Simulation-Based Optimization of Renewable Energy Systems

Exploring simulation optimization in various energy system domains

Master Thesis

Lucas Klootwijk

TU Delft

Simulation-Based Optimization of Renewable Energy Systems

Exploring simulation optimization in various energy system domains

by

Lucas Klootwijk

to obtain the degree of Master of Science in Sustainable Energy Technology
at Delft University of Technology

To be defended publicly on Tuesday the 12^{th} of August 2025

Student number:	5351820	
Thesis committee:	Milos Cvetkovic	Chair
	Gautham Ram Chandra Mouli	Core member

Acknowledgements

This thesis was completed as a part of the MSc Sustainable Energy Technology program at Delft University of Technology and marks the end of my journey at this university. Before presenting my research, there are a few people I would like to acknowledge for their contribution to these past five years.

First, I would like to thank my thesis supervisor, Milos Cvetkovic. When I was looking for a research topic related to energy system simulation, he welcomed me warmly. His optimism and knowledge inspired me to take on this challenge. I extend my thanks to my daily supervisors, Despoina Georgiadi and Jort Groen, who supported me by reviewing my work and by helping me shape my graduation project into what it ultimately became. I wish you good luck with the continued development of The Illuminator and with your research.

Over the years, I have collaborated with numerous colleagues, to whom I would like to express my gratitude. There is, however, one colleague I would like to thank in particular: Daan Schat. From the first Master's course to the last, we took every chance to work together, as we were an excellent team. Thank you for being a great companion both inside and outside of academia.

My study period was not always easy and demanded a lot from me. The Electrotechnische Vereeniging offered me a place to recharge and to develop myself outside of education. A warm thank you goes to everyone involved in the association for providing me with (sometimes deserved) distractions from my studies.

Speaking of distractions: Erik van Weelderen. He was part of the majority of the memories I made as a student. From random creative outbursts and unplanned parties to good conversations and study sessions. Thank you for being such a good friend, for your support, and for all the amazing times we spent together.

I am thankful for my sister Christel, who patiently read through my thesis and caught many spelling mistakes I would have otherwise missed. Your ambition and everything you have achieved in this short time is inspiring. I am glad that we have grown closer over the past years.

My deepest appreciation goes to my parents. I am grateful for all the opportunities they have given me and for their everlasting support. Thank you for always believing in me, especially during challenging times.

Lucas Klootwijk
Delft, August 2025

Disclaimer: Assistance from LLMs was used in parts of this document for language and structure refinement purposes.

Abstract

The increasing complexity of renewable energy systems characterized by multiple energy carriers and local intermittent resources, calls for accurate tools for effective design, operation, and planning. This thesis investigates simulation-based optimization as a tool to support such decision-making processes.

To build a foundation for the proposed method, a background study was conducted on optimization theory in general and on simulation-based optimization with a primary focus on energy systems. Additionally, the functionality of the simulation software used in this thesis, *The Illuminator*, was explored.

Based on this foundation, a new optimization framework was developed by extending The Illuminator software and through the integration of three algorithms: Particle Swarm Optimization (PSO), Genetic Algorithm (GA), and a gradient-based algorithm (L-BFGS-B). Parallelization was implemented to increase the efficiency of the algorithms. To expand the modeling capability of The Illuminator, several new hydrogen-related component models were developed. The framework was tested across multiple domains by using three distinct scenarios: (1) a hydrogen production facility (hydrogen domain, continuous variables, system design domain), (2) a residential energy hub (electric domain, continuous variables, system operation domain), and (3) an electric vehicle charging station (electric domain, discrete variables, system planning domain).

Among the explored algorithms, Particle Swarm Optimization (PSO) proved to be the most suitable across the three presented scenarios, achieving the lowest average gaps to the best-found solutions in each case (0.107%, 0.363%, and 20.145%, respectively). Parallelization of the population-based algorithms improved the total run time by a factor of almost 5.

The results show that simulation-based optimization is a promising approach for supporting the design, operation, and planning of complex renewable energy systems.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AOI	Angle Of Incidence
APS	Announced Pledges Scenario
CO	Continuous optimization
CPU	Central Processing Unit
CSV	Comma Separated Values
DHI	Diffuse Horizontal Irradiance
DNI	Direct Normal Irradiance
EV	Electric Vehicle
GA	Genetic Algorithm
GHI	Global Horizontal Irradiance
HDPE	High-Density Polyethylene
IEA	International Energy Agency
L-BFGS-B	Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Box constraints
MH	Meta-heuristic
MO	Multi-objective
NL	Non-linear
NZE	Net Zero Emissions
PSO	Particle Swarm Optimization
PV	Photovoltaic
RES	Renewable Energy Sources
SMR	Steam Methane Reforming
SO	Single-objective
SOC	State Of Charge
SSB	Single-solution based
STC	Standard Test Conditions
STEPS	Stated Policies Scenario
TSO	Transmission System Operator
TSP	Traveling Salesman Problem
YAML	YAML Ain't Markup Language

Constants

Symbol	Definition	Value	Unit
R	characteristic gas constant	8.314	$J/mol \cdot K$
γ	specific heat ratio	1.41	-
e_{grav,H_2}	gravimetric energy density of hydrogen	120	MJ/kg
M_{H_2O}	molar mass H_2O	18.015	g/mol

1 Introduction

The Stated Policies Scenario (STEPS), developed by the International Energy Agency (IEA), anticipates a notable expansion of the worldwide renewable energy deployment [1]. The globally installed capacity for renewable energy generation is expected to grow from 4250 GW in 2024 to 9750 GW in 2030. This significant increase is a consequence of the current international initiatives and announced governmental policies to reduce emissions. To follow a pathway aligned with net-zero emissions by 2050, an additional 1750 GW increase of capacity is required before 2030 [1]. This growth is illustrated in Figure 1.1, in which the following three scenarios are compared:

- **Stated Policies Scenario (STEPS)**, which takes into account the latest market, policy, and economic status of countries;
- **Announced Pledges Scenario (APS)**, which assumes all climate targets set by governments will be met;
- **Net Zero Emissions by 2050 scenario (NZE)**, which is the scenario in which net zero emission will be reached by 2050 and the temperature increase will be limited to 1.5 °C.

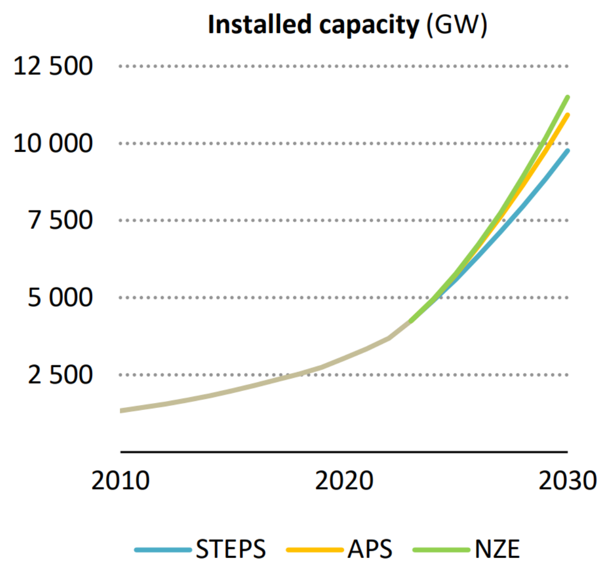


Figure 1.1: Global installed renewable energy capacity prospects for three scenarios: Stated Policies Scenario (STEPS), Announced Pledges Scenario (APS), and Net Zero Emissions by 2050 scenario (NZE) [1].

The growing use of renewable energy is beneficial for the reduction of greenhouse gases such as CO_2 and thus aid in the battle against climate change caused by humans. Additionally, renewable energy sources

(RES) are inexhaustible. This makes the implementation of these technologies a long-term solution for the global increase of energy demand. Furthermore, the transition towards a RES-dominated energy portfolio grants significant economical advantages [2].

Regardless of the aforementioned advantages of RES, their increasingly large share introduces serious challenges. One of these challenges is the intermittent energy generation pattern introduced by many RES, such as solar and wind technologies. An example of a direct cause of this pattern is the fluctuation of solar irradiance during the day, and its complete absence during nighttime. Energy shortages during times of high demand and energy surplus during times of low consumption are a consequence of this intermittency. As a result, it becomes increasingly difficult to keep the grid stable. The instability is amplified in RES-dominated grids due to a lack of physical inertia, which is the ability to resist sudden frequency deviations by temporarily storing and releasing kinetic energy through rotating masses. This is a feature that is inherent to conventional generators, such as gas turbines. In addition, the current energy infrastructure is often unable to facilitate the decentralization of RES. Unlike conventional energy generation technologies such as gas turbines and coal plants, RES are frequently distributed at lower grid levels, which are not originally designed to accommodate significant energy injections.

Aside from energy generation, the energy transition also alters consumption patterns, resulting in additional challenges for the grid. An example of this is the electrification of transport and heating technologies such as electric vehicles and heat pumps, which add substantial and unpredictable loads to the grid. When the grid cannot deliver the power required by the loads, because the demand is higher than what the grid connection can deliver, grid congestion occurs.

Together, the changes in demand and supply of energy increase the complexity of (renewable) energy systems, and the demand for alternative energy carriers. One of these alternative energy carriers is hydrogen, which is prospected to have a share of 24% of the final energy consumption in 2050 [3]. The reasons for this large share, are the many advantages of this energy carrier. For example the possibility to store it in large quantities, its transportability, and its environmentally benign nature [4]. Due to the increasing complexity of energy systems, the growing role of hydrogen, and the diversity of energy systems, there is a need for methods that aid the design, operation, and planning of robust and efficient systems.

Optimization plays a key role in addressing the aforementioned challenges. By providing the possibility to identify optimal parameters, optimization makes the design, operation, and planning of energy systems simple and accurate. In optimization models however, an entire system is described by large mathematical equations. This becomes intricate when the system is complex, affecting flexibility.

To optimize a system while still offering flexibility in the design, operation, and planning of the system, simulation optimization can be used. For the simulation part, the behavior of the individual components of the energy system are modeled. After this, the individual models are interconnected, and the system as a whole is simulated. An example of software that can be used for this is The Illuminator toolbox (v3.0.0-beta.5) [5], an open source software that can simulate energy systems by combining individual energy system components. Due to the educational purpose of The Illuminator, the software is especially useful for prototyping purposes, because of its flexibility and modularity. This becomes especially clear when comparing it to software that has a different scope, such as PyPSA and PowerFactory. Furthermore, The Illuminator is beginner friendly due to its gentle learning curve, as opposed to other energy system simulation software such as OSeMOSYS [6].

Simulation optimization requires a framework that, based on a cost function and observations from simulation, optimizes the energy system parameters. As the modeled system is composed of component models, the system can be quickly adapted by changing model parameters, or even by changing the combination of models used in the simulation. This approach offers higher flexibility, and is especially advantageous over optimization models for complex system optimization [7].

Therefore, the objective of this thesis is to develop a simulation optimization framework around the Illuminator toolbox and to investigate its application to energy system design and control.

