0.66 (18.32)	70.59	0.67 (58.32)
St.P̃ölten	3,608.94	24.90	0.65 (21.40)	68.87	0.65 (58.57)
Wien	3,610.24	24.79	0.65 (19.65)	71.32	0.65 (59.03)

solution within their respective time limits of 3,600 s and 1,000 s. Furthermore, also the metaheuristic approaches LNS, GRASP, and CMSA algorithms were executed for their set time limit of 1,000 s.

CHAPTER 6

Case Study: BESS

From homeowners determining the optimal battery capacity and peak PV installation size to operators of large industrial facilities coordinating wind farms and multiple production sites, the challenge of determining the optimal operation of battery storage systems arises across a broad range of applications. Typical examples include deciding when a battery should be charged or discharged, or assessing whether an increase in installed battery capacity would result in additional cost savings by better complementing an existing PV system. In general, these scenarios can be formulated as the Battery Energy Storage System (BESS) problem presented in this chapter. A schematic representation of the core components and their respective energy flows is shown in Figure 6.1.

As the name suggests, the objective of the problem is to optimize the operation of a BESS. In this context, optimization refers to minimizing the total operating costs over a predefined planning horizon. The model takes as input an electricity generation profile, an electricity demand profile, and a BESS that can be charged or discharged at each timestep. By varying these inputs, different operational scenarios and investment decisions can be evaluated with respect to their expected cost savings.

Since planning horizons for such systems often span multiple years, uncertainty in the underlying data becomes an important factor that must be taken into account. Furthermore, battery degradation becomes increasingly significant over extended operating periods. To capture these effects, the proposed problem incorporates degradation mechanisms that account for both calendar aging and cycle aging, thereby reflecting degradation caused by the passage of time as well as cumulative battery usage.

While the HoWaFlex problem presented in Chapter 5 focuses on a more specific setting, the BESS problem presented here is more broadly applicable. This is particularly relevant because BESS are often considered as components of larger installations rather than being optimized in isolation, with additional external influences affecting their operation. Consequently, the problem remains a valuable case study for evaluating the proposed

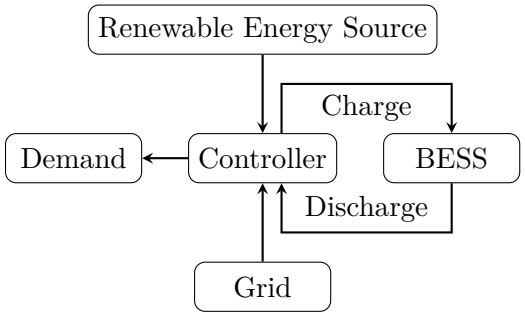


Figure 6.1: Schematic representation of the core components and energy flows of the BESS optimization problem.

heuristic framework. Although Section 6.5 demonstrates that IESopt, in combination with conventional optimization solvers, already achieves strong performance on this problem, the concepts presented here can serve as inspiration for developing future heuristics in related but more complex settings.

6.1 Problem Description

The problem is defined over a fixed time horizon, which is discretized into individual timesteps $t = 1, \ldots, T$, with the total number of timesteps denoted by T. The duration of a single timestep is given by Δ, expressed as a fraction of one hour.

The problem concerns the operation of a battery energy storage system (BESS), which can be charged or discharged in each timestep. Charging is defined as the inflow p_t^{in}, where $[p_t^{\text{in}}] = \text{kW}$, during timestep t, while discharging is defined as the outflow p_t^{out}, where $[p_t^{\text{out}}] = \text{kW}$. The energy stored in the battery at timestep t is denoted by e_t, where $[e_t] = \text{kWh}$. Both flows are subject to their respective maximum bounds P^{in} and P^{out}, where $[P^{\text{in}}] = [P^{\text{out}}] = \text{kW}$, and are constrained to be non-negative. Charging and discharging are associated with conversion losses, represented by the efficiencies η^{in} and η^{out}. Since the flow variables are defined prior to the application of losses, the state of charge e_t evolves according to

$$e_{t+1} = e_t + p_t^{\text{in}} \cdot \eta^{\text{in}} \cdot \Delta - p_t^{\text{out}} \cdot \Delta. \tag{6.1}$$

In practice, a battery cannot charge and discharge simultaneously. While this restriction can be imposed via a mutual exclusivity constraint between the two flow variables, its inclusion increases the computational complexity of the problem. Without this restriction, all decision variables remain continuous within their bounds, resulting in an LP. The introduction of mutual exclusivity requires binary decision variables, thereby transforming the model into a MILP.

While simultaneous charging and discharging may appear suboptimal at first glance, as both operations incur conversion losses and their simultaneous application therefore

effectively dissipates energy, such behavior can nevertheless be economically rational under certain market conditions. In particular, electricity markets with a high share of renewable generation (e.g., PV and wind) may exhibit negative spot prices. In these situations, deliberately dissipating energy can be profitable, as consuming or storing electricity effectively generates revenue rather than incurring a cost. Furthermore, the optimal decision between charging and discharging cannot be determined solely from the current electricity price. As for example, discharging may still be optimal during a timestep with a negative price if the stored energy was originally acquired at an even lower price. Consequently, a mutual exclusivity constraint between charging and discharging is, in principle, at least required for all timesteps with negative electricity prices.

Nevertheless, the proposed formulation enforces mutual exclusivity for all timesteps. The primary motivation is to preserve the extensibility of the model. Additional constraints or future model extensions may introduce situations in which simultaneous charging and discharging would again become economically attractive. By enforcing mutual exclusivity unconditionally, such physically unrealistic behavior is prevented independently of the specific problem setting.

The battery storage is characterized by a minimum and maximum energy capacity, denoted by $E^{\min}$ and $E^{\max}$, respectively, with $[E^{\min}] = [E^{\max}] = \text{kWh}$. The initial energy is given by E^{init}, where $[E^{\text{init}}] = \text{kWh}$. It is typically chosen as the midpoint between $E^{\min}$ and $E^{\max}$, corresponding to the standard shipping state of charge of batteries. In contrast to the HoWaFlex problem introduced in Chapter 5, the cyclic constraint in the BESS problem requires the final energy to equal E^{init}. Thus, the battery must end the planning horizon at exactly the same energy level at which it started, rather than merely requiring the final energy to be at least E^{init}. This is motivated by the assumption that the BESS continues to operate beyond the planning horizon or, alternatively, is resold at the end of the planning horizon at the same state of charge at which it was purchased.

Furthermore, two degradation mechanisms affect the maximum usable capacity over time. Calendar degradation results from chemical processes that occur naturally as the battery ages, even when it is not being actively used. In contrast, cyclic degradation is caused by battery operation and is influenced by factors such as charging and discharging cycles, current rates, and temperature. To account for these effects, we introduce the remaining maximum usable capacity $e_t^{\max}$, which evolves according to

$$e_{t+1}^{\max} = e_t^{\max} \cdot \gamma_t^{\text{cal}} - \frac{p_t^{\text{in}} + p_t^{\text{out}}}{2} \cdot \Delta \cdot \gamma^{\text{cyc}}, \tag{6.2}$$

with $e_1^{\max} = E^{\max}$, where calendar degradation γ_t^{cal} captures aging effects over time and cycle degradation γ^{cyc} represents degradation per equivalent full cycle. A full cycle corresponds to charging and subsequently discharging an amount of energy equal to the battery's full usable capacity. The division by 2 in Equation 6.2 ensures that a complete charge–discharge cycle is counted as one equivalent full cycle rather than two, thereby preventing the degradation from being counted twice.

Calendar degradation is computed using the formulation proposed by Popp [Pop14]. The constant A_0 denotes the absolute aging factor under reference conditions, SoC_0 is the reference state of charge, and SoC is the state of charge at which the degradation is evaluated. The parameter c is a cell-type-dependent scaling factor. The state of health (SoH) at timestep t is given by

$$SoH_t = A_0 \cdot e^{\frac{SoC - SoC_0}{c}} \cdot \sqrt{t \cdot \Delta \cdot 3600}. \tag{6.3}$$

The calendar degradation factor is then defined as

$$\gamma_t^{\text{cal}} = \frac{SoH_t}{SoH_{t-1}}, \tag{6.4}$$

with $SoH_0 = 1$.

In principle, calendar degradation could be computed dynamically by replacing SoC with e_t in Equation 6.3 and evaluating γ_t^{cal} accordingly. However, this is not compatible with the underlying MILP formulation. The model simultaneously determines the battery state of charge e_t, and the dynamic capacity limit $e_t^{\max}$, both of which being decision variables. Computing calendar degradation dynamically would therefore introduce nonlinearities. In particular, as shown in Equation 6.2, the capacity limit is updated by multiplying the previous capacity limit with the calendar degradation factor. If γ_t^{cal} depends on e_t, this update would involve products of decision variables, violating the linearity requirements of MILPs.

To circumvent this issue, it is common practice to precompute the calendar degradation factor assuming a fixed SoC. Since calendar degradation varies only weakly with the state of charge, the approximation error introduced by this assumption is generally considered negligible.

In contrast, cycle degradation γ^{cyc} depends on the total number of full cycles N^{cyc} that the battery can perform before reaching its expected end of life. The expected end-of-life capacity fraction is denoted by ρ^{eol}. Assuming a linear relationship between capacity loss and the number of full cycles, the cycle degradation per cycle is given by

$$\gamma^{\text{cyc}} = \frac{1 - \rho^{\text{eol}}}{N^{\text{cyc}}}. \tag{6.5}$$

Next, the considered household or commercial entity is assumed to be equipped with a renewable energy source (RES). In the instances presented in Section 6.3, this source corresponds to a PV system. However, for larger commercial or industrial consumers, other technologies, such as wind turbines or hybrid renewable energy systems, may also be relevant. The available renewable generation at each timestep is represented by the generation profile P_t^{res}, where $[P_t^{\text{res}}] = \text{kW}$. The portion of this generation that is ultimately used to satisfy demand or charge the BESS is denoted by p_t^{res}, where $[p_t^{\text{res}}] = \text{kW}$.

The electrical demand at each timestep is given by D_t, where $[D_t] = \text{kW}$. If local generation and storage are insufficient to cover demand, additional energy can be drawn from the grid. The corresponding grid import is denoted by p_t^{grid}, where $[p_t^{\text{grid}}] = \text{kW}$. Consequently, power balance at each timestep t is enforced by the following nodal balance constraint

$$D_t = p_t^{\text{grid}} + p_t^{\text{res}} - p_t^{\text{in}} + p_t^{\text{out}} \cdot \eta^{\text{out}}, \tag{6.6}$$

where η^{out} denotes the discharging efficiency introduced at the beginning of this section.

Each energy flow is associated with a cost component. Analogously to the HoWaFlex problem discussed in Chpater5, grid electricity costs consist of a fixed grid usage fee C^{grid}, where $[C^{\text{grid}}] = €/\text{kWh}$, and a supplier-dependent tariff C_t^{sup}, where $[C_t^{\text{sup}}] = €/\text{kWh}$, which may be constant or time-varying depending on the spot market price.

For renewable energy generation, several cost modeling approaches are possible, including investment costs (CAPEX), operational costs (OPEX), or feed-in revenues for surplus electricity exported to the grid. However, to isolate the operational behavior of the BESS, these effects are neglected in the base formulation. This assumption is appropriate for small- to medium-scale installations, where renewable generation can simply be curtailed in the event of excess production. For larger industrial installations, however, curtailment may not always be feasible or economically desirable, and these additional cost components may therefore need to be incorporated into the model.

Finally, the cost model of the BESS itself is introduced. The investment cost is denoted by C^{inv}, where $[C^{\text{inv}}] = €/\text{kWh}$ of installed storage capacity. In scenarios where the battery is assumed to be pre-installed, this parameter is set to $C^{\text{inv}} = 0$. Based on the investment cost and the expected number of full cycles N^{cyc} within its expected lifetime, a simplified levelized cost of storage (LCOS) is defined as

$$C^{\text{lcos}} = \frac{C^{\text{inv}}}{N^{\text{cyc}}}, \tag{6.7}$$

which is expressed in €/kWh of energy throughput.

Alternatively, degradation can be priced explicitly by introducing the degradation cost C^{deg}, where $[C^{\text{deg}}] = €/\text{kWh}$ of degraded capacity, which accounts for the reduction in usable battery capacity over time and provides a direct economic representation of battery wear. Due to the recursive computation of $e_t^{\max}$, the resulting capacity loss depends on the timing of energy throughput. For example, charging or discharging the battery earlier results in a smaller final capacity loss than performing the same action later in the planning horizon. This motivates the definition of the time-dependent degradation cost C_t^{deg}, where $[C_t^{\text{deg}}] = €/\text{kWh}$ of charged or discharged energy, which represents the degradation cost relative to energy throughput at timestep t. It is defined as

$$C_t^{\text{deg}} = \frac{1}{2} \cdot C^{\text{deg}} \cdot \gamma^{\text{cyc}} \cdot \prod_{t<t' \leq T} \gamma_{t'}^{\text{cal}}. \tag{6.8}$$

The total cost c_t incurred in each timestep is then given by

$$c_t = p_t^{\text{grid}} \cdot \Delta \cdot \left(C^{\text{grid}} + C_t^{\text{sup}} \right) + p_t^{\text{in}} \cdot \Delta \cdot C^{\text{LCOS}} + (p_t^{\text{in}} + p_t^{\text{out}}) \cdot \Delta \cdot C_t^{\text{deg}}. \tag{6.9}$$

6.1.1 MILP Formulation

Sets of Indices

$$\mathcal{T} = \{1, \dots, T\} \qquad \text{set of timesteps}$$

Model Parameters

$$\begin{aligned}
\Delta &: \text{duration of one timestep}, && [\Delta] = \text{h} \\
P^{\text{in}} &: \text{maximum charging power}, && [P^{\text{in}}] = \text{kW} \\
P^{\text{out}} &: \text{maximum discharging power}, && [P^{\text{out}}] = \text{kW} \\
\eta^{\text{in}} &: \text{charging efficiency}, && [\eta^{\text{in}}] = 1 \\
\eta^{\text{out}} &: \text{discharging efficiency}, && [\eta^{\text{out}}] = 1 \\
E^{\min} &: \text{minimum battery capacity}, && [E^{\min}] = \text{kWh} \\
E^{\max} &: \text{initial maximum battery capacity}, && [E^{\max}] = \text{kWh} \\
E^{\text{init}} &: \text{initial state of charge}, && [E^{\text{init}}] = \text{kWh} \\
D_t &: \text{electrical demand}, && [D_t] = \text{kW} \\
P_t^{\text{res}} &: \text{available renewable generation}, && [P_t^{\text{res}}] = \text{kW} \\
C^{\text{grid}} &: \text{grid usage tariff}, && [C^{\text{grid}}] = €/\text{kWh} \\
C_t^{\text{sup}} &: \text{supplier tariff}, && [C_t^{\text{sup}}] = €/\text{kWh} \\
C^{\text{lcos}} &: \text{levelized cost of storage}, && [C^{\text{lcos}}] = €/\text{kWh} \\
C^{\text{deg}} &: \text{degradation cost}, && [C^{\text{deg}}] = €/\text{kWh} \\
\gamma_t^{\text{cal}} &: \text{calendar degradation factor}, && [\gamma_t^{\text{cal}}] = 1 \\
\gamma^{\text{cyc}} &: \text{cycle degradation factor}, && [\gamma^{\text{cyc}}] = 1
\end{aligned}$$

Decision Variables

$$\begin{aligned}
p_t^{\text{in}} &\in \mathbb{R}_+ && \text{charging power}, && [p_t^{\text{in}}] = \text{kW} \\
p_t^{\text{out}} &\in \mathbb{R}_+ && \text{discharging power}, && [p_t^{\text{out}}] = \text{kW} \\
e_t &\in \mathbb{R}_+ && \text{stored energy}, && [e_t] = \text{kWh} \\
e_t^{\max} &\in \mathbb{R}_+ && \text{maximum usable battery capacity}, && [e_t^{\max}] = \text{kWh} \\
p_t^{\text{grid}} &\in \mathbb{R}_+ && \text{grid import}, && [p_t^{\text{grid}}] = \text{kW} \\
p_t^{\text{res}} &\in \mathbb{R}_+ && \text{renewable generation utilized}, && [p_t^{\text{res}}] = \text{kW} \\
x_t &\in \{0, 1\} && \text{charging/discharging mode} && [x_t] = 1
\end{aligned}$$

Objective Function and Constraints

$$\min \sum_{t \in \mathcal{T}} \left(p_t^{\text{grid}} \Delta (C^{\text{grid}} + C_t^{\text{sup}}) + p_t^{\text{in}} \Delta C^{\text{lcos}} \right) + (E^{\max} - e_{T+1}^{\max})\, C^{\text{deg}} \tag{6.10}$$

$$\text{s.t. } e_{t+1} = e_t + p_t^{\text{in}}\eta^{\text{in}}\Delta - p_t^{\text{out}}\Delta \qquad \forall t \in \mathcal{T} \quad (6.11)$$

$$e_1 = e_{T+1} = E^{\text{init}} \quad (6.12)$$

$$e_{t+1}^{\max} = e_t^{\max}\gamma_t^{\text{cal}} - \frac{p_t^{\text{in}} + p_t^{\text{out}}}{2}\Delta\gamma^{\text{cyc}} \qquad \forall t \in \mathcal{T} \quad (6.13)$$

$$e_1^{\max} = E^{\max} \quad (6.14)$$

$$E^{\min} \leq e_t \leq e_t^{\max} \qquad \forall t \in \mathcal{T} \quad (6.15)$$

$$p_t^{\text{grid}} + p_t^{\text{res}} - p_t^{\text{in}} + p_t^{\text{out}}\eta^{\text{out}} = D_t \qquad \forall t \in \mathcal{T} \quad (6.16)$$

$$0 \leq p_t^{\text{res}} \leq P_t^{\text{res}} \qquad \forall t \in \mathcal{T} \quad (6.17)$$

$$0 \leq p_t^{\text{in}} \leq x_t P^{\text{in}} \qquad \forall t \in \mathcal{T} \quad (6.18)$$

$$0 \leq p_t^{\text{out}} \leq (1 - x_t)P^{\text{out}} \qquad \forall t \in \mathcal{T} \quad (6.19)$$

$$x_t \in \{0, 1\} \qquad \forall t \in \mathcal{T} \quad (6.20)$$

(6.10) **Objective function**: minimizes the cost of grid electricity while accounting for LCOS and capacity degradation costs.

(6.11) **Stored energy**: controls the evolution of the stored energy.

(6.12) **Cyclic constraint**: ensures that the BESS starts and ends the planning horizon at the same energy level E^{init}.

(6.13),(6.14) **Maximum capacity degradation**: models the reduction of the maximum usable capacity due to calendar and cycle degradation, with the initial capacity set to $E^{\max}$.

(6.15) **Capacity bounds**: ensures that the stored energy remains between the minimum capacity $E^{\min}$ and the maximum usable capacity $e_t^{\max}$.

(6.16) **Demand satisfaction**: ensures that demand is satisfied by grid electricity, renewable generation, and power discharged from the BESS, while accounting for the power used to charge the BESS.

(6.17) **Renewable generation**: ensures that the utilized renewable generation does not exceed the available generation P_t^{res}.

(6.18),(6.19) **Charging and discharging limits**: limit the charging and discharging power and prevent the BESS from charging and discharging simultaneously.

(6.20) **Binary control**: determines whether the BESS is charging or discharging at each timestep.

6.2 Framework Implementation

Considering the overarching goal of generalizability in this thesis, one may ask whether the principles developed for the HoWaFlex problem can also be applied to the BESS

problem, or whether the differences between the two are too substantial. After evaluating several approaches, we concluded that, although a direct reuse of the greedy construction algorithm is possible, it is not advisable.