Based on the motivation provided above, this thesis is driven by the following research question:

1. ***How can simulation optimization support the development of improved configurations of renewable energy systems?***

This question explores the feasibility and effectiveness of simulation optimization. The aim is to determine whether this approach can support and simplify decision making in energy system design, by identifying more optimal configurations. Additionally, this question encourages the investigation of the technical feasibility and effectiveness of simulation optimization as an energy system design tool. To help answer the main research question, the following sub-questions are addressed:

1.1 ***What are the main technical prerequisites and challenges for combining optimization algorithms with simulation models?***

1.2 ***Which optimization algorithms are applicable to simulation optimization in energy system modeling, and how do they differ from one another?***

1.3 ***How generalizable is a simulation optimization framework to various energy system problem domains?***

1.4 ***How can multi-carrier energy systems be effectively modeled to enable meaningful insights?***

To answer the research questions in a structured and understandable manner, this thesis is subdivided in different sections:

- *Chapter 1* introduces the thesis. This is done by outlining the current situation and prospects of the energy transition, providing the motivation for this research, establishing the objective, and defining the research question.
- *Chapter 2* expands upon the introduction by providing the reader with background knowledge of the simulator (the Illuminator), hydrogen supply chain, and optimization theory.
- *Chapter 3* presents the research methodology used in this paper, required to achieve the objectives and to answer the research question presented in the introduction.
- *Chapter 4* demonstrates and analyzes the results obtained from the implementation of the methodology.
- *Chapter 5* discusses the results of the previous chapter, and provides the reader with an answer to the research question.
- *Chapter 6* concludes the thesis by summarizing the findings and by providing recommendations for future research.

2 Background

The following chapter provides the reader with essential context that is required to comprehend the research presented in this thesis. Firstly the simulation tool used in this research is discussed. This section is followed by an introduction to the supply chain of hydrogen in energy systems. Finally, a section is dedicated optimization techniques that can be used for energy systems.

2.1. The Illuminator

The Illuminator (v3.0.0-beta.5) [5] is a simulation toolbox that can be used to model and simulate the integration of energy systems. The benefits of the Illuminator are its flexibility, scalability, and accessibility as it is open-source software [6]. To use the software, predefined models of components found in energy systems can be selected and adapted to ones liking. That is, the parameters and input data can be customized. Consequently, a configuration of an energy system can be created, after which the simulator can be instructed to simulate a scenario. The software can be considered as a general solution to conceptualize energy systems, and is an excellent tool to perform quick analysis and the prototyping of energy systems.

The configurations of the systems are defined in scenarios. The scenarios are described in YAML files in order to instruct the simulator what the scenario consists of. YAML is a file format that is used for the configuration of applications. In the case of the Illuminator, it serves this purpose as well, as it contains all the information needed to simulate a scenario. More specifically, it consists of the following parts:

- **Scenario specification:** In this part the name, the time frame of the simulation, and the time resolution are specified;
- **Models:** Here the parameters of the predefined models that are used in the system are specified.
- **Connections:** This list defines how the models are interconnected;
- **Monitor:** Here the data streams that the user wants to monitor are listed.

To show what The Illuminator can be used for, a simple simulation is provided below. This simulation scenario contains a household, a solar panel (PV), a wind turbine, and a battery. All these components are connected to each other. To simulate this scenario, a CSV file containing solar irradiation data in The Netherlands is linked to a PV model [8]. This is repeated for a CSV file containing wind speeds and the wind model. Inside these models, the generation is calculated based on their respective CSV input data. Another CSV file contains fabricated demand data and is linked to a load model. The three models, together with a battery model, are connected to a controller model, which monitors the output of the demand and generation. The controller then decides the flow in and out of the battery. The connections are displayed in Figure 2.1.

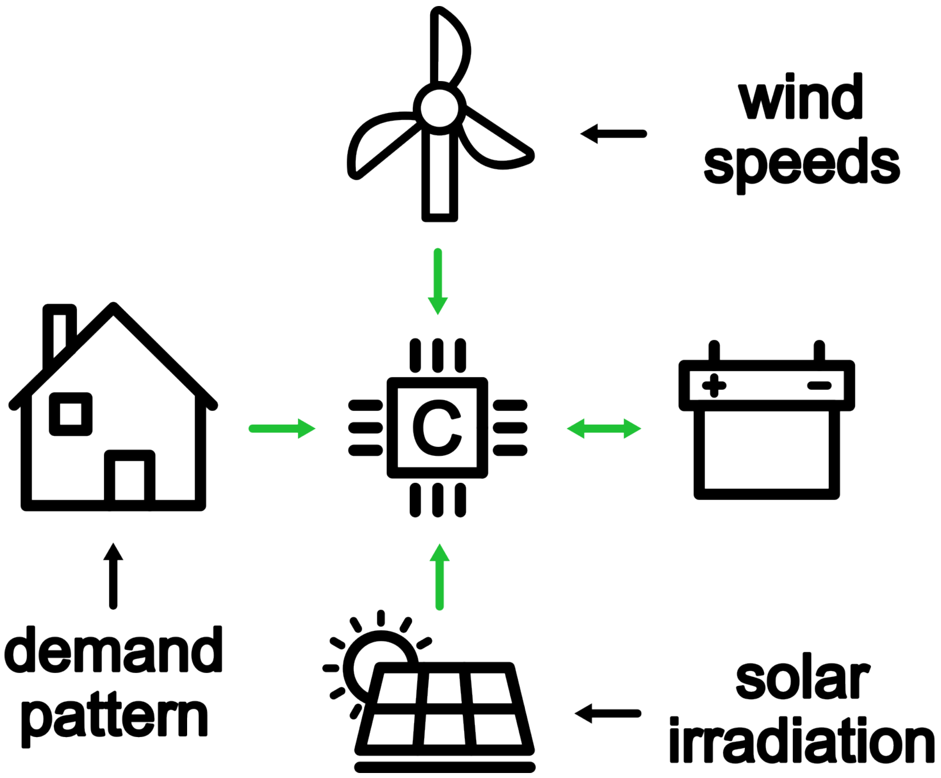


Figure 2.1: The connection of models and CSV files for the simulation example.

Inside of the YAML file, the parameters of the most important models are defined as in Table 2.1.

Table 2.1: Parameters for an example simulation.

Model	Parameter	Value	Unit
PV	m_area	4	m^2
	$NOCT$	44	°C
	$m\_efficiency\_stc$	0.198	–
	G_NOCT	800	$\mathrm{W/m}^2$
	P_STC	250	W
	$peak\_power$	600	W
	m_tilt	14	°
	m_az	180	°
	cap	500	W
Wind	p_rated	0.3	kW
	u_rated	10.3	m/s
	u_cutin	2.8	m/s
	u_cutout	25	m/s
	cp	0.40	–
	$diameter$	2	m
Battery	max_p	0.8	kW
	min_p	-0.8	kW
	Max_energy	10	kWh
	$charge\_efficiency$	90	%
	$discharge\_efficiency$	90	%
	soc_min	10	%
	soc_max	90	%

After defining all the parameters, connections, and outputs that need to be monitored, the simulation is run. The data from the output CSV file was processed and plotted, in order to analyze the system. In Figure 2.2 the battery power, the power from renewable energy sources (wind and solar), and the demand power are shown. It can be observed what the three power patterns look like. Additionally, the plot illustrates that the controller makes sure that the battery feeds power into the system, whenever the renewable energy sources do not generate sufficient power to cover the demand.

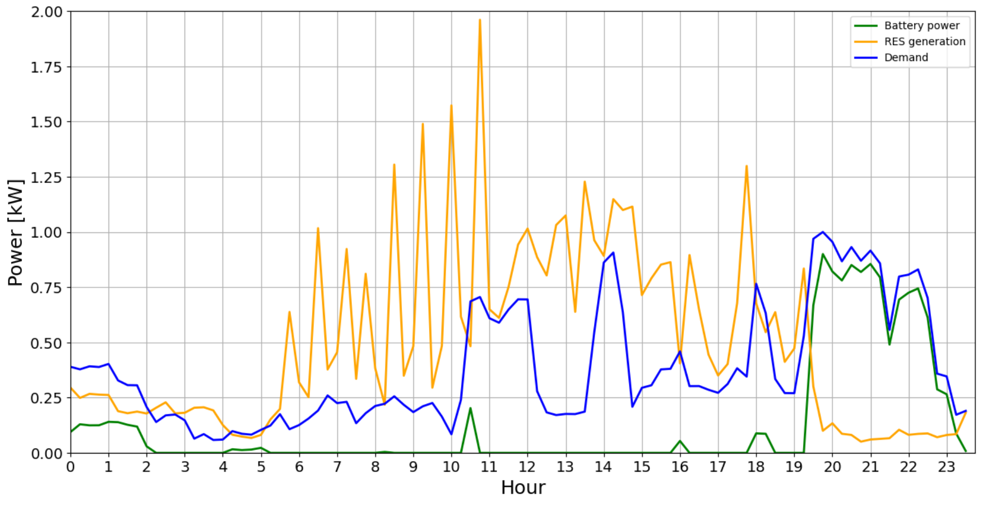


Figure 2.2: The battery, renewable source generation, and demand power of the example simulation.

In Figure 2.3 it can be seen how the state of charge (SOC) of the battery changes in relation to the patterns in Figure 2.2. The SOC decreases when battery power is used to meet the demand, while the SOC increases when the generation of renewable energy sources is larger than the demand.

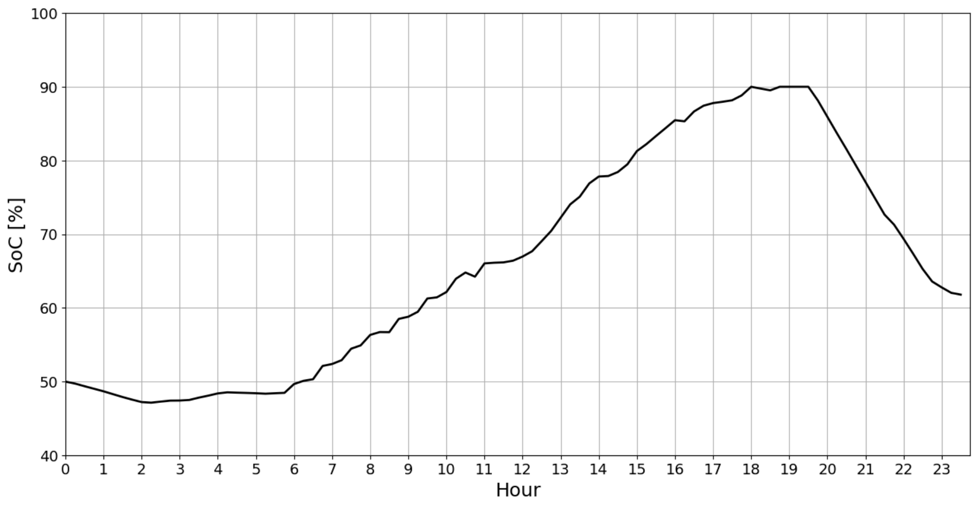


Figure 2.3: The state of charge of the system battery of the example simulation.

2.2. Hydrogen supply chain

Over the past decade, hydrogen has become a relevant energy carrier for the energy transition. The reason for this is its versatility, as it can operate as an energy carrier, a storage medium, a fuel, or a feedstock. Furthermore, it has a relatively high gravimetric energy density. It has several advantages compared to other energy carriers such as fossil fuels. One of the facets in which hydrogen outperforms fossil fuels, are the emissions that result from the use of the energy carrier. Where fossil fuels emit air pollutants such as CO_2 and NO_x, hydrogen only produces water vapor when used in a fuel cell or when it is combusted.

Another advantage of hydrogen is that it is the most commonly found element in the universe, whereas the presence of fossil fuels is much more limited. It should be acknowledged that this statement is somewhat misleading. Although the hydrogen element is abundant and common on planet Earth, it is not present in its pure form, which is H_2. In fact, to obtain this pure form of hydrogen, it must be extracted from other molecules, such as water or hydrocarbons.

After this extraction, hydrogen can be recombined with other elements, or be transitioned into another phase, to make the supply chain more efficient and safer. The conversion step can be followed by a storage stage, in which the energy carrier is stashed to use it at a later moment, or to keep a stock to guarantee energy at all times. Another stage is the transport stage, in which the energy carrier is physically transported from one place to the other. The final stage in the hydrogen supply chain is the utilization stage, in which hydrogen is consumed. The following sections dive deeper into the stages of the supply chain, and reveal the components that can be found in hydrogen systems.

2.2.1. Production

As already mentioned, hydrogen does not appear in the universe in its purest form. Instead, it needs to be extracted from other molecules. There is a large variety of methods to obtain hydrogen from other molecules. One of these methods is Steam Methane Reforming (SMR), in which steam reacts with methane to form carbon-oxide and hydrogen. Its reaction formula is provided in Equation 2.1.

$$CH_4 + H_2O \rightarrow CO + 3H_2. \tag{2.1}$$

If the carbon-oxide that remains after hydrogen extraction is captured, the hydrogen is considered blue hydrogen. If this is not the case it is considered gray hydrogen.

Two other common hydrogen production methods are pyrolysis and electrolysis. In pyrolysis, natural gas is decomposed by applying a high temperature, with the absence of steam and oxygen. As a result, hydrogen is extracted together with solid carbon, which can be easily captured. This reaction is shown in Equation 2.2. Electrolysis on the other hand, uses electricity to extract hydrogen from water. If the electricity used is coming from renewable sources, the process is considered green. As electricity is used to produce hydrogen, and hydrogen can be used to produce electricity, this method can be used to store the electric energy as chemical energy. The device used to produce hydrogen with electrolysis is called an electrolyzer. The reaction for electrolysis is provided in Equation 2.3.

$$CH_4 \rightarrow C + 2H_2. \tag{2.2}$$

$$2H_2O \rightarrow O_2 + 2H_2. \tag{2.3}$$

2.2.2. Conversion

After hydrogen is extracted from another medium, it can be converted to have more appealing properties for storage, usage or transport. This can done by combing the hydrogen molecules with other elements, for example with nitrogen to form ammonia, which has a higher energy density.

Another method to achieve better energy density is compression. This is done by a hydrogen compressor, which uses electrical energy to compress the initial hydrogen gas to a higher pressure and as a result occupy less space.

2.2.3. Storage

As mentioned before, hydrogen can be stored before it is used. This can be done in liquid or in gaseous form. The most important parameter in hydrogen storage is the capacity or size of the storage. The investment costs are proportional to size, and therefore hydrogen is stored by occupying low volume. This is done by using a compressor, as mentioned in Section 2.2.2.

2.2.4. Transport

The produced or stored hydrogen can be transported when it must be used in a different location than the production site. This can be done by using tube trailers or pipelines. An important factor to consider in these transportation methods is leakage and pressure loss. This is highly dependent on the material of the storage medium, and can influence the efficiency of the hydrogen supply chain. Steel for example, has lower permeability than polymer based materials, but is more expensive. For short distances or at low pressures, the losses incurred from transport are often negligible.

2.2.5. Utilization

After the hydrogen passed one or more of the previous stages, the hydrogen can be finally used. The hydrogen can be used directly as a fuel, for example in hydrogen fueled vehicles. However, it can also be converted (back) to electricity by using red-ox reactions. The device responsible for this conversion is a fuel cell.

2.3. Optimization

To find optimal energy system configurations, optimization theory can be applied. This is particularly useful when designing an energy system in which parameters are still undefined. Furthermore, it can be used to evaluate an existing system by comparing it to an optimal configuration. This section starts with basic optimization theory and a classification of optimization problems. After this, the optimization algorithms are classified. Finally, a short summary of the algorithms used in simulation-optimization is provided.

2.3.1. Optimization theory

Optimization is referred to the study of problems that look for the minimal of maximal values for an objective function on a predetermined domain [9]. Optimization problems are often formulated as in Equation 2.4:

$$\begin{aligned} \min_{\theta \in \Theta} \quad & f(\theta), && (2.4) \\ \text{s.t.} \quad & g_i(\theta) \leq 0, \quad i = 1, \ldots, m, && (2.5) \\ & h_j(\theta) = 0, \quad j = 1, \ldots, p. && (2.6) \end{aligned}$$

This equation defines an objective function $f(\theta)$ that must be either minimized or maximized. The decision variable θ is bound to search space Θ, which is a set including all possible solutions. The search space is bounded by constraints, which can be categorized into two types: inequality constraints (Equation 2.5) and equality constraints (Equation 2.6).

There are various optimization problem types. Each problem type is characterized by properties of the objective function, the constraints, and the decision variables. The optimization approach for solving a problem depends on the type of optimization problem, which makes it important to identify the problem type. Some common types of problems are:

- **Linear problems:** the objective function and constraints are linear;
- **Nonlinear problems:** the objective function or any of the constraints are nonlinear;
- **Multi-objective problems:** multiple objective functions are optimized simultaneously;
- **Stochastic problems:** some of the parameters are uncertain.

Multiple types can be grouped into broader classes of problem types based on their mathematical properties and solution approaches. Figure 2.4 illustrates a taxonomy of these problem types.

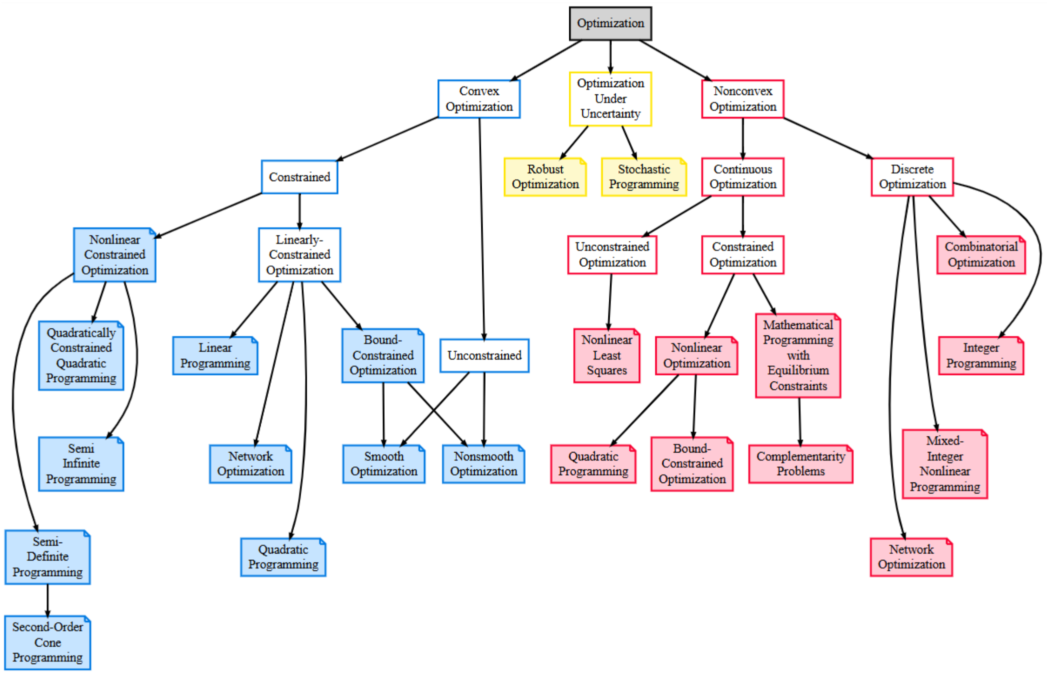

Optimization
Convex Optimization
Optimization Under Uncertainty
Nonconvex Optimization
Constrained
Robust Optimization
Stochastic Programming
Continuous Optimization
Discrete Optimization
Nonlinear Constrained Optimization
Linearly-Constrained Optimization
Unconstrained Optimization
Constrained Optimization
Combinatorial Optimization
Quadratically Constrained Quadratic Programming
Linear Programming
Bound-Constrained Optimization
Unconstrained
Nonlinear Least Squares
Nonlinear Optimization
Mathematical Programming with Equilibrium Constraints
Integer Programming
Semi Infinite Programming
Network Optimization
Smooth Optimization
Nonsmooth Optimization
Quadratic Programming
Bound-Constrained Optimization
Complementarity Problems
Mixed-Integer Nonlinear Programming
Semi-Definite Programming
Quadratic Programming
Network Optimization
Second-Order Cone Programming

Figure 2.4: Classification tree of optimization problems [10].

One of the distinctions that can be made between optimization problems is whether the problem is convex or non-convex. This distinction is made based on the objective function $f(\theta)$. The objective function is convex when it meets the condition defined in Equation 2.7 derived from [11].

$$f(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda f(\theta_1) + (1-\lambda) f(\theta_2). \tag{2.7}$$

In which λ is a number between 0 and 1. According to the equation, any function value on a straight line between two points θ_1 and θ_2 on the function $f(\theta)$ is less than or equal to the weighted average of $f(\theta_1)$ and $f(\theta_2)$. More concisely, if the function is truly convex, it is impossible to draw a straight line between two points on the function that cuts through the function itself. If on the other hand it is possible to do this, the function is considered non-convex, and non-convex optimization is required to find the global optimum.

An optimization problem can be further defined by the properties of the search space. In the case it consist of a definite set of possible values for the decision variables, the problem is called discrete and discrete optimization needs to be applied. A simple example of a discrete problem is a problem in which the decision variables are required to be integers ($\theta \in \mathbb{Z}$). This is called integer programming. Another example of discrete optimization is combinatorial programming, in which the optimum is a combination of a set of discrete variables. In continuous optimization on the other hand, the search space is a continuous set, meaning that the decision variables can take any real value allowed by the constraints ($\theta \in \Theta \subseteq \mathbb{R}^n$).

Another distinction between problems that can be made is based on the presence of constraints. When the decision variables are not subject to any constraints, it is referred to as unconstrained optimization. Meanwhile, if there are constraints in place, it is referred to constrained optimization. The latter is generally more computationally expensive than the former, as constraints reduce the search space in such a way that the optimal value is harder to find [12].

As previously mentioned, simulators provide a way to model and evaluate complex energy system under various conditions. However, a simulator does not inherently provide the most optimal configuration of an energy system. Rather, it computes outputs based on inputs provided by the user. Therefore, by combining a simulator with optimization techniques, an optimal simulation configuration can be generated. In literature, the combination of the two fields is referred to as simulation optimization. The simulator can introduce stochasticity to the simulation on its own, the user can use random variables to simulate a scenario, or the simulation can be deterministic (absence of stochasticity). Either way, the expected value of the output of the simulation is the component that is being optimized in simulation optimization. Provided that the output of the simulator is defined as $f(x)$, Equation 2.4 can be rewritten as shown in Equation 2.8.

$$\min_{\theta \in \Theta} \quad E[f(\theta, \omega)]. \tag{2.8}$$

In this equation ω represents the possible randomness or noise included by the simulator, and $E[\cdot]$ represents the expected value of the simulation output.