The main difference between the two problems is that, in the HoWaFlex problem, the demand is known in advance. This allows the construction of a sweep algorithm that processes the planning horizon chronologically. In the BESS problem, however, we not only have to decide at which timesteps the battery should be charged, but also at which timesteps it should be discharged. We initially tested a two-step approach in which we first selected a discharge timestep, discharged as much energy as the demand allowed, and then applied a charging algorithm similar to the one developed for the HoWaFlex problem. Selecting multiple discharge timesteps simultaneously is not advantageous because the feasibility of these discharge decisions has to be verified, namely whether a corresponding charging sequence can still be found. Furthermore, the effort required to either ensure feasibility or to revert a set of decisions after reaching an infeasible solution is typically larger than simply processing the discharge timesteps sequentially.

From a computational complexity perspective, the main challenge of this approach does not lie in the charging decisions themselves. In each sweep, only a relatively small number of charging timesteps must be determined, making this step computationally inexpensive. Instead, the primary computational effort arises from updating the current state of charge e_t, the dynamic upper bound $e_t^{\max}$, and, in this case, a newly introduced dynamic lower bound $e_t^{\min}$, which would be required for that approach when selecting future discharge timesteps. Note that, in principle, $e_t^{\max}$ would also need to be updated for all future timesteps to account for battery degradation, however, this aspect is addressed later in Section 6.2.2.

In the HoWaFlex problem, stored energy degrades over time, which constrains the evolution of both $e_{h,t}$ and $e_{h,t}^{\max}$. More specifically, each preceding timestep can potentially accommodate a slightly higher charging level, meaning that, when backpropagating from timestep t, the resulting value of $e_{h,t}^{\max}$ exceeds the hard upper bound $E_h^{\max}$ relatively quickly. Conversely, supplying the demand solely from the energy stored in the tank causes $e_{h,t}$ to fall below the minimum capacity $e_{h,t}^{\min}$ relatively quickly, thereby terminating the update process.

In contrast, although the maximum battery capacity $e_t^{\max}$ decreases over time in the BESS problem due to degradation, the stored energy e_t itself is assumed to remain unaffected by degradation. This modeling assumption was made by the project partners, as the BESS is considered a short-term storage system that is typically operated for approximately one full cycle per day. Consequently, energy is generally not stored for extended periods, making degradation of the stored energy negligible under the expected operating conditions. For long-term storage applications, where energy may remain stored for substantially longer periods, this effect would need to be explicitly considered. Unfortunately, this means that updates to these variables must be propagated across the entire planning horizon. However, since all affected values are essentially shifted by the

same amount, this does not substantially increase the computational complexity. Instead, it merely requires a different update procedure, which is presented in Section 6.2.2.

6.2.1 Implementation of the Generic Structures

We define the following problem-specific implementations of the generic structures introduced in Section 4.1.1.

`BESSInstance` Specific implementation of `GenericInstance`. This structure stores the *Model Parameters* specified in Section 6.1.1 and additionally precomputes the degradation cost relative to energy throughput C_t^{deg} introduced in Equation 6.8. The precomputed values are used by the algorithm presented in Section 6.2.2 as part of the priority calculation.

`BESSSolution` Specific implementation of `GenericSolution`. This structure stores the *Decision Variables* defined in Section 6.1.1.

`BESSConfiguration` Specific implementation of `GenericConfiguration`. This structure stores the following hyperparameters, which are used to control the algorithm presented in Section 6.2.2:

- W^{radius} Defines the size of the temporal window used to select charging candidates for the currently considered discharge candidate.
- $\pi^{\text{threshold}}$ Defines the minimum priority value required for accepting an energy transfer. Setting this parameter to a value greater than 0.0 allows low-profit energy transfers to be disregarded.
- k^{rcl} Defines the maximum size of the RCL used to introduce randomization into the heuristic.
- $M^{\text{flow}} \in \{\texttt{LP}, \texttt{HEU}\}$ Defines the method used to compute the energy flows.
- $M^{\text{solver}} \in \{\texttt{fix}, \texttt{start}\}$ Specifies whether the MILP model during the `solve` operation is restricted by fixing the selected decisions (`fix`) or whether these decisions are provided as initial values (`start`) when computing the objective value.

`BESSState` Specific implementation of `GenericState`. This structure stores an iteration counter tracking the current execution of `generate_solution`. This enables the metaheuristics presented in Section 4.2 to avoid randomization during their first execution, ensuring that they initially start from the deterministic greedy solution before introducing randomized decisions.

`BESSProblem` Specific implementation of `StatefulProblem`.

6.2.2 Implementation of `generate_solution`

As already discussed in Section 6.1, the BESS problem fundamentally differs from the HoWaFlex problem presented in Chapter 5. Rather than satisfying a predefined demand profile, the objective is to construct a battery charging and discharging schedule that minimizes the overall operating costs compared to directly supplying the demand from the grid.

The cyclic constraint **??** requires the BESS to store the same absolute amount of energy E^{init} at the end of the planning horizon as at its beginning. Consequently, every charging operation must eventually be matched by a corresponding discharging operation, and vice versa, such that the battery returns to its original state of charge. This observation suggests that charging and discharging decisions should not be treated independently but rather as two components of a single operation. We therefore model the problem in terms of energy transfers, where each transfer represents the movement of a quantity of energy between a charging timestep and a discharging timestep. This perspective forms the basis of the heuristic presented in this section.

Each energy transfer simultaneously selects one charging timestep and one discharging timestep and transfers the maximum feasible amount of energy between them while accounting for charging and discharging losses. By considering both timesteps jointly, each transfer is feasible by construction and preserves the cyclic constraint. Multiple energy transfers may involve the same charging or discharging timesteps, as illustrated in Figure 6.2. However, a specific pair of charging and discharging timesteps can occur at most twice, once for transferring excess renewable energy and once for transferring energy purchased from the grid. Since each transfer uses the maximum feasible amount of energy, no additional transfer between the same pair of timesteps is possible. Nevertheless, distinguishing between these two transfer types is necessary, as excess renewable energy is typically available at a different effective cost than grid electricity, which leads the heuristic to prioritize them differently.

Based on these observations, we can now motivate the greedy construction procedure. Since the objective is to minimize the total costs, the problem reduces to finding the most profitable set of energy transfers that yields the highest total cost savings while satisfying all capacity constraints. In particular, the battery state of charge must remain within its dynamic operating limits throughout the planning horizon. The algorithm constructs this set greedily by starting with an empty solution and iteratively introducing energy transfer decisions. The pseudocode of the final algorithm is presented in Algorithm 6.1.

Initially, we experimented with a two-headed priority queue that paired the currently most expensive timestep with the currently cheapest timestep. However, since these timesteps can be arbitrarily far apart, computing the maximum amount of energy that can be transferred between them becomes difficult. Because we expect the battery to be charged and discharged regularly, we therefore opted to select discharge timesteps t^{d} by iterating over all timesteps in order of their expected energy purchase price, C_t^{exp}, where

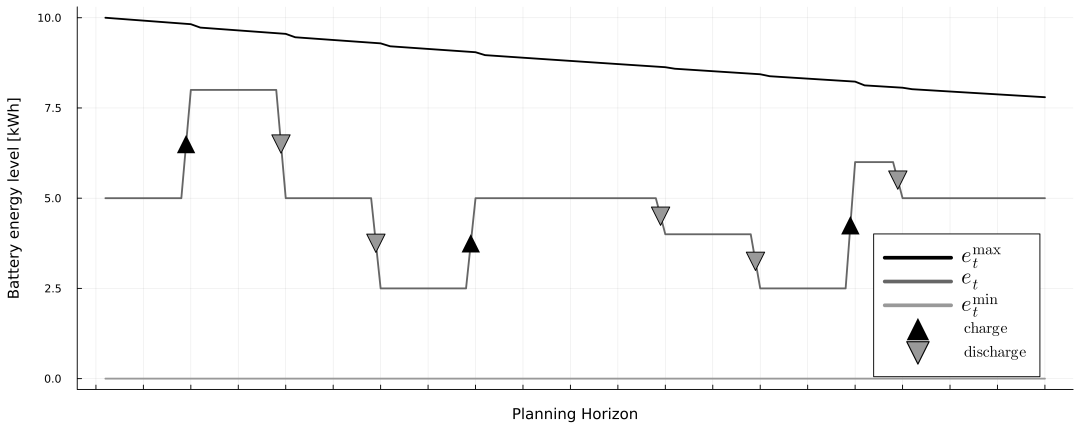


Figure 6.2: Illustration of a battery state of charge over a planning horizon. Five energy transfers are performed, with the last three sharing the same charging timestep.

$[C_t^{\exp}] = €/\text{kWh}$, defined as

$$C_t^{\exp} = \begin{cases} 0, & \text{if } D_t \leq P_t^{\text{res}}, \\ C^{\text{grid}} + C_t^{\sup}, & \text{otherwise.} \end{cases} \tag{6.21}$$

This quantity corresponds to the cost of purchasing electricity from the grid, and is zero whenever renewable generation is sufficient to satisfy demand. To further reduce the number of timesteps that can be selected for charging, we introduce, similarly to the activation window in the HoWaFlex problem presented in Chapter 5, the *charge window*, defined as the interval

$$\left[\max\left(t^{\text{d}} - W^{\text{radius}}, 1\right), \min\left(t^{\text{d}} + W^{\text{radius}}, T\right)\right]. \tag{6.22}$$

Here t^{d} denotes the discharge timestep around which the window is centered and the hyperparameter W^{radius} controls the window size.