When an analytical form of the objective function $f(\theta)$ and possibly also the constraints associated with the decision variable θ are unknown or too complex to formulate, the optimization is referred to as black-box optimization. In this type of optimization, only the inputs and outputs of the objective function are used, as its internal working is unknown or considered irrelevant. This is for example the case for physical experimentation, such as the trial-and-error processes used in the synthesis of molecules to find better material properties [13]. Another example of black-box optimization is found when the objective function and constraints are dependent on computer simulation [14]. In the case of black-box optimization, the derivative cannot be analytically found. Therefor black-box optimization is also referred to as derivative free optimization. The derivative can however be approximated by applying methods such as the finite differences method, although this can result in large evaluation times.

2.3.2. Algorithms

As mentioned in Section 2.3.1, the approach used to solve an optimization problem is highly dependent on the nature of the problem. Additionally, the desired outcome of the optimization influences the choice of the solving method. Because of the numerous problem types in optimization theory, various problem solving algorithms have been developed over time. Each of these algorithms has strengths and weaknesses, and corresponds to a subset of the problem types. Figure 2.5 illustrates a systematization to the classification optimization algorithms. Note that each algorithm can be a member of multiple unrelated classes.

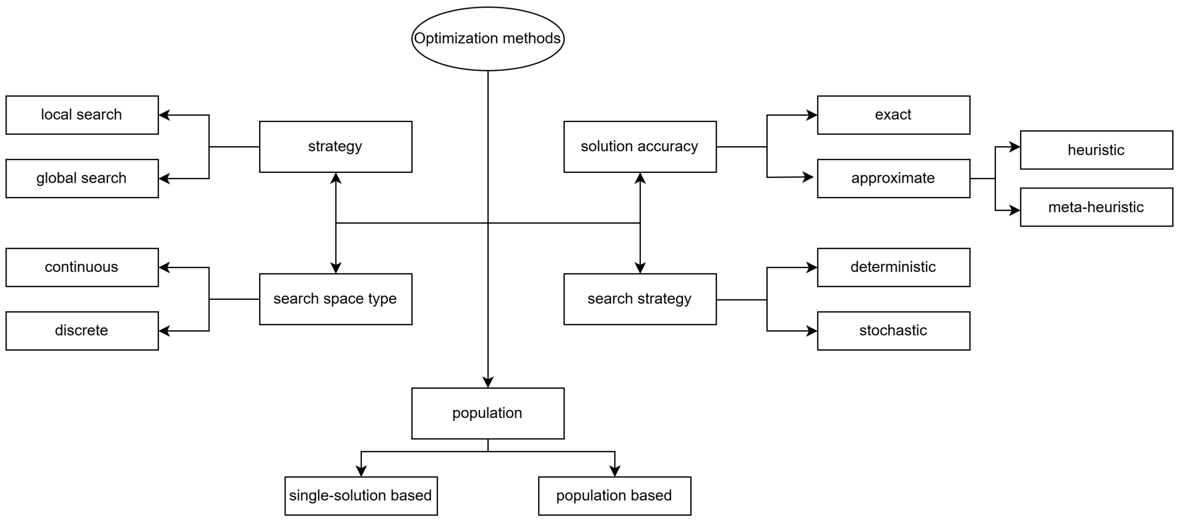


Figure 2.5: Classification tree of optimization algorithms.

The first distinction is made between discrete and continuous techniques. In discrete techniques, the search space is limited by a finite set or by a set that is countably infinite. Typical sets are integers, permutations, or graphs [15]. Continuous techniques, however, are not limited by this requirement, and the search space is continuous, such as a function. An example of a discrete (combinatorial) problem that can be solved with discrete techniques, is the Traveling Salesman problem (TSP) [16]. In this problem a salesman is given the task to visit cities from a set of cities. The constraints are the following:

- The salesman must visit all cities once;
- the salesman must end in the city in which he started his journey

In this problem, the search space is a finite set, namely the set of cities. More precisely, the solution is a permutation of this set.

Optimization algorithms are further defined based on solution accuracy. Algorithms can be either exact or approximate algorithms. Approximate algorithms look for a satisfactory (close to optimal) solution, whereas exact algorithms do not stop until the most optimal solution is found. Ideally the best solution is found, but for large and complex problems this can be time-consuming. Approximate algorithms are beneficial for these types of problems, but have the disadvantage that they are more sensitive to parameter settings, making the configuration of the algorithm more challenging [17].

Within approximate algorithms there is a differentiation between heuristic and meta-heuristic algorithms. A heuristic is a rule or strategy used to find a satisfactory solution to a specific problem. An example of a heuristic based optimization algorithm is greedy search, of which the rule is to take the best local optimal solution for each step from the point of view of the current state in the hope of finding the global optimum [18]. Meta-heuristic algorithms are algorithms that guide heuristics in the exploration and exploitation of the search space in an efficient way. Because of this characteristic, the algorithms are often more general and can be applied to different types of problems [19]. An example of a meta-heuristic algorithm is a genetic algorithm, in which the exploration of the search space is done in an efficient way by recombining solution properties found in different points of the search space. The algorithm applies a set of rules in an efficient manner, namely by recombination based on a fitness function and by introducing randomness (mutation).

Next to these categories, optimization algorithms can be classified based on their search scope. When the scope is local, the algorithm searches for an optimum within a local domain of the search space, and converges to the nearest local minimum or maximum. Often, this is not the best solution in the entire search space in non-convex problems. Global methods, on the other hand, aim to find the best solution in the entire search space, namely the global optimum. Figure 2.6 illustrates the difference between a local and a global minimum.

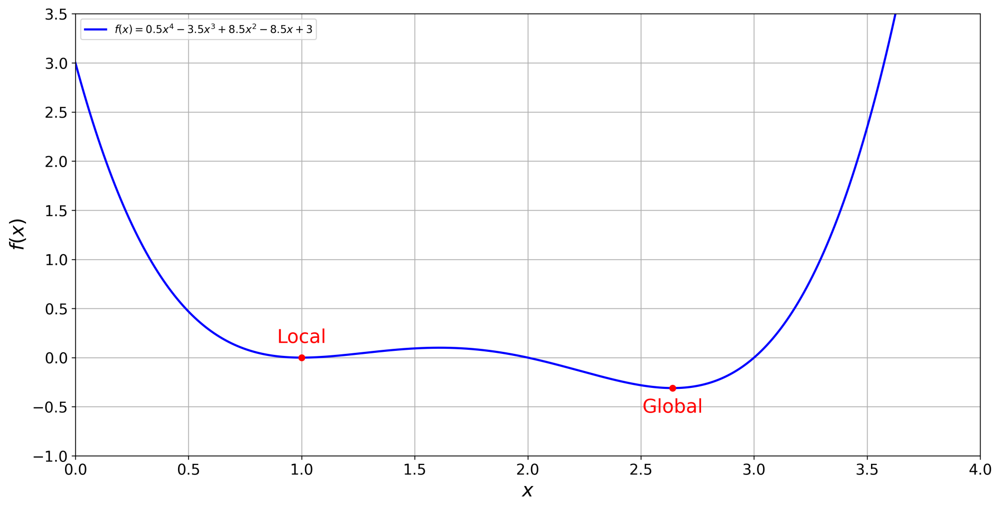


Figure 2.6: Local and global minimum of an example function.

Local search algorithms typically require fewer iterations and are generally less computationally demanding to find the optimum compared to global search algorithms. In case a local algorithm is used to solve a non-convex problem, a risk is introduced. Since in non-convex problems a local optimum is not necessarily a global optimum, it is possible for the local algorithm to get 'trapped' in this local solution, and never finding the global optimum. Because of this, the starting points needed to initiate the algorithm, need to be carefully chosen [20]. Global optimization algorithms have a reduced risk to converge to local optima. The reason for this, is that this type of algorithm is designed to escape from the local optima and to continue the search for a global optimum. Moreover the solution found by a good algorithm in this category can find the optimal solution irrespective of the starting points [21].

Besides the search scope, optimization algorithms can be categorized depending on their search strategy. Namely, algorithms can be defined as deterministic or stochastic. Deterministic algorithms follow rigorous mathematical approaches. They are therefore often referred to as mathematical programming. The algorithms rely on linear algebra as they frequently make use of gradients [22]. In stochastic algorithms on the other hand, the search towards an optimal solution involves a random process. This includes the use of a random initialization or the use of randomness in search moves. An example of a stochastic algorithm is the previously mentioned genetic algorithm, in which the first population is randomly chosen and mutation occurs with a certain probability.

Another distinction is made between optimization algorithms that are single-solution based and population based. In short, population based algorithms use multiple solution candidates to find the optimum in an efficient manner, while single-solution based algorithms iteratively try to improve single solution to achieve the optimum. Population based algorithms provide a better exploration of the search space, are less sensitive to convergence into local optima, and have a high potential in the domain of parallel computation compared to single-solution based algorithms. The downside however, is that the computations required are often more extensive, making the iterations slower. Furthermore, more memory is required compared to single-solution based algorithms.

A list of simulation optimization algorithms that are used in literature, including algorithms that are used for simulation optimizations, is presented in Table 2.2. A more complete version of this table containing a summary of each algorithm can be found in Appendix A.1 (Table A.1).

Table 2.2: Classification of (simulation) optimization algorithms used in literature.

Algorithm	**Multi-objective compatible**	**Meta-heuristic**	**Deterministic**	**Population based**	**Continuous**	**Global**	**Ref.**
Simulated Annealing	X	X			X	X	[23][1], [24][1], [25]
Genetic Algorithm	X	X		X	X	X	[26][1]
Tabu Search		X	X			X	[27][1]
Particle Swarm		X		X	X	X	[28][1]
Ant Colony		X		X		X	[29][1]
Bee colony		X	X	X	X	X	[30][1]

[1]Reference uses the algorithm for simulation optimization.

3 Methodology

3.1. Simulation

The simulator software that was used in this thesis is The Illuminator (v3.0.0-beta.5) [5]. This toolbox contains numerous models, but at the time of writing missed some hydrogen components. This section describes the hydrogen models that were added in this thesis project, and briefly discusses the models required to develop the scenarios presented later in this chapter.

3.1.1. Addition of hydrogen models

To broaden the model library of The Illuminator, the following hydrogen-related models were added:

- Hydrogen compressor;
- pipeline;
- electrolyzer;
- fuel cell;
- hydrogen storage.

To create these models, it was important to define the level of abstraction. In fact, energy systems can be modeled with varying degrees of complexity. For example, modeling the operation of an electrolyzer or fuel cell can be done as specific as on a molecular level, taking into account the various technologies and materials. By looking at the existing models, which will be briefly described in Section 3.1.2, it was noted that the models are defined as black box models. This means that the models focus is on the inputs and outputs of components, rather than their internal workings. For simulations at system level this was considered to be a satisfactory level of detail. The specific details of a component are less relevant for the operation of an entire system. Additionally, if the models contain calculation that increase computation costs while not adding significant value to the simulation of the system, simulation optimization becomes less appealing. Therefore, the models added in this thesis are at black-box level, and several assumptions were made to facilitate this.

Hydrogen compressor

The first model that was added to the Illuminator library, was a hydrogen compressor model. The purpose of a compressor is to increase the energy density of hydrogen. In other words, to store more energy in a smaller volume. This is useful to, for example, reduce the long term hydrogen storage size or to allow for a smaller fuel tank in hydrogen powered vehicles. The most important factors to take into account when modeling a hydrogen compressor are the hydrogen volumetric output and its power usage. Based on the parameters provided by the user of the Illuminator, such as the input pressure, desired output pressure, and compressor efficiency, these characteristics are calculated at each simulation step. Table B.1 contains all variables that were used to model the compressor model, along with their units. A black box representation of the compressor model is given in Figure 3.1.

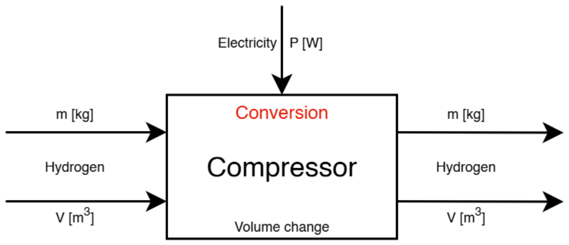


Figure 3.1: Black box representation of the hydrogen compressor model.

To find the power usage of the hydrogen compressor, the work required to compress the hydrogen from the input pressure p_2 to the output pressure p_2 is needed. For simplicity, the compression in this model was assumed to be an isentropic process, meaning that no heat is transferred to the surroundings of the compressor (adiabatic), and that no energy is lost due to friction or other loss inducing forces (reversible). Under this assumption, the work required to compress a gas from one pressure to another is given by Equation 3.1, which was derived from the European Forum for Reciprocating Compressors [31].

$$w_{isentropic} = \frac{\gamma}{\gamma - 1} \cdot R \cdot T_1 \cdot \left(\left(\frac{p2}{p1} \right)^{\frac{\gamma-1}{\gamma}} - 1 \right) \quad \text{[J/mol]}. \tag{3.1}$$

To account for losses in the compressor model, an efficiency factor was implemented by using Equation 3.2, resulting in the real electrical work required by the compressor. This was necessary as isentropic compression is not possible in real life.

$$w_{real} = \frac{w_{isentropic}}{\eta_{compressor}} \quad [J/mol]. \tag{3.2}$$

Considering $\dot{m}_{in}$ is the input flow of hydrogen in kg/s, Equation 3.3 was formulated to provide the power consumed by the compressor in each timestep.

$$P = w_{real} * \frac{\dot{m}}{M} \quad \text{[J/mol]}. \tag{3.3}$$

After the power required to compress hydrogen is calculated, the model calculates the change in volumetric output flow. For this, the volumetric energy density is needed, which is dependent on pressure and temperature. Figure 3.2 describes this behavior.

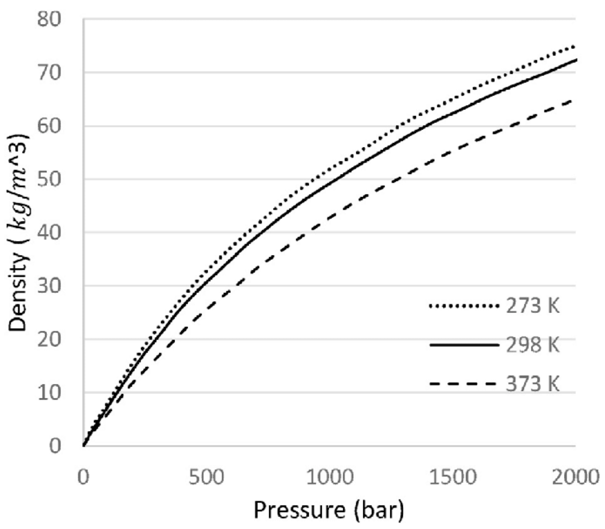


Figure 3.2: The dependency of the volumetric density of hydrogen on pressure and temperature [32].

Often, the volumetric density of gases is approximated by the ideal gas law described in Equation 3.4.

$$\rho = \frac{p}{RT} \quad [\text{kg/m}^3]. \tag{3.4}$$

As the name suggests, this equation assumes ideality, neglecting physical properties of the molecules of gas at different temperatures and pressures. The difference in the behavior of a real gas compared to an ideal gas becomes apparent at higher pressures. To account for this, the model calculates the volumetric density calculation with the use of Equation 3.5, which includes a Z-factor. This factor, which has been experimentally derived by comparing the ideal gas with hydrogen gas, is presented in the table in Figure B.1.

$$\rho = \frac{p_{H_2} M_{H_2}}{Z \cdot R \cdot T} \quad [\text{kg/m}^3]. \tag{3.5}$$

Knowing the volumetric density, the volumetric output flow $\dot{V}$ in m^3/s is calculated by using Equation 3.6.

$$\dot{V} = \frac{\dot{m}}{\rho} \quad [\text{m}^3/\text{s}]. \tag{3.6}$$

Pipeline

The next model that was added to the Illuminator was the pipeline model. This model can be used for systems in which hydrogen needs to be transported over a long distance with the use of pipes. Specifically, this model has been created to simulate hydrogen pressure and mass loss in pipes. The variables used in the pipeline model are listed in Table B.2. Figure 3.3 shows a black-box representation of the pipeline model.

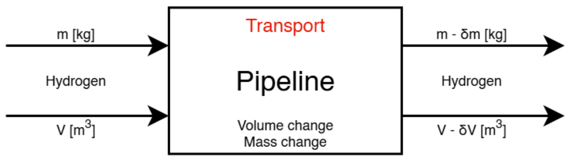


Figure 3.3: Black box representation of the pipeline model.

For the calculation of the pressure loss over a pipeline, the model uses the Darcy-Weisbach equation (Equation 3.7).

$$\Delta p = f \cdot \frac{L \cdot \rho \cdot v^2}{2 \cdot D} \quad [\text{Pa}]. \tag{3.7}$$

In this equation f is the friction factor, which is dependent on characteristics of the hydrogen and the pipe, and the type of flow. The type of flow is determined by Reynolds number Re. For values lower than 2300 the flow is considered laminar, while if it is larger than 4000 it is considered turbulent. Between these thresholds the flow is transitional, which is a combination of both laminar and turbulent flow. For simplicity, this type of flow has been neglected in the model, and turbulent flow was adopted for Reynolds numbers higher than 2300. The equation used for the calculation of the friction factor is provided in Equation 3.8 [33]. The first part of the equation is used for laminar flow. For the turbulent flow the Colebrook–White equation is normally used, but as this is an implicit function it can be approximated with the Haaland equation, which is the second part of Equation 3.8. This equation shows a maximum error of 3% compared to the Colebrook–White depending on the roughness and Reynolds number, which is considered accurate for this case [34].

$$f = \begin{cases} \frac{64}{Re} & \text{if } Re < 2300, \\ (-1.8 \log_{10}[(\frac{\epsilon}{3.7D})^{1.11} + \frac{6.9}{Re}])^{-2} & \text{if } Re > 4000. \end{cases} \tag{3.8}$$

Reynolds number is a dimensionless quantity used in fluid dynamics used to predict patterns in fluid flow. It is given by Equation 3.9. The density in this equation can be found by using the previously defined Equation 3.5. From this it can be concluded that the flow regime the hydrogen finds itself in is mainly dependent on the variables of pipeline diameter and flow velocity.

$$Re = \frac{\rho \cdot v \cdot D}{\mu}. \tag{3.9}$$

In addition to the pressure loss, hydrogen can leak through the pipe wall because of permeation. The amount of leakage is negligible when materials with low permeability are used, such as a steel. However, as polymer pipes are sometimes used for natural gas transport, and reusing gas pipes is considered for hydrogen transport, permeation was accounted for in the model. Equation 3.10 describes the amount of hydrogen lost per second. The equation was derived from Fick's first law of diffusion.

$$Q = P'_{H_2-poly} \cdot \frac{A/10^{-4} \cdot (p_{in} - p_{out})}{d/10^{-2}} \quad [\text{cm}^3/\text{s}]. \tag{3.10}$$

In this equation P_{H_2-poly} is the permeability coefficient of hydrogen for a specific material. As natural gas pipes are typically made from High-Density Polyethylene (HDPE), this material was considered in this model. The coefficient is dependent on the pipe material, the type of gas, and the pressure of the gas. The values of these coefficients for hydrogen in HDPE pipes at different pressures can be found in Table B.3. The values were taken from a paper that determines them experimentally and through estimation [35]. The values obtained by the latter method were used in this model. Note that the paper provides the coefficients in the unit cm^3(STP)·cm/(cm^2·s·cmHg). In Table B.3 this has been converted to be in cm^3(STP)·cm/(cm^2·s·Pa) for simplicity (1cmHg≈ 133.3Pa).

Electrolyzer

The third model that was created was an electrolyzer model. In hydrogen systems, an electrolyzer is a machine that uses electrical energy to extract hydrogen from water. In fact, electrolyzers convert electrical energy to hydrogen. The two key features that needed to be modeled are the hydrogen output flow and the required water input flow. The variables used by the model are listed in Table B.4. Figure 3.4 shows a black-box representation of the model.

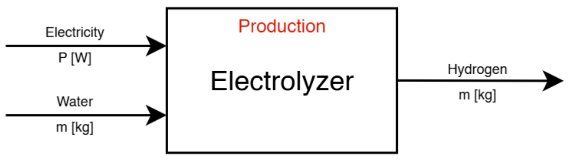


Figure 3.4: Black box representation of the electrolyzer model.

The model was defined by an input power, water consumption, and the hydrogen output flow. In Equation 3.11 the hydrogen output flow is provided based on the power input.

$$\dot{m}_{H_2,out} = \frac{P_{in} \cdot \eta_{ez}}{hhv_{H_2}} \cdot M_{H_2}/1000 \quad [\text{kg/s}]. \tag{3.11}$$

Electrolyzers are often unable to increase or decrease production over a short period of time, and therefore the power provided to the electrolyzer cannot change abruptly. Additionally the power flow is limited by a maximum input power rating. These constraints were modeled by using Equation 3.13 and Equation 3.12 respectively.

$$0 \leq P_{in} \leq P_{in,rated}. \tag{3.12}$$

$$|P_{in,t} - P_{in,t-1}| \leq P_{in,ramp}. \tag{3.13}$$

The amount of water necessary for the production of hydrogen is related to the input power and the efficiency. The general chemical reaction to produce hydrogen from water is provided in Equation 3.14, from which it has been derived that the molar ratio of water used to hydrogen produced is one to one. Consequently the model calculates the required water flow by using Equation 3.15.

$$2H_2O \rightarrow 2H_2 + O_2. \tag{3.14}$$

$$\dot{m}_{water} = \frac{P_{in} \cdot \frac{M_{H_2O}}{M_{H_2}}}{hhv_{H_2} \cdot \eta_{ez}} \quad [\text{kg/s}]. \tag{3.15}$$

Fuel cell

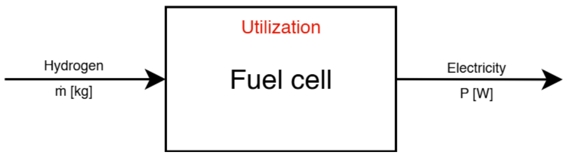


Figure 3.5: Black box representation of the fuel cell model.