To determine the priority $\pi_{t^{\text{c}},t^{\text{d}}}$ according to which energy transfers are selected, we first define the current energy purchase price c_t^{cur}, where $[c_t^{\text{cur}}] = €/\text{kWh}$, as

$$c_t^{\text{cur}} = \begin{cases} 0, & \text{if } D_t + p_t^{\text{res}} < P_t^{\text{res}}, \\ C^{\text{grid}} + C_t^{\sup}, & \text{otherwise.} \end{cases} \tag{6.23}$$

Comparable to $C_t^{\exp}$, this quantity depends on the amount of excess renewable generation that is still available at timestep t. The priority of an energy transfer $\pi_{t^{\text{c}},t^{\text{d}}}$, where $[\pi_{t^{\text{c}},t^{\text{d}}}] = €/\text{kWh}$, between charge timestep t^{c} and discharge timestep t^{d}, is computed as

$$\pi_{t^{\text{c}},t^{\text{d}}} = c_{t^{\text{d}}}^{\text{cur}} \cdot \eta^{\text{out}} - \left(\frac{c_{t^{\text{c}}}^{\text{cur}} + C^{\text{LCOS}} + C_{t^{\text{c}}}^{\text{deg}}}{\eta^{\text{in}}} + C_{t^{\text{d}}}^{\text{deg}}\right). \tag{6.24}$$

Algorithm 6.1: Implementation of `generate_solution` for the BESS problem.

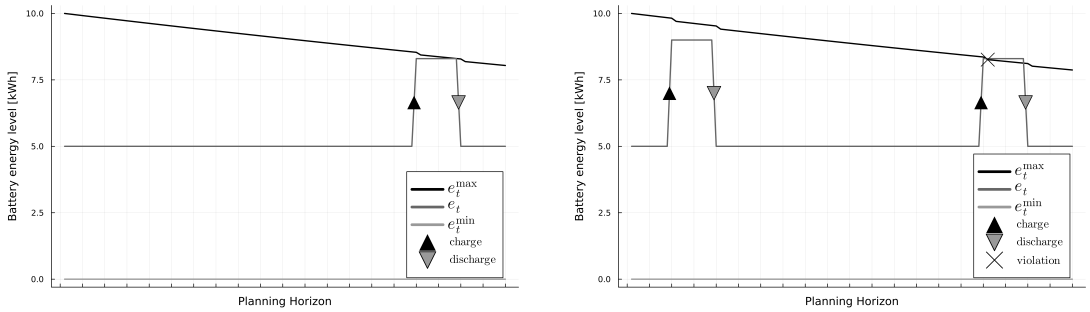
Input: `BESSProblem` p, restricted candidate list size k^{rcl}, charge window radius W^{radius}, priority threshold $\pi^{\text{threshold}}$ energy flow method M^{flow}

Output: `BESSSolution` s.

Initialize an empty `BESSSolution` s;
Set discharge candidates $D \leftarrow \mathcal{T}$;
Sort D in descending order of $C^{\text{exp}}_{t^{\text{d}}}$;
while *D is not empty* **do**
    Select and remove a timestep t^{d} uniformly at random from the first $\min(k^{\text{rcl}}, |D|)$ elements of D;
    Set charge window as $t^{\text{c}}_{\text{start}} \leftarrow \max(t^{\text{d}} - W^{\text{radius}}, 1)$ and $t^{\text{c}}_{\text{end}} \leftarrow \min(t^{\text{d}} + W^{\text{radius}}, T)$;
    Set charge candidates $C \leftarrow \{t^{\text{c}} \mid t^{\text{c}}_{\text{start}} \leq t^{\text{c}} \leq t^{\text{c}}_{\text{end}},\ p^{\text{out}}_{t^{\text{c}}} = 0,\ \pi_{t^{\text{c}},t^{\text{d}}} > \pi^{\text{threshold}}\}$;
    Sort C in descending order of $\pi_{t^{\text{c}},t^{\text{d}}}$;
    while *C is not empty* **do**
        Select and remove a timestep t^{c} uniformly at random from the first $\min(k^{\text{rcl}}, |C|)$ elements of C;
        Update s by transferring the maximum feasible amount of energy from t^{c} to t^{d};
        Remove t^{c} from D;
    end
end
if $M^{\text{flow}} =$ *HEU* **then**
    Compute $e^{\max}_t$ considering cycle degradation and the flows stored in s according to Equation 6.2;
    Reduce flows in s to not violate $e_t \leq e^{\max}_t$;
end
return s

Note that charging and discharging losses must be taken into account, since these losses would be avoided if the required energy were purchased directly from the grid rather than supplied by the battery.

The transferable amount of energy between t^{c} and t^{d} is restricted by several constraints. First, the charging and discharging power limits P^{in} and P^{out} must be respected. Furthermore, the battery state of charge must remain within the hard lower bound $E^{\min}$ and the dynamic upper bound $e^{\max}_t$. If charging occurs before discharging, the dynamic upper bound $e^{\max}_t$ has to be respected for every intermediate timestep. Therefore, the



(a) Late cycle close to the maximum.

(b) Early cycle leads to violation.

Figure 6.3: Illustration of the restriction introduced by cycle degradation.

transferable energy is limited by

$$\min_{t^{\mathrm{c}}<t\leq t^{\mathrm{d}}} \{e_t^{\max} - e_t\},$$

which can be computed efficiently for all timesteps within the charge window using dynamic programming. Similarly, if discharging occurs before charging, the hard lower bound $E^{\min}$ must be satisfied throughout the enclosed interval, resulting in the limit

$$\min_{t^{\mathrm{d}}<t\leq t^{\mathrm{c}}} \left\{e_t - E^{\min}\right\}.$$

The final and most challenging restriction arises from battery degradation. In principle, degradation also affects the previously discussed capacity constraint because every charging operation reduces the available battery capacity and therefore directly decreases $e_t^{\max}$. The primary challenge, however, stems from the cumulative nature of this effect across all subsequent timesteps. Consider a future timestep t at which the battery is scheduled to be nearly fully charged. An additional charging and discharging cycle performed beforehand may degrade the battery sufficiently such that the stored energy e_t would then exceed the reduced value of $e_t^{\max}$. Consequently, every transfer decision requires identifying the future timestep that is most likely to violate the capacity constraint due to the additional degradation. This effect is illustrated in Figure 6.3.

Importantly, this critical timestep is generally not the one for which e_t is closest to $e_t^{\max}$. Since calendar degradation is applied recursively to the dynamic upper bound, the remaining capacity at a future timestep depends on the cumulative product of all calendar degradation factors γ_t^{cal} between the considered charging window and that timestep. As a result, identifying the most restrictive future timestep requires evaluating this cumulative degradation across all subsequent timesteps.

Although the additional computation of finding the critical timestep alone would increase the algorithmic complexity to a level that is no longer attractive, an even more fundamental issue arises in the context of the greedy procedure. Suppose that very high electricity prices

occur near the end of the planning horizon. The greedy algorithm may therefore decide to charge the battery close to its maximum capacity within the corresponding charge window. While this decision appears locally optimal, it can prevent any earlier charging and discharging operations because the additional degradation caused by those transfers would reduce $e_t^{\max}$ below the stored energy level at the future timestep. Consequently, all such transfers would become infeasible, even if they would otherwise yield a significant profit. As a result, the greedy algorithm can easily reach a dead end in which no further beneficial energy transfers are feasible.

To circumvent this issue, first note that capacity violations caused by cycle degradation primarily affect earlier energy transfers in the planning horizon by either blocking them entirely or reducing the amount of energy that can be transferred between the two timesteps. However, provided that the degradation costs C_t^{deg} are incorporated into the savings calculation in Equation 6.24, cycle degradation does not fundamentally alter the transfer decisions themselves.

Based on this observation, we propose to initially neglect cycle degradation and reincorporate it in a subsequent step using one of two approaches. This prevents the greedy construction heuristic from terminating prematurely because no additional energy transfers can be selected solely due to cycle-degradation-induced capacity constraints. Moreover, it significantly reduces the computational complexity. Explicitly propagating the reduction in maximum capacity $e_t^{\max}$ caused by cycle degradation and identifying the timestep that imposes the most restrictive bound on the transferable energy requires $O(T)$ time for each candidate transfer. In contrast, by temporarily neglecting cycle degradation, the feasibility check for accepting a single energy transfer requires only $O(W^{\text{radius}})$ time.

To reincorporate the constraints imposed by cycle degradation, we propose two approaches. Recall that the only binary decision variables in the underlying MILP formulation are the x_t variables, which determine whether the battery is charging or discharging at a given timestep. All remaining variables are continuous and describe the energy flows between the photovoltaic system, the electrical grid, the battery, and the demand profile. Consequently, once the XOR decisions are fixed, the remaining optimization problem reduces to a LP that can be solved efficiently using conventional LP solvers.

Our first approach, denoted by $M^{\text{flow}} = \texttt{LP}$, exploits this observation by fixing the x_t variables in the MILP according to the charge/discharge decisions obtained through the construction heuristic while neglecting cycle degradation. The resulting LP then determines the continuous decision variables while fully accounting for cycle degradation. Consequently, this approach relies on the LP to compute the continuous energy flows and therefore cannot produce a feasible solution by itself.

The second approach, denoted by $M^{\text{flow}} = \texttt{HEU}$, aims to eliminate this reliance by computing feasible energy flows directly. As before, we first determine a set of energy transfers while neglecting cycle degradation. We then recompute the capacity limits $e_t^{\max}$ induced by the resulting energy flows, while accounting for cycle degradation. Since

the selected energy transfers are at that point already fixed, the resulting reductions in maximum capacity need to be propagated only once over the planning horizon.

Finally, we reevaluate the energy transfers by sweeping through the planning horizon from beginning to end. Whenever a transfer exceeds the updated capacity limit, its transferred energy is reduced until the corresponding capacity constraint is satisfied. Because this procedure only decreases the transferred energy, it can only decrease the resulting cycle degradation. Consequently, the earlier propagated capacity limits remain valid throughout the adjustment process, ensuring that the final energy flows satisfy all capacity constraints.

6.2.3 Implementation of `next_neighbor`

Due to the fact that most variables in the BESS problem are continuous, the only meaningful way to define a local search neighborhood is over the binary decision variables, i.e., the x_t variables. Consequently, such a local search is only applicable in combination with the $M^{\text{flow}} = \texttt{LP}$ variant of the greedy heuristic, as otherwise we would again need to compute and propagate the effects of cycle degradation. However, as demonstrated by the benchmark results in Section 6.5, this variant already produces high-quality solutions. Therefore, the expected benefit of applying a local-search-based metaheuristic on top of this procedure was considered limited, making the additional implementation effort required for a meaningful `next_neighbor` operator difficult to justify. This is particularly true because, although the construction heuristic is in some cases significantly faster than solving the complete MILP, repeatedly solving the LP, as required by methods such as local search or GRASP, would quickly lead to computation times in our benchmark comparable to those of solving the MILP directly.