The chemical energy that hydrogen holds can be transformed into electrical energy. The electrochemical reactions to achieve this conversion take place in a fuel cell. The electrical power that is produced by a fuel cell can be calculated with Equation 3.16.

$$P_{fc,t} = \frac{\dot{m}_{H_2} \cdot \eta_{fc} \cdot hhv_{H_2}}{M_{H_2}} quad[\text{W}]. \tag{3.16}$$

Similarly to the electrolyzer model, the output of this model was limited by the physical properties of the fuel cell, setting maximum output (Equation 3.17) and ramp up values (3.18).

$$0 \leq \dot{m}_{H_2} \leq \dot{m}_{H_2,max}. \tag{3.17}$$

$$P_{fc,t} = min(P_{fc,t-1} + \Delta P_{fc,max}, P_{demand}). \tag{3.18}$$

Hydrogen storage

The final model added to the Illuminator library was a hydrogen storage model. The model keeps track of the hydrogen that flows in- and out of the storage while taking possible losses into account. Furthermore, the model considers the state of charge, and based on this determines its in and output flows. The variables used in this model are listed in Table B.6.

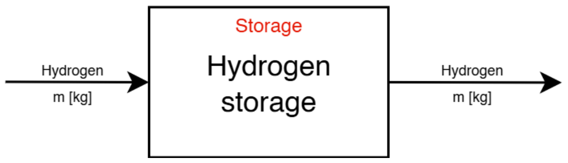


Figure 3.6: Black box representation of the hydrogen storage model.

The hydrogen storage model was modeled to have a negative net flow when the output flow is larger than the input flow and vice versa. Equation 3.19 describes this behavior.

$$Q_{H_2-sto,t} = \begin{cases} Q_{H_2-sto,t-1} + \dot{Q}_{H_2-sto-in,t} \cdot \eta_{H_2-sto,charge} & \text{for} \quad \dot{Q}_{H_2-sto-in,t} > 0, \\ Q_{H_2-sto,t-1} + \frac{Q_{H_2-sto-in,t}}{\eta_{H_2-sto,discharge}} & \text{for} \quad \dot{Q}_{H_2-sto-in,t} < 0. \end{cases} \tag{3.19}$$

The flow of the hydrogen storage is bounded by the minimal and maximal in- and output flow. Furthermore, the state of charge of the storage is limited by a minimum and a maximum. These thresholds define the operational range of the storage model.

3.1.2. Existing models

In this section the existing models that were used in the scenarios of Section 3.3.1, 3.3.2, and 3.3.3 are briefly described.

Battery

The battery model input, defined as $flow2b$, receives a value that is positive when the battery must be charged and negative when the battery must be discharged. Based on this input, the model keeps track of the state of charge (SOC) internally. This is done by taking into account the following model parameters:

- max_p: the rated input power [kW];
- min_p: the rated output power [kW];
- max_energy: the total battery capacity [kWh];
- $charge\_efficiency$: the efficiency with which the battery is charged [%];
- $discharge\_efficiency$: the efficiency with which the battery is discharged [%];
- soc_min: the lowest achievable state of charge [%];
- soc_max: the highest achievable state of charge [%].

The most relevant variable in this model is the state of charge, which can be used to make decisions in the system.

Photovoltaic (PV)

The photovoltaic (PV) model requires irradiation data as input. This data should consist of the Global Horizontal Irradiance (GHI), Diffuse Horizontal Irradiation (DHI), Direct Normal Irradiation (DNI), ambient temperature, solar height, form factor, and the solar azimuth at each timestep. Together with solar panel specific data, such as its efficiency under standard test conditions and its power rating, the power generation per square meter can be determined. Based on the area specified in the parameters, the output provides the generated power per timestep.

Electrical demand

The electrical demand model is driven by a demand file. In this file, the demand, for example of a household, is given for each timestep. In the demand model this demand can be scaled, if necessary, by a tuning parameter. The output represents the total demand per timestep.

Wind turbine

The input of the wind turbine model expects wind speed data for each timestep of the simulation. The parameters that need to be specified for this model are the rated power, the rated wind speed, the cut-in and cut-out speed, the power coefficient, and the diameter of the rotors. Together with the input, the parameters are used to calculate the generated power at each timestep, which is provided at the output of the model.

Electric vehicle (EV)

The electric vehicle (EV) model provides the power required at each timestep for charging an electric vehicle. The power required during a charging cycle, the battery capacity, the timestep in which the EV should start charging, and the state of charge at arrival are defined in the parameters. The input requires presence data, indicating whether the EV is present at a charging station. Based on these parameters, the state of charge of the EV is monitored for each timestep.

3.2. Optimization

3.2.1. Problem type

Selecting an optimization algorithm compatible with The Illuminator toolbox required defining the type of optimization problems that potentially need to be solved. Therefore, characteristics of the possible search spaces and objective functions had to be analyzed.

The last part of Section 2.3.1 mentioned that optimization problems in which the analytical objective function is unknown or complex are called black-box optimization problems. In the case of this thesis, the objective function is dependent on simulation outputs. Since the computations in the simulation are complex and unknown to the optimizer, the optimization problems considered in this thesis were defined as black-box optimization problems. Nevertheless, it was possible to make assumptions and educated guesses regarding the properties of the search space. By applying these assumptions, the algorithm choice, which is described in the next section, became simpler.

First of all, the search space of potential problems was assumed to be continuous in most cases. Most problems faced in energy systems are regarding the optimization of one or more variables that can attain any value within a range of values and not a value from a discrete set. An exception of this is the problems regarding system planning.

Additionally, the assumption was made that the simulations were deterministic, meaning that each simulation yields the same results when the same parameters are used. In other words, stochasticity does not play a role in the simulations.

Furthermore, by looking at the nature of the models used in the simulator, it was assumed that the search space contains non-linearity. An example of this property is the influence of the sun's angle of incidence (AOI) on PV generation. Namely, the power output of PV systems is dependent on trigonometric relationships with the AOI, introducing non-linear behavior.

Another property that has been considered is convexity. Most models are expected to be individually convex, such as the battery model. However, in the case that the system requires a form of control, the decisions made by the system controller can introduce non-convex behavior in the search space. Examples of control methods that can introduce non-convexity are switching behaviors (e.g., charging or discharging of a battery), binary logic (e.g., toggling a generator), or piecewise functions (e.g., when the production of a generator is dependent on thresholds). In other words, control can cause discontinuities in the search space, making it non-convex. It must however be noted that it is possible for the search space to be piecewise-convex, meaning that it is an accumulation of convex functions.

Finally, an assumption was made regarding the dimensionality of the problems that need to be solved with simulation optimization. Dimensionality refers to the number of decision variables pertaining to the optimization problem. Due to large dimensionalities, search spaces increase, negatively influencing the computation cost. In this thesis, the assumption was made that only a small number of variables need to be optimized. To summarize, the search space is assumed to have the following properties:

- continuous;
- non-linear;
- non-convex or piecewise-convex;
- small dimensionality.

3.2.2. Algorithm choice

Aside from the properties mentioned in the previous section, it was important to take evaluation time into consideration for the choice of a suitable algorithm. The evaluation time, i.e. the time it takes to do one simulation, is in the order of minutes. Large number of evaluations are therefore practically infeasible[36]. It must be noted that this number is problem specific and that it is usually found empirically.

Nevertheless, literature on optimization algorithms reveals a trend. In comparative studies that use complex problems as benchmark, the most efficient algorithms in terms of accuracy and the number of evaluations are Particle Swarm optimization (PSO), Genetic algorithm (GA), and Simulated Annealing

(SA). Among these three, the general consensus is that particle swarm optimization outperforms the other two when the same problem is evaluated [37] [38] [39]. In the specific field of simulation optimization, comparative studies point out that PSO and GA are the most accurate algorithms for the shortest computation time [40].

For simple convex problems of which the analytical objective function is known, gradient-based algorithms are more efficient. However, as it is unknown whether this is the case in black-box optimization, simulation optimization literature does usually not consider this type. Estimation of the gradient requires additional evaluations, which could make it inefficient. Additionally, as most gradient-based algorithms are local algorithms, they are not applicable to non-convex search spaces. However, as explained in Section 3.2.1, the search space for black-box optimization problems can be piecewise-convex. For this reason, gradient-based algorithms are considered in this thesis.

Section 3.2.1 also describes that the search space is assumed to be most often continuous. The discrete algorithms from Table 2.2 are therefore not considered. Adaptations to discrete algorithms that work with continuous search spaces can be made, but these are generally less efficient than algorithms that are specifically made for this purpose.

Based on this information, the following three algorithms were evaluated in this thesis:

- Particle Swarm Optimization;
- Gradient-based algorithm (L-BFGS-B);
- Genetic algorithm.

3.2.3. Integration

Before diving deeper into the theory of the selected optimization algorithms, a blueprint for the implementation of simulation optimization was created. To merge simulation with optimization, an optimization layer has been built around The Illuminator core code. As explained in Section 2.1, a scenario is defined using a configuration file in YAML format. The Illuminator takes this configuration file and possible data input files, and executes the simulation. The scenario defines the system to be simulated and its parameters. The output of each simulation is a comma separated values (CSV) file, that contains the values of predefined simulation parameters at each timestep. Figure 3.7 illustrates the workflow of the Illuminator.

Optimization algorithms begin by initializing decision variables. Subsequently, the objective function is evaluated with these variables. Based on this evaluation, the variable values are either changed according to an algorithm-specific strategy or accepted as optimal values. Whether or not the algorithm continues and changes the decision variables depends on the stopping criterion. This criterion is either a maximum number of iterations, or converging characteristics. If the stopping criterion is not met, the algorithm updates the values and enters the evaluation loop. The general flow of optimization algorithms is illustrated in Figure 3.8.

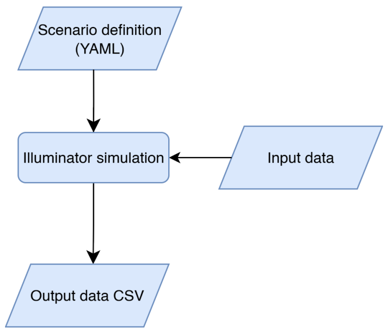
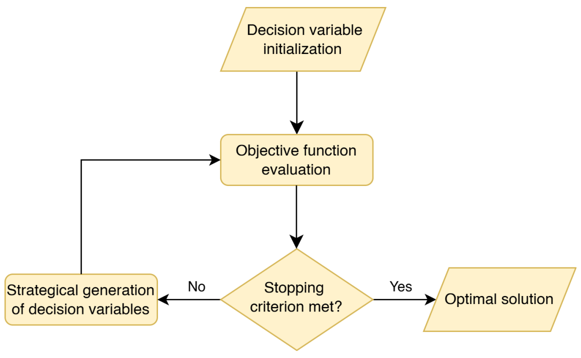


Figure 3.7: The operational flow of the Illuminator.


Decision variable initialization
Objective function evaluation
Strategical generation of decision variables
No
Stopping criterion met?
Yes
Optimal solution
Figure 3.8: The general operational flow of an optimization algorithm.

The concept of simulation optimization is to combine the two above-mentioned flows. To do this, two integration steps were added.

The first integration step consisted of the integration of the insertion layer. This layer is necessary, as the decision variables obtained from the optimization flow cannot be directly used in a simulation. The layer makes a copy of the original scenario file and replaces the desired parameters with the values obtained from the optimization loop.

As optimization algorithms are unable to process raw CSV output files, an evaluation layer was added as the second integration step. This layer extracts the relevant values from the output file and uses these to evaluate a cost function. The cost function, in combination with the simulation part of the flow, essentially replaced the objective function evaluation of the original optimization flow.

The resulting flow of the simulation optimization framework is illustrated in Figure 3.9.

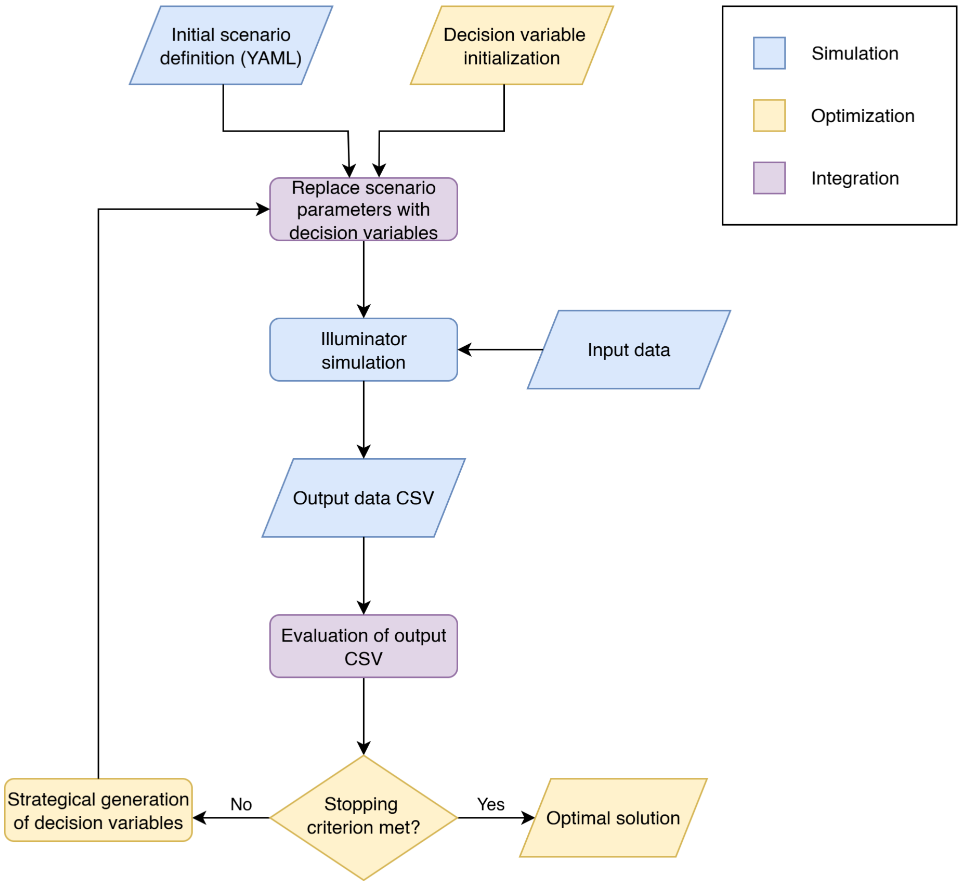

Initial scenario definition (YAML)
Decision variable initialization
Simulation
Optimization
Integration
Replace scenario parameters with decision variables
Illuminator simulation
Input data
Output data CSV
Evaluation of output CSV
Stopping criterion met?
No
Yes
Strategical generation of decision variables
Optimal solution

Figure 3.9: The operational flow of simulation optimization, obtained by combining the operation of the Illuminator and an optimization framework. Two integration layers are added to facilitate this.

3.2.4. Algorithm evaluation

The algorithms that were selected in Section 3.2.2 were compared by applying them to three optimization problems. The problems are presented later on in this thesis (section 3.3.1, 3.3.2, and 3.3.3). The algorithms were compared based on the set of criteria listed below, which are based on three performance categories, namely efficiency, reliability, and quality of solution [41]:

1. **Solution accuracy**: This criterion indicates how close the solution found by an algorithm is to the global optimum. Since the true global optima of the problems were unknown, the accuracy was measured by comparing the cost found by the algorithm with the best found cost among all algorithms. This was done by calculating the gap, which is defined by Equation 3.20 and is measured in percentages. $\mathcal{C}_i$ is the cost found by algorithm i and $\mathcal{C}^*$ is the best-found cost among all algorithms.

$$gap = \frac{|\mathcal{C}_i - \mathcal{C}^*|}{|\mathcal{C}^*|} \cdot 100 \quad [\%]. \tag{3.20}$$

2. **Convergence speed**: This is the speed with which the algorithm finds a stable optimal solution. In this thesis, this is defined by the number of iterations required until the optimization finds a stable solution. This criterion is used to get an indication of the algorithm efficiency. This is important since, as explained in Section 3.2.2, the number of iterations should be minimized to make simulation optimization a feasible technique.
3. **Flexibility**: This metric indicates the flexibility with which the algorithm can be applied to different types of problems. In other words, how well the optimizer works on different problem types. This is not a quantitative metric, but rather a general indicator.

3.2.5. Particle Swarm Optimization

After the blueprint for simulation optimization was created, the optimization algorithms were implemented. The first algorithm was Particle Swarm Optimization (PSO).

To begin the algorithm, a population of particles that are randomly distributed across the search space is initialized. Each particle represents a position in the problem's search space, corresponding to a set of parameter values. In each iteration, a simulation is executed for each particle by using its current parameter set. After this, the resulting output of each simulation is evaluated. Based on the cost or quality of the solution, each particles is moved to a new location that is evaluated in the next iteration. The movement of each particle is dependent on the particle's velocity, which is determined by the particle's previous velocity (based on an inertia factor), the particle's personal best solution, and the population's global best solution. The latter two factors include stochasticity, and their influence on the particle's new velocity can be tuned with weight factors. The velocity determines the next position of the particle, i.e., the next evaluated solution. The personal best solution of each particle and the global best solution of the population are updated in each iteration. After the stopping criterion is met, the global best solution at that moment is nominated as the optimal solution for the problem. The pseudo-code for PSO is provided in Algorithm 1.

Algorithm 1 PSO pseudo-code

```
1:  procedure PSO($f$, $n_p$, $d$, $w$, $c_1$, $c_2$, $maxIter$)
2:      Initialize position $x_i$ and velocity $v_i$ for each particle $i \in \{1, \dots, n_p\}$ in $d$ dimensions
3:      Set personal best $pbest_i \leftarrow x_i$ for each particle
4:      Evaluate $f(pbest_i)$ and find global best $gbest$ among all $pbest_i$
5:      Set iteration counter $k \leftarrow 0$
6:      while $k < maxIter$ AND (not converged) do
7:          for each particle $i$ do
8:              Generate random numbers $r_1$, $r_2 \sim \mathcal{U}(0, 1)$
9:              Update velocity:
                    $v_i \leftarrow w \cdot v_i + c_1 \cdot r_1 \cdot (pbest_i - x_i) + c_2 \cdot r_2 \cdot (gbest - x_i)$
10:             Update position: $x_i \leftarrow x_i + v_i$
11:             Evaluate $f(x_i)$
12:             if $f(x_i) < f(pbest_i)$ then
13:                 $pbest_i \leftarrow x_i$
14:         Update $gbest$ if any $pbest_i$ is better
15:         $k \leftarrow k + 1$
16:     return $gbest$
```

f: objective function
n_p: population size
d: dimensions (number of decision variables)
w: inertia weight
c_1, c_2: weight factors
$maxIter$: maximum number of iterations
x_i: position of particle i
r_1, r_2: random numbers
v_i: velocity of article i
p_i^{best}: personal best of particle i
g^{best}: global best

The implementation of the PSO algorithm in Python was done by using the *pymoo* [42] optimization package (version *0.6.1.3*). Each particle position requires an evaluation of the cost function, and therefore a simulation run. Hence, for population size n_p and the number of generations n_g (either generations required or maximum number of iterations $maxIter$), the total number of simulations required is defined by Equation 3.21.

$$n_s = n_p \cdot n_g. \tag{3.21}$$

By default *pymoo* 0.6.1.3 executes the evaluation of each particle sequentially, meaning that the total simulation $t_{s-total}$ time is equal to the total amount of simulations required for one problem n_s times the run time for one simulation t_s, which is defined in Equation 3.22.

$$t_{s-total} = n_s \cdot t_s \quad \text{[s]}. \tag{3.22}$$

To reduce the total computation time, parallelization was implemented. Since in PSO the particles within a generation are independent of each other, their simulations can be computed in parallel. That is, *line 11* until *line 13* of Algorithm 1 can be executed in parallel. After all the parallel simulations within a generation are executed, the solutions are evaluated and a new generation is created (the code is resumed from *line 14*). Parallelization was implemented by using the standard Python 3.11 *multiprocessing* package. After this implementation, which is illustrated in Figure 3.10, the theoretical new total simulation time is equal to the number of generations n_g times the singular simulation time t_s(Equation 3.23).

$$t_{s-total} = n_g \cdot t_s \quad \text{[s]}. \tag{3.23}$$

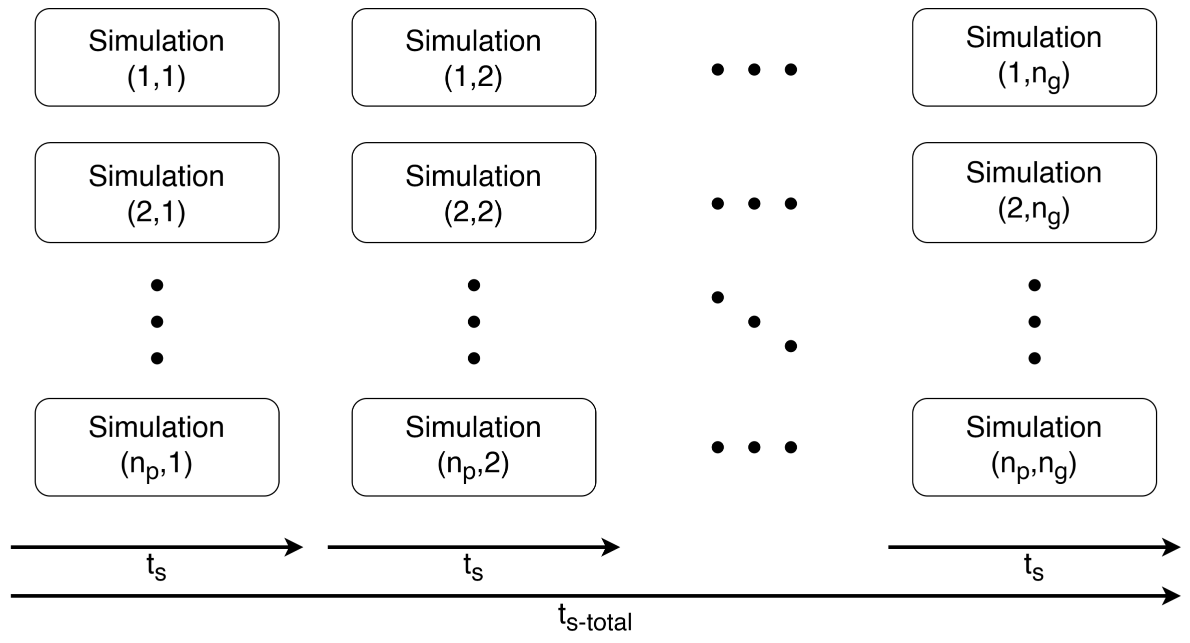


Figure 3.10: Parallelization of the particles in each generation of the PSO. The rows represent the particles, the columns represent the generations. At the bottom the time flow is defined.