For this reason, we focused our analysis on comparing the two approaches to reincorporate the effects of cycle degradation rather than developing a neighborhood function to enable the application of additional metaheuristics provided by the framework. This decision illustrates that the framework does not necessarily require neighborhood based search to be effective. Instead, this case study demonstrates that a carefully designed construction heuristic alone can already produce competitive solutions. Moreover, comparing two alternative construction procedures highlights the influence that different heuristic design choices can have on both solution quality and computational performance.

6.2.4 Implementation of `solve`, `candidates` and `solution`

The only notable distinction from the implementations presented for the HoWaFlex problem in Section 5.2.4 concerns the definition of a solution component. Depending on the value of the hyperparameter M^{flow}, either all decision variables or only the x_t variables are included in a solution component. For $M^{\text{flow}} = \texttt{HEU}$, a solution component corresponds to the complete set of decision variables introduced in Section 6.1.1 for a single timestep t. In contrast, for $M^{\text{flow}} = \texttt{LP}$, only the x_t variables are considered part of a solution component.

Apart from this distinction, the specific implementations of `solve`, `candidates`, and `solution` are relatively straightforward and follow the same principles as those presented in Section 5.2.4, with the only difference being that they return a `BESSSolution` and work with different IESopt variables.

Note that when $M^{\text{flow}} = \texttt{LP}$, `solve` must be invoked after `generate_solution` to compute the corresponding power flows and obtain a feasible solution, as the solution returned by `generate_solution` is not necessarily feasible in this case.

6.3 Generation of Benchmark Instances

When constructing benchmark instances for practical optimization problems, a fundamental choice is whether to focus on challenging edge cases that stress the evaluated algorithms across a wide range of conditions or on realistic application scenarios. While the former approach enables a comprehensive assessment of algorithmic robustness, the latter provides results that are more representative of real-world applications. For the BESS problem, the algorithmic complexity primarily arises from variable electricity prices and the resulting economic incentives to dissipate energy. Consequently, stress-testing the algorithms or constructing substantially more challenging instances would require price profiles that are unlikely to occur in realistic market conditions. Given the practical orientation of this thesis, we therefore focus exclusively on instances that reflect realistic application scenarios.

Consequently, we aim to construct instances that resemble a realistic household energy management scenario as closely as possible. While PV generation data can be obtained from the work of Pfenninger and Staffell [PS16] and day-ahead electricity prices are publicly available through the European Network of Transmission System Operators for Electricity (ENTSO-E) [Eur26], obtaining realistic household electricity consumption profiles is considerably more difficult, particularly profiles that account for external factors such as weather conditions. Fortunately, an internal source provided a real household electricity consumption profile for a household in Korneuburg, covering the period from 2024 to 2025 at a measurement interval of 15 min. We therefore set $\Delta = \frac{1}{4}$. Based on this dataset, we investigate the impact of different PV system configurations to identify the most suitable overall system design.

The considered nominal PV peak capacities, denoted by $P^{\text{res}}_{\text{nom}}$, were 10 kWp, 15 kWp, and 20 kWp. As this case study considers a single household, these capacities represent realistic residential PV system sizes that can reasonably be accommodated on the roof of a typical detached house. The PV generation profiles were again obtained using the model of Pfenninger and Staffell [PS16]. The default system configuration was employed, corresponding to a south-facing installation with a tilt angle of 35°, which is assumed to provide a near-optimal annual energy yield. The system loss parameter was set to 0.1. Since the generated PV profiles are available at an hourly resolution, the energy generated during each hour was distributed uniformly across the corresponding timesteps.

Table 6.1: Parameters used to compute the calendar degradation model according to Popp [Pop14]. The constant A_0 denotes the absolute aging factor under reference conditions, SoC_0 is the reference state of charge, and c is the cell-type-dependent scaling factor. As discussed in Section 6.1, the state of charge is fixed in advance to $SoC = 50\,\%$.

Parameter	Value
A_0	1.8344×10^{-5}
SoC_0	$50\,\%$
SoC	$50\,\%$
c	17.2127

To obtain battery systems that naturally match the considered PV installations, we assume the nominal battery capacity in kWh to be equal to the installed PV capacity $P^{\text{res}}_{\text{nom}}$, where $[P^{\text{res}}_{\text{nom}}] = \text{kWp}$. Since lithium-ion batteries are typically not operated over their entire nominal capacity in practice, we assume a 10% reserve at both the upper and lower limits of the state of charge. Furthermore, residential battery systems commonly operate at C-rates below 1, implying that a full charging cycle requires more than one hour due to the limited charging power. We therefore consider C-rates of 0.5 and 0.25, corresponding to charging times of approximately two and four hours, respectively. The resulting battery parameters are computed as follows,

$$E^{\min} = 0.1 \cdot P^{\text{res}}_{\text{nom}}, \tag{6.25}$$

$$E^{\max} = 0.9 \cdot P^{\text{res}}_{\text{nom}}, \tag{6.26}$$

$$P^{\text{in}} = P^{\text{out}} = C\text{−rate} \cdot P^{\text{res}}_{\text{nom}}. \tag{6.27}$$

Furthermore, we assume a battery lifetime of $N^{\text{cyc}} = 5{,}000$ full charge-discharge cycles before replacement. The round-trip efficiency is set to $85\,\%$ and is distributed equally between charging and discharging efficiencies,

$$\eta^{\text{in}} = \eta^{\text{out}} = \sqrt{0.85}. \tag{6.28}$$

For precomputing the calendar degradation γ^{cal}_t, we use the parameter values presented in Table 6.1.

The cycle degradation cost C^{deg} is computed according to Equation 6.5, using the previously introduced lifetime of $N^{\text{cyc}} = 5{,}000$ cycles and an assumed end-of-life capacity fraction of $\rho^{\text{EoL}} = 0.8$.

Finally, we specify the cost parameters used throughout the optimization. Since the considered household is located in Korneuburg and the demand profile covers the years 2024 and 2025, the applicable grid costs C^{grid} were taken from the regulations published by the Austrian Bundeskanzleramt[1], summarized in Table 6.2.

[1] https://ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2023_II_395/BGBLA_2023_II_395.pdf
https://ris.bka.gv.at/Dokumente/BgblAuth/BGBLA_2024_II_370/BGBLA_2024_II_370.pdf

Table 6.2: Grid usage costs C^{grid} considered for the BESS benchmark instances. Since the planning horizon spans both 2024 and 2025, the corresponding annual grid usage costs are applied for each respective period.

Year	Grid cost [€/kWh]
2024	0.0577
2025	0.0820

Table 6.3: Energy tariffs considered for the BESS benchmark instances. The tariff type identifier denotes fixed (F) and variable (V) pricing models. The eFRIENDS tariff consists of a fixed surcharge in addition to the day-ahead market price C_t^{da}, whereas the Verbund tariff remains fixed over the entire planning horizon.

Type	Supplier	Energy tariff [€/kWh]
F	V-Strom-Relax-H-KW4725	0.1439
V	eFRIENDS – der bessere FLEX15	$C_t^{\text{da}} + 0.0118$

To additionally investigate the impact of a fixed electricity tariff compared with a tariff linked to day-ahead market prices, we consider the two tariffs shown in Table 6.3. The fixed tariff was provided together with the household consumption data by the internal source. The variable tariff was obtained using the E-Control Tarifkalkulator[2] on May 27, 2026, assuming an annual electricity consumption of approximately 13 MWh for a household located in Korneuburg. The reported prices include all applicable charges and fees, thereby enabling a direct comparison with the fixed tariff. Note that the variable tariff was obtained in 2026 but is applied to the years 2024 and 2025. Although this does not fully reflect the historical market conditions, the resulting discrepancy was considered negligible for the purposes of this thesis.

Lastly, since the battery is assumed not to be resold after the end of the planning horizon, an investment cost of $C^{\text{inv}} = 250\,€/\text{kWh}$ of installed storage capacity is considered, and applying the LCOS formulation given in Equation 6.7. Furthermore, degradation costs are set to zero to avoid accounting for the battery cost twice.

In summary, we consider three different PV system sizes, two battery C-rates, and two electricity pricing models, resulting in a total of twelve benchmark instances covering every possible parameter combination. Unlike the HoWaFlex problem, where the consumption profiles are generated using a randomized model, all twelve instances in this study use the same measured household consumption profile. This allows the influence of the individual system parameters to be analyzed.

[2] https://www.e-control.at/tarifkalkulator

6.4 Experimental Setting

All experiments for the BESS problem were again conducted on the TU Wien HPC cluster of the Algorithms and Complexity Group[3] using a compute node equipped with a single AMD EPYC 9554P 64-Core processor, i.e., the same hardware used for the HoWaFlex benchmarks. The algorithms were implemented in Julia 1.12.5 [BEKS17] and executed on a single thread. Furthermore, all experiments were performed using both the HiGHS 1.13.1 [HH18] and Gurobi 12.0.3 [Gur26] solvers in order to compare their runtime performance and to assess the trade-offs that can be expected in practical applications. Since both solvers benefit from parallelization, they were allowed to utilize up to eight threads.

As discussed in Section 6.2.3, no neighborhood function was implemented for the BESS problem. Consequently, no experiments were conducted using the LNS or GRASP metaheuristics. Instead, we evaluate the two variants of the greedy construction heuristic, discussed in Section 6.2.2 and distinguished by the hyperparameter M^{flow}, denoted throughout the remainder of this section as $\text{G}^{\texttt{LP}}$ and $\text{G}^{\texttt{HEU}}$, respectively. The $\text{G}^{\texttt{LP}}$ variant fixes only the XOR decisions that determine whether the battery charges or discharges at each timestep, while the corresponding power flows are obtained by solving the resulting LP. In contrast, $\text{G}^{\texttt{HEU}}$ determines all power flows heuristically by computing the resulting upper bound $e_t^{\max}$ and reducing any energy transfers that would exceed this capacity. Consequently, this variant no longer relies on solving the LP to determine the power flows, using it solely to evaluate the objective value of the constructed solution.

We furthermore conducted experiments on the LP relaxations, the MILP formulation, the CMSA meta-heuristic, and the MILP formulation initialized with the solution obtained from the construction heuristic (IMILP). As before, the runtime limit for both the HiGHS and Gurobi solvers was fixed at 3,600 s. However, this limit was never reached for this benchmark, as all instances were solved to optimality beforehand. The time limit imposed on the CMSA and IMILP meta-heuristics was set to 250 s, which roughly corresponds to the time required by the HiGHS solver to reach optimality. All runtimes were measured as wall-clock time.

For each considered algorithm and benchmark instance, including the LP and MILP baselines, we performed 30 independent runs. This allows for a statistically meaningful analysis of the results and provides a more reliable assessment of the algorithms' runtime characteristics.

The parameters of the MILP solvers were again kept largely at their default settings. The only exception was the MIP gap, which was set to 10^{-5} to reduce noise that could otherwise appear in the presented results in this section.

The charging window radius was set throughout all benchmarks to $W^{\text{radius}} = 3 \cdot \frac{24}{\Delta} = 288$, corresponding to three days. Unlike in the HoWaFlex problem investigated in Chapter 5, this value represents a radius and is therefore applied both before and after the considered

[3] https://www.ac.tuwien.ac.at/

discharging timestep, resulting in a total charging window spanning approximately six days. This setting provided a good trade-off between runtime and solution quality. Increasing W^{radius} raises the computational effort by increasing the number of candidate charging timesteps, but it also increases the flexibility of the heuristic and may therefore improve the solution quality. Furthermore, unlike in the HoWaFlex problem, the stored energy e_t does not degrade over time. Consequently, charging the battery as close as possible to the corresponding discharging timestep is less critical.

The priority threshold was set to $\pi^{\text{threshold}} = 0.0$ for all benchmark instances. This choice was made to simplify the comparison of the results presented in Section 6.5. However, tuning this parameter could potentially improve the solutions obtained by the G^{HEU} heuristic for individual instances. Increasing the threshold filters out less profitable energy transfers and thereby reduces the number of transfers that influence the computation of the maximum energy level $e_t^{\max}$ before the power flows are adjusted to satisfy the resulting capacity limits. Consequently, selecting an appropriate threshold can improve the resulting energy limits, as including additional, less profitable transfers during construction may unnecessarily reduce the computed maximum capacity. The threshold should therefore balance the inclusion of sufficiently profitable transfers with the avoidance of unnecessary transfers that may overly restrict the resulting energy limits.

Lastly, we specify the hyperparameters used for the metaheuristics. For the CMSA algorithm, the restricted candidate list size was set to $k^{\text{rcl}} = 3$. Together with $S^{\text{iter}} = 5$ and $C^{\text{lim}} = 1.05 \cdot T$, this configuration resulted in stable algorithmic performance. The maximum component age was set to $a^{\max} = 3$. Recall that, for the CMSA algorithm, a solution component corresponds to a single XOR decision. Consequently, the metaheuristic operates with $M^{\text{flow}} = \texttt{LP}$. However, rather than solving a separate LP to determine the continuous power flows, the restricted MILP solved by CMSA computes them directly.

For the IMILP benchmark, we again employ the hyperparameter $M^{\text{solver}} = \texttt{start}$. The initial solution is obtained from a run of G^{HEU} and comprises both the XOR decision variables and the corresponding power flow variables. These values are provided to the MILP solver as a MIP start, allowing the solver to use the heuristic solution as an initial incumbent while remaining free to modify any decision variable during the optimization process.

6.5 Experimental Results

As discussed in Section 6.3, the benchmark instances were designed to reflect real-world operating conditions as closely as possible while restricting the model to the aspects relevant to the considered problem. This design choice led to a rather unexpected observation, which is reflected in the results presented in Table 6.4. We refer to Section 5.5.2 for a description of the reported performance metrics and to Equation 5.27 for the definition of relative gap $\delta^\star$, as they are identical for the present evaluation.

Table 6.4: Comparison of the IESopt baseline and the G^{LP} heuristic over the BESS benchmark instances. The *Price* column denotes the fixed (F) and the variable (V) pricing model defined in Table 6.3. The Z^{UB} denotes the theoretical objective value if no optimization is performed. The median objective value is denoted by $\tilde{z}^{\star}$, the relative gap is denoted by $\tilde{\delta}^{\star}$ and computed according to Equation 5.27, and the median best bound reported by the MILP solver is denoted by $\tilde{b}^{MILP}$.

Instance				**IESopt**				**Heuristic**	
P^{res}_{nom} [kWp]	C-rate	Price	Z^{UB} [€]	$\tilde{z}^{LP}$ [€]	$\tilde{b}^{MILP}$ [€]	$\tilde{z}^{MILP}$ [€]	$\tilde{\delta}^{MILP}$ [%]	$\tilde{z}^{G^{LP}}$ [€]	$\tilde{\delta}^{G^{LP}}$ [%]
10	0.50	V	3,910.6	3,412.9	3,412.9	3,412.9	0.00	3,413.8	0.03
		F	4,448.6	4,175.4	4,175.4	4,175.4	0.00	4,175.4	0.00
	0.25	V	3,910.6	3,483.4	3,483.4	3,483.4	0.00	3,484.6	0.03
		F	4,448.6	4,175.6	4,175.6	4,175.6	0.00	4,175.6	0.00
15	0.50	V	3,766.0	3,001.1	3,001.1	3,001.1	0.00	3,003.6	0.08
		F	4,253.4	3,693.0	3,693.0	3,693.0	0.00	3,693.0	0.00
	0.25	V	3,766.0	3,067.5	3,067.5	3,067.5	0.00	3,069.1	0.05
		F	4,253.4	3,693.4	3,693.4	3,693.4	0.00	3,693.4	0.00
20	0.50	V	3,660.3	2,672.5	2,672.5	2,672.5	0.00	2,676.6	0.15
		F	4,116.5	3,283.3	3,283.3	3,283.3	0.00	3,283.3	0.00
	0.25	V	3,660.3	2,727.1	2,727.1	2,727.1	0.00	2,730.8	0.14
		F	4,116.5	3,283.4	3,283.4	3,283.4	0.00	3,283.4	0.00

The key observation is that the MILP solution is identical to the LP solution for all benchmark instances. This indicates that there is no timestep at which the electricity price is sufficiently low to make simultaneous charging and discharging of the battery profitable. While negative electricity prices do occur for the variable pricing model, they are not sufficiently low to offset the degradation and investment costs associated with additional battery cycling.

Consequently, the MILP formulation is comparatively easy to solve, as the LP relaxation at the root node already yields the optimal solution. Therefore, the MILP baselines obtains the optimal solution for every benchmark instance, meaning that the relative gaps reported in Tables 6.4 and 6.5 represent the deviation from the true optimum. Notably, the G^{LP} heuristic also identifies the optimal solution for all instances with a fixed pricing model (denoted by F), whereas the G^{HEU} heuristic performs worse on these instances.

The median relative gap for all investigated optimization methods is presented in Table 6.5. In this context, it is noteworthy that the performance of the G^{HEU} heuristic follows an interesting pattern across the different instances. Besides performing less effectively for fixed tariffs (denoted by F), its performance also depends on the C-rate of the considered battery. This behavior can be attributed to the procedure used to compute

Table 6.5: Comparison of the median relative gap $\tilde{\delta}^{\star}$, over the 30 runs conducted for each heuristic approach on the BESS benchmark instances. The optimization solver had no impact on the obtained relative gaps, with the exception of the CMSA benchmark, where the results for the respective solvers are reported separately. The *Price* column denotes the fixed (F) and the variable (V) pricing model defined in Table 6.3. The median relative gap over the 30 conducted runs of an algorithm is denoted by $\tilde{\delta}^{\star}$ and computed according to Equation 5.27.

Instance			**Relative Gap $\tilde{\delta}^{\star}$**						
$P^{\text{res}}_{\text{nom}}$	C-rate	Price	LP	MILP	G^{HEU}	G^{LP}	CMSA HiGHS	CMSA Gurobi	IMILP
[kWp]			[%]	[%]	[%]	[%]	[%]	[%]	[%]
10	0.50	V	0.00	0.00	1.54	0.03	0.02	0.02	0.00
		F	0.00	0.00	1.74	0.00	0.00	0.00	0.00
	0.25	V	0.00	0.00	0.75	0.03	0.02	0.02	0.00
		F	0.00	0.00	0.74	0.00	0.00	0.00	0.00
15	0.50	V	0.00	0.00	5.25	0.08	0.06	0.05	0.00
		F	0.00	0.00	6.69	0.00	0.00	0.00	0.00
	0.25	V	0.00	0.00	2.62	0.05	0.03	0.03	0.00
		F	0.00	0.00	3.19	0.00	0.00	0.00	0.00
20	0.50	V	0.00	0.00	9.24	0.15	0.12	0.10	0.00
		F	0.00	0.00	13.37	0.00	0.00	0.00	0.00
	0.25	V	0.00	0.00	5.00	0.14	0.10	0.08	0.00
		F	0.00	0.00	7.36	0.00	0.00	0.00	0.00

the final capacity upper bound $e^{\max}_t$. Higher renewable generation and charging power enable a larger number of energy transfers between timesteps, thereby increasing the accumulated cycle degradation. Consequently, the computed upper bound $e^{\max}_t$ becomes more restrictive, requiring larger corrections during the heuristic adjustment step and leading to larger deviations from the optimal solution. Preliminary experiments further showed that adjusting the hyperparameter $\pi^{\text{threshold}}$ does unfortunately not improve the performance of G^{HEU} for these instances.

Furthermore, the two CMSA benchmarks exhibit different performance despite using the same metaheuristic configuration, which can be attributed to the different optimization solvers employed for solving the restricted MILPs. Due to the higher computational efficiency of Gurobi, more CMSA iterations could be performed within the same runtime, resulting in a larger number of evaluated randomized greedy solutions and, consequently, a lower median relative gap.

Figure 6.4 compares the solution quality achieved by the considered algorithms across the BESS benchmark instances. For each instance and algorithm, the reported value is the median over 30 independent runs. The results are grouped by PV size, pricing model,

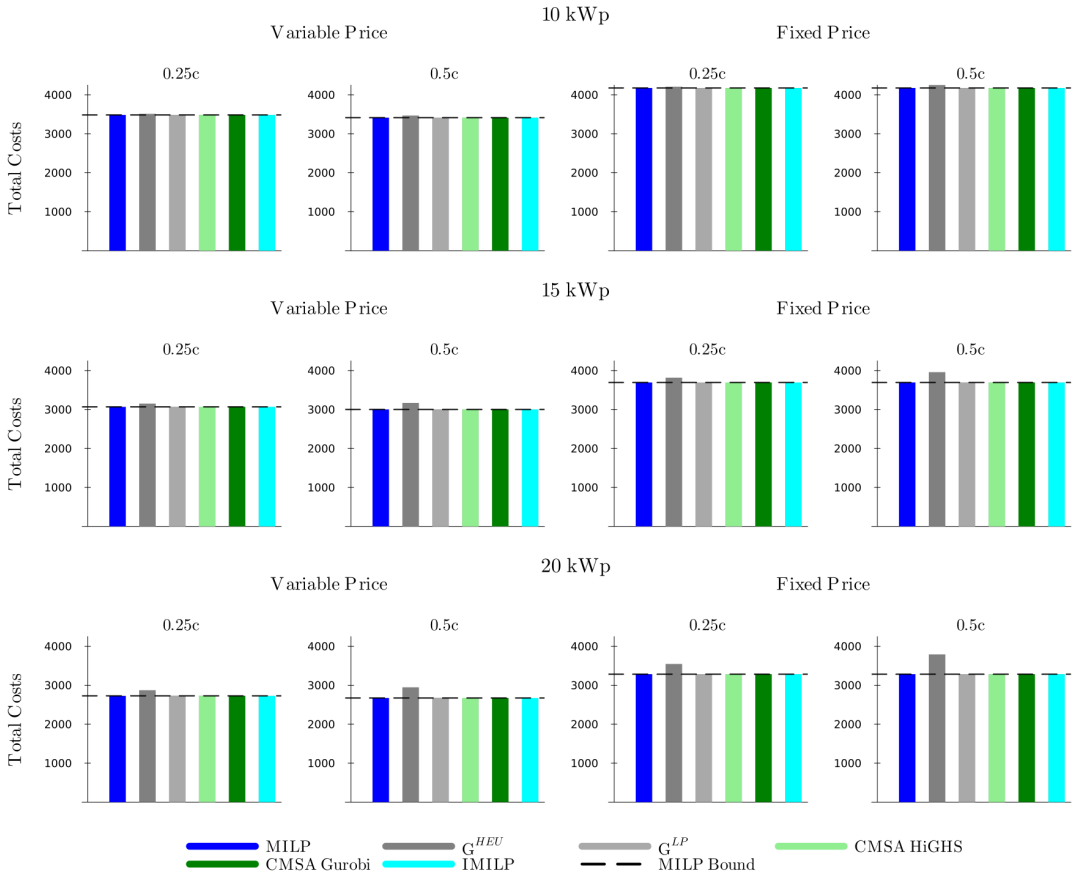


Figure 6.4: Comparison of the median objective value $\tilde{z}^{\star}$, over the 30 independent runs of each algorithm on the BESS benchmark instances. The objective value represents the cumulative total cost of operating the BESS and satisfying all electricity demands over the planning horizon, measured in €. The dashed line indicates the MILP bound b^{MILP}, which provides a lower bound on the optimal objective value and serves as a benchmark for assessing the quality of the obtained solutions.

and battery configuration, facilitating comparison between the different configurations and indicating which configuration ultimately incurs the lowest total costs over the planning horizon.

Since the heuristic approaches achieve solutions that are very close to the optimum, the comparison of the median runtimes, presented in Table 6.6, becomes particularly relevant. In particular, the G^{LP} algorithm achieves relative gaps of at most 0.15 %, which are negligible for typical use cases, while reducing the computational effort by approximately a factor of at least 7 compared to solving the MILP directly with either optimization solver. Regarding the IMILP benchmark, although it also terminates at the optimal solution, providing an initial solution does not necessarily reduce the runtime

compared to solving the MILP without initialization. Furthermore, while the CMSA metaheuristic improves upon the solution obtained by a single run of G^{LP}, it does not reach the optimum for instances where the construction heuristic fails to identify it. Consequently, the additional computational effort required by CMSA is difficult to justify for the considered benchmark instances.

Lastly, considering the differences in median runtime between the HiGHS and Gurobi optimization solvers, the results reveal an unexpected behavior. For the considered instances, the LP solver used by HiGHS appears to be different from the one used by the corresponding MILP solver. Since the LP relaxation is typically solved at the root node of the MILP branch-and-bound process, the MILP solution time would generally be expected to be at least as large as the LP solution time. This expected behavior is observed for the Gurobi runs, but not for HiGHS. One possible explanation is that the LP solver employed by HiGHS performs less preprocessing than the MILP solver and therefore solves a more difficult formulation. Interestingly, the instance with $P_{\text{nom}}^{\text{res}} = 20\,\text{kWp}$ and a C-rate of 0.5 represents a clear outlier in this behavior.

Table 6.6: Comparison of the median wall-clock runtimes. The *Price* column denotes the fixed (F) and the variable (V) pricing model defined in Table 6.3. The median runtime is computed over 30 independent runs per algorithm. For G^{HEU}, the heuristic runtime is reported first, while the total runtime including solving the restricted MILP is given in parentheses.

(a) Comparison of the median wall-clock runtimes using HiGHS as the optimization solver.

Instance			**Median Runtime $\tilde{t}^{\star}$ using HiGHS**					
$P_{\text{nom}}^{\text{res}}$ [kWp]	C-rate	Price	LP [s]	MILP [s]	G^{HEU} [s]	G^{LP} [s]	CMSA [s]	IMILP [s]
10	0.50	V	627.45	236.47	2.71 (18.80)	21.43	250.78	396.11
		F	622.54	193.93	3.15 (18.65)	21.50	251.01	204.18
	0.25	V	558.88	248.87	2.61 (18.53)	21.60	250.76	403.02
		F	542.81	157.94	3.31 (18.71)	21.35	251.74	191.18
15	0.50	V	612.17	267.09	2.06 (18.11)	21.18	250.93	441.36
		F	611.93	241.74	3.33 (18.95)	21.78	250.74	213.38
	0.25	V	540.88	264.10	2.53 (18.57)	21.70	251.69	428.15
		F	540.93	263.40	2.88 (18.32)	22.03	251.25	372.67
20	0.50	V	16.57	632.45	1.99 (17.84)	21.62	250.80	470.75
		F	15.78	920.41	2.61 (17.93)	21.17	251.61	216.43
	0.25	V	577.96	722.25	2.68 (18.53)	21.54	252.28	461.19
		F	576.91	950.61	3.54 (18.80)	21.63	251.33	207.32

(b) Comparison of the median wall-clock runtimes using Gurobi as the optimization solver.

Instance			**Median Runtime $\tilde{t}^{\star}$ using Gurobi**					
$P_{\text{nom}}^{\text{res}}$ [kWp]	C-rate	Price	LP [s]	MILP [s]	G^{HEU} [s]	G^{LP} [s]	CMSA [s]	IMILP [s]
10	0.50	V	5.58	54.07	1.56 (6.76)	7.25	250.14	57.51
		F	5.47	55.18	3.71 (7.55)	7.84	250.16	59.51
	0.25	V	5.58	52.48	1.83 (7.40)	6.66	251.73	55.59
		F	5.38	53.78	3.55 (7.39)	7.59	250.16	58.55
15	0.50	V	6.06	54.17	2.25 (6.83)	7.48	250.13	57.81
		F	5.39	54.49	3.68 (7.29)	7.38	250.15	59.01
	0.25	V	5.99	53.37	2.37 (7.37)	7.29	250.14	56.61
		F	5.63	53.70	3.54 (7.37)	7.79	250.15	57.67
20	0.50	V	5.98	54.42	2.53 (7.09)	7.25	250.13	58.03
		F	5.59	54.24	3.58 (7.32)	7.72	250.16	58.72
	0.25	V	5.98	54.11	2.80 (7.17)	7.55	250.13	57.91
		F	5.44	54.07	3.79 (7.38)	7.44	250.15	58.42

CHAPTER 7

Conclusion

As the case studies presented in Chapters 5 and 6 have shown, heuristic optimization has substantial potential to improve both solution quality and solution-generation efficiency, particularly for larger and more complex instances. Therefore, especially in applications where solutions must be generated quickly, investigating the potential of heuristic optimization can be well worth the associated development effort. The first part of the research question can consequently be answered positively.

The second part of the research question, concerning the complexity and effort required for implementation, is more nuanced. The specific heuristics developed as part of this thesis are relatively sophisticated and required a considerable amount of development effort, even for the comparatively simple greedy construction heuristics. Although some of the concepts and mechanisms developed in this thesis can be reused in future research projects, problem-specific heuristic components will generally still need to be designed and implemented specifically for each new application.

In practice, the initial construction of the heuristics was relatively straightforward. The more substantial development effort arose from improving their performance and ensuring their compatibility with the underlying IESopt models. For example, in the BESS case study presented in Chapter 6, the basic heuristic idea of selecting the most valuable energy transfers could be implemented relatively quickly. However, additional development was required to account efficiently for cycle degradation and thereby substantially reduce the overall runtime of the heuristic.

A second major challenge concerned ensuring consistency between the heuristic and the underlying IESopt model. In order to fix decision variables in the IESopt model, the resulting assignments must form part of a feasible solution. This can be challenging in some cases because IESopt is designed to provide a high-level and easy-to-use modeling interface and therefore abstracts away many of the internal relationships and constraints of the underlying optimization model. However, some of these details may still need to be

considered explicitly when developing a heuristic that directly manipulates the model's decision variables. This suggests that the development of a heuristic is likely to be more efficient if it is carried out by the same developers who construct the corresponding IESopt model, or if both are developed concurrently. Such an approach would reduce potential compatibility issues and allow the heuristic to account for relevant model-specific details from the outset.

However, given that these were the primary challenges encountered, we strongly recommend allocating at least a small portion of project resources to exploring heuristic optimization when initial experiments with an IESopt model yield poor solution quality or require substantial model simplifications. Even a preliminary heuristic implementation can provide valuable insights into the achievable solution quality and computational performance. Although such an initial heuristic may not be as highly optimized as the approaches presented in this thesis, it can help determine whether further investment in heuristic optimization is justified.

Regarding future work, possible extensions to the framework have already been discussed in Section 4.3. In particular, extending the framework with additional hybrid heuristic approaches could broaden the range of optimization strategies available also to future applications and reduce the need to develop such methods from scratch for each new project. Thereby making the initial development efforts of such methods possibly easier to justify.

Furthermore, it will be interesting to investigate the improvements that can be achieved through heuristic optimization for increasingly complex IES models. This is particularly relevant as the field of integrated energy systems continues to expand and an increasing number of smart devices and technologies are being integrated into these systems. For example, the smart boilers considered in the Campus Domus case study in Section 5.3.1 introduce additional degrees of operational flexibility and, consequently, new optimization opportunities. As the number of such technologies and their interactions increases, the resulting optimization problems are unlikely to become easier to solve, further highlighting the potential role of heuristic optimization in future energy system applications.

Overview of Generative AI Tools Used

The primary AI tool that I used during the development of this thesis was ChatGPT[1], specifically the GPT-5.6 Luna model. ChatGPT was used throughout the development of the source code to assist in identifying and resolving programming errors and to support the interpretation and lookup of software documentation. It was also used to revise the thesis text. In particular, I wrote the original paragraphs myself and subsequently used ChatGPT to improve spelling, grammar, and phrasing. All revisions generated with the assistance of ChatGPT were subsequently reviewed and revised to identify and correct potential errors in reasoning, interpretation, and content. I therefore retain full responsibility for the accuracy and content of the final text.

Another AI-based tool used to a limited extent during the literature research was Connected Papers[2]. It was primarily used to identify potentially relevant research papers and to provide inspiration for further literature searches. The majority of the literature research, however, was conducted using Google Scholar[3] and targeted keyword searches.

[1] https://www.chatgpt.com/
[2] https://www.connectedpapers.com/
[3] https://www.scholar.google.com/

List of Figures

List of Algorithms

Bibliography

[AAL+26] E. Aliyan, J. Almeida, S. Limmer, S. Ramos, and J. Soares. Adaptive large neighborhood search for optimal clustering of prosumers into energy communities. In *Machine Learning, Optimization, and Data Science*, pages 448–462, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-21477-5_30.

[AKB23] M. A. Akbay, C. B. Kalayci, and C. Blum. Application of CMSA to the electric vehicle routing problem with time windows, simultaneous pickup and deliveries, and partial vehicle charging. In *Metaheuristics*, pages 1–16, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-26504-4_1.

[BEKS17] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi:10.1137/141000671.

[BEP+25] P. Bunnak, H. Eldardiry, M. Pavičević, J. Z. Feng, and S. Galelli. Pownet 2.0: A production cost modeling framework for large-scale power systems. *Journal of Open Source Software*, 10(111):8035, 2025. doi:10.21105/joss.08035.

[BPLIL16] C. Blum, P. Pinacho, M. López-Ibáñez, and J. A. Lozano. Construct, merge, solve & adapt a new general algorithm for combinatorial optimization. *Computers & Operations Research*, 68:75–88, 2016. doi:10.1016/j.cor.2015.10.014.

[BR16] C. Blum and G. R. Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer International Publishing, 2016. doi:10.1007/978-3-319-30883-8.

[Chm25] A. Chmielewska. Characteristics of domestic hot water consumption profiles in multi-family buildings for energy modeling purposes. *Energies*, 18(17), 2025. doi:10.3390/en18174578.

[CMB+22] S. Candas, C. Muschner, S. Buchholz, R. Bramstoft, J. van Ouwerkerk, K. Hainsch, K. Löffler, S. Günther, S. Berendes, S. Nguyen, and A. Justin. Code exposed: Review of five open-source frameworks for modeling renewable energy systems. *Renewable and Sustainable Energy Reviews*, 2022. doi:10.1016/j.rser.2022.112272.

[DIN23] DIN Media. Din en iec 60379:2023-11 – methods for measuring the performance of electric storage water heaters for household purposes, 2023. Accessed: 2026-07-01. URL: https://www.dinmedia.de/en/standard/din-en-iec-60379/371546932.

[Dre57] S. E. Dreyfus. Computational aspects of dynamic programming. *Operations Research*, 5(3):409–415, 1957. doi:10.1287/opre.5.3.409.

[Eur26] European Network of Transmission System Operators for Electricity (ENTSO-E). Entso-e transparency platform – total load, 2026. Accessed: 2026-06-01. URL: https://transparency.entsoe.eu/load/total/.

[FGFJ19] D. Ferone, A. Gruler, P. Festa, and A. A. Juan. Enhancing and extending the classical GRASP framework with biased randomisation and simulation. *Journal of the Operational Research Society*, 70(8):1362–1375, 2019. doi:10.1080/01605682.2018.1494527.

[FR89] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989. doi:10.1016/0167-6377(89)90002-3.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. doi:10.5555/186897.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

[GJF+17] A. Grasas, A. A. Juan, J. Faulin, J. de Armas, and H. Ramalhinho. Biased randomization of heuristics using skewed probability distributions: A survey and some applications. *Computers & Industrial Engineering*, 110:216–228, 2017. doi:10.1016/j.cie.2017.06.019.

[GP19] M. Gendreau and J. Potvin, editors. *Handbook of Metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham, 3 edition, 2019. doi:10.1007/978-3-319-91086-4.

[Gur26] Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2026. URL: https://www.gurobi.com.

[HBH+24] L. Hellemo, E. F. Bødal, S. E. Holm, D. Pinel, and J. Straus. Energymodelsx: Flexible energy systems modelling with multiple dispatch. *Journal of Open Source Software*, 9(97):6619, 2024. doi:10.21105/joss.06619.

[HH18] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. doi:10.1007/s12532-017-0130-5.

[IBM26] IBM ILOG CPLEX Optimization Studio. Ibm ilog cplex optimizer, 2026. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984. doi:10.1007/BF02579150.

[LDDG+23] M. Lubin, O. Dowson, J. Dias Garcia, J. Huchette, B. Legat, and J. P. Vielma. Jump 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 15(3):581–589, 2023. doi:10.1007/s12532-023-00239-3.

[MdSLAVB15] F. D. Moya, L. C. P. da Silva, J. C. Lopez A., and P. Vergara B. GRASP model for smart home electrical loads scheduling. In *2015 International Energy and Sustainability Conference (IESC)*, pages 1–6, 2015. doi:10.1109/IESC.2015.7384384.

[MGBPCR22] J. Molina Gomez, M. Bennassar, E. Palacio, and S. Crespi Rotger. Low efficacy of periodical thermal shock for long-term control of legionella spp. in hot water system of hotels. *Pathogens*, 11, 01 2022. doi:10.3390/pathogens11020152.

[MH97] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997. doi:10.1016/S0305-0548(97)00031-2.

[MPB+25] R. Macdonald, F. Pecci, L. Bonaldo, J. W. Law, Y. Weng, D. Mallapragada, and J. Jenkins. MacroEnergy.jl: A large-scale multi-sector energy system framework. *ArXiv preprint*, 2025. doi:10.48550/arXiv.2510.21943.

[MRR+53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 06 1953. doi:10.1063/1.1699114.

[Pop14] H. Popp. Einfluss von vehicle-to-grid applikationen auf die alterung von lithium-ionen batterien, 12 2014. doi:10.13140/2.1.2828.1925.

[PR19] D. Pisinger and S. Ropke. *Large Neighborhood Search*, pages 99–127. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-319-91086-4_4.

[PS16] S. Pfenninger and I. Staffell. Long-term patterns of european pv output using 30 years of validated hourly reanalysis and satellite data. *Energy*, 114:1251–1265, 2016. doi:10.1016/j.energy.2016.08.060.

[RKB+23] R. M. Rosati, L. Kletzander, C. Blum, N. Musliu, and A. Schaerf. Construct, merge, solve and adapt applied to a bus driver scheduling problem with complex break constraints. In *AIxIA 2022 – Advances in Artificial Intelligence*, pages 254–267, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-27181-6_18.

[RP06] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40:455–472, 11 2006. doi:10.1287/trsc.1050.0135.

[Sha98] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/3-540-49481-2_30.

[SM24] S. Strömer and K. Maggauer. Iesopt: A modular framework for high-performance energy system optimization. In *2024 Open Source Modelling and Simulation of Energy Systems (OSMSES)*, 9 2024. doi:10.1109/OSMSES62085.2024.10668965.

[SRL23] H. de O. M. Serrano, C. Reiz, and J. B. Leite. Capacity management in smart grids using greedy randomized adaptive search procedure and tabu search. *Processes*, 11(8), 2023. doi:10.3390/pr11082464.

[Tal09] El-Ghazali Talbi. *Parallel Metaheuristics*, pages 460–534. John Wiley & Sons, 2009. doi:10.1002/9780470496916.ch6.

[YLJ25] L. Yang, P. Li, and J. Jian. Solving unit commitment problems with graph neural network based initial commitment prediction and large neighborhood search. *ArXiv preprint*, 2025. doi:10.48550/arXiv.2505.14408.