In practice, the number of parallel simulations is limited by the number of available CPU cores of the system used. In fact, by using process pools, one simulation is assigned to each core of the CPU, making the maximum number of parallel simulations equal to the number of cores in the system(n_{cores}). Furthermore, as in sequential operation the workload of a single simulation can be divided among multiple cores, a single simulation run is slower in parallel operation. To account for this, a core isolation bias δ_{t-core} must be added to the previously defined individual simulation time t_s. This bias is proportional to the computational complexity of the simulation.

Because of the two aforementioned complications, Equation 3.23 no longer holds. Instead, assuming that each simulation has the same simulation time, the equation for the total simulation time becomes as defined in Equation 3.24. This equation takes into account that the number of particles within a generation can be larger than the number of available cores, i.e. $n_p > n_{cores}$, and the core isolation bias δ_{t-core}.

$$t_{s-total} = n_g \cdot \left\lceil \frac{n_p}{n_{cores}} \right\rceil \cdot (t_s + \delta_{t-core}) \quad \text{[s]}. \tag{3.24}$$

3.2.6. (Parallel) L-BFGS-B

The Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Box constraints(L-BFGS-B) is a gradient-based algorithm, meaning that it uses the slope of the objective function to find a local minimum. This single solution based algorithm is considered deterministic, as the same solution is obtained for each run. In addition to using the gradient, the algorithm approximates the hessian, which is the second order partial derivative matrix of the objective function. The hessian provides curvature information, allowing for quick convergence in problems with a single optimum that require a high convergence speed. In problems for which the search space is highly non-convex or discontinuous, this algorithm is less suitable, as it is characterized by its inability to escape local minima.

Since it is not possible to obtain gradients from the objective functions of black-box optimization problems, it is approximated with the finite difference method. In this method, a function is evaluated at two nearby points, after which they are subtracted and divided by the difference of the two points. The method is defined by Equation 3.25, in which the gradient is estimated by perturbing a single variable x with step size ϵ.

$$\frac{\partial f}{\partial x} \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}. \tag{3.25}$$

When dealing with multiple variables, the method is defined as in Equation 3.26, which estimates the whole gradient. In this equation the vector $\mathbf{e_i}$ is the standard basis vector, which makes it possible to perturb a single variable of vector $\mathbf{x}$. Concisely, its value is 1 for the single coordinate to be perturbed, and 0 otherwise.

$$\nabla f(\mathbf{x}) = \frac{\partial f}{\partial x_i} \approx \frac{f(\mathbf{x} + \epsilon_i \mathbf{e}_i) - f(\mathbf{x})}{\epsilon_i} \quad \text{for } i = 1, 2, \ldots, n. \tag{3.26}$$

Each perturbation of the solution represents a perturbation in the variable of a simulation. This means that after the simulation is evaluated for a point, an additional simulation is run for a nearby point to estimate the gradient of that point. The pseudo-code of the L-BFGS-B algorithm in combination with the finite difference method is shown in Algorithm 2.

Algorithm 2 L-BFGS-B Algorithm with finite difference method

```
1: procedure L-BFGS-B($f$, $\mathbf{x_0}$, $maxIter$, $\epsilon$)
2:     Set current point $\mathbf{x} \leftarrow \mathbf{x_0}$
3:     Compute initial gradient $\mathbf{g} \leftarrow$ FiniteDifferenceGradient($f$, $\mathbf{x}$, $\boldsymbol{\varepsilon}$)
4:     Set iteration counter $k \leftarrow 0$
5:     while $k < maxIter$ and not converged do
6:         Compute Hessian approximation $\mathbf{H_k}(\mathbf{g})$
7:         Compute search direction $\mathbf{p_k}(H_k^0)$
8:         Determine step size $\alpha_k(\mathbf{H_k^0})$
9:         Update point: $\mathbf{x}_{\text{new}} \leftarrow \mathbf{x} + \alpha_k \mathbf{p_k}$
10:        Evaluate function at new point: $f_{\text{new}} \leftarrow f(\mathbf{x}_{\text{new}})$
11:        Compute new gradient $\mathbf{g}_{\text{new}} \leftarrow$ FiniteDifferenceGradient($f$, $f_{new}$, $\mathbf{x}_{\text{new}}$, $\boldsymbol{\varepsilon}$)
12:        Update $\mathbf{x} \leftarrow \mathbf{x}_{\text{new}}$
13:        Update $\mathbf{g} \leftarrow \mathbf{g}_{\text{new}}$
14:        $k \leftarrow k + 1$
15:    return $f_{\text{new}}$
16: procedure FiniteDifferenceGradient($f$, $f_{new}$ $\mathbf{x}$, $\boldsymbol{\varepsilon}$)
17:    Initialize gradient vector $\mathbf{g} \leftarrow \mathbf{0}$ of size length($\mathbf{x}$)
18:    for $i = 1$ to length($\mathbf{x}$) do
19:        Create perturbed vector $\mathbf{x}^{(i)} \leftarrow \mathbf{x}$ with $x_i^{(i)} \leftarrow x_i + \varepsilon_i$
20:        $f_i \leftarrow f(\mathbf{x}^{(i)})$
21:        $g_i \leftarrow \frac{f_i - f_{new}}{\varepsilon_i}$
22:    return $\mathbf{g}$
```

f: objective function
$\mathbf{x}_0$: initial point
$maxIter$: maximum number of iterations $\boldsymbol{\varepsilon}$: small perturbation vector for finite difference gradient
$\mathbf{x}$: current point (vector of decision variables)
$\mathbf{g}$: gradient of the objective function f
k: iteration counter
$\mathbf{H_k}$: approximation of the inverse Hessian at iteration k
$\mathbf{p}_k$: search direction at iteration k
α_k: step size at iteration k
$\mathbf{x}_{\text{new}}$: updated point
f_{new}: objective at updated point
$\mathbf{g}_{\text{new}}$: gradient at updated point

L-BFGS-B was implemented by using the *SciPy(1.14.1)* [43] python package was used. The initial solution $\mathbf{x_0}$ is defined by Equation 3.27, the center point of the search space with respect to the upper and lower boundaries $\mathbf{x_u}$ and $\mathbf{x_l}$.

$$\mathbf{x_0} = \frac{\mathbf{x_u} + \mathbf{x_l}}{2}. \tag{3.27}$$

It is crucial to note that, because of the use of the finite difference method (procedure *FiniteDifference-Gradient* of Algorithm 2), each iteration requires an additional simulation evaluation for each variable in **x**. If the time required for one simulation is defined as t_s, the total time required for one iteration of the algorithm $t_{s-iteration}$ is defined by Equation 3.28.

$$t_{s-iteration} = t_s \cdot (1 + length(\mathbf{x})) \quad [\mathrm{s}]. \tag{3.28}$$

To shorten the total simulation time, parallelization was used again. Unlike PSO, L-BFGS-B is single solution based, making parallelization of each iteration impossible. Initially, as the perturbation of each variable in **x** is independent, the finite difference method was parallelized. This meant that the perturbation and corresponding evaluation of each variable was executed in parallel (*line 18* until *line 21* in Algorithm 2). This parallelization is illustrated in Figure 3.11. As a result, the simulation time for each iteration $t_{s-iteration}$ was reduced to the theoretical Equation 3.29, as the evaluation of all perturbations occurred at once.

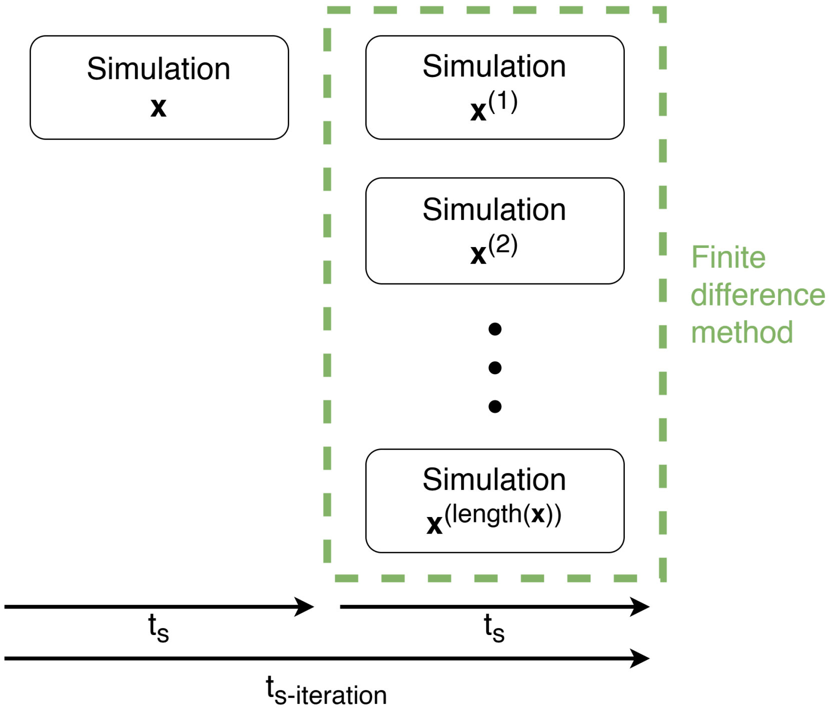


Figure 3.11: One iteration of the parallelized L-BFGS-B algorithm.

$$t_{s-iteration} = t_s \cdot 2 \quad [\mathrm{s}]. \tag{3.29}$$

This, however, is the theoretical simulation time, as the parallelization is limited by the number of cores in the system's CPU. Furthermore, the time for a single parallelized simulation is increased by a bias δ_{t-core} as a consequence of core-isolation. A more realistic estimation of the simulation time per iteration is defined by Equation 3.30

$$t_{s-iteration} = t_s + \left\lceil \frac{length(\mathbf{x})}{n_{cores}} \right\rceil \cdot (t_s + \delta_{t-core}) \quad [\mathrm{s}]. \tag{3.30}$$

While implementing this parallelization method, another implementation of parallelization was identified. L-BFGS-B is known to be inefficient in complex and non-convex search spaces. Furthermore, it is unable to escape local optima. If the algorithm is initialized in a variety of places in the search space, it is possible that one of the instances finds a local optimum that is also the global optimum. Even though the instances do not act as an interdependent population such as in PSO, an optimum can be found by an individual instance.

The initial solutions are either spread evenly over the search space bounded by its constraints or spread randomly. By implementing this parallelization, the simulation time per iteration returns to be represented by Equation 3.28. However, for this implementation, n_p (population size) instances of the algorithm are run in parallel. Therefore, the simulation time of one instance is not reduced, but the search space is explored more thoroughly while theoretically taking an equal amount of computing time. By taking a core isolation bias (δ_{t-core}) into account, the total simulation time is described by Equation 3.31. A single iteration of the resulting algorithm is illustrated in Figure 3.12.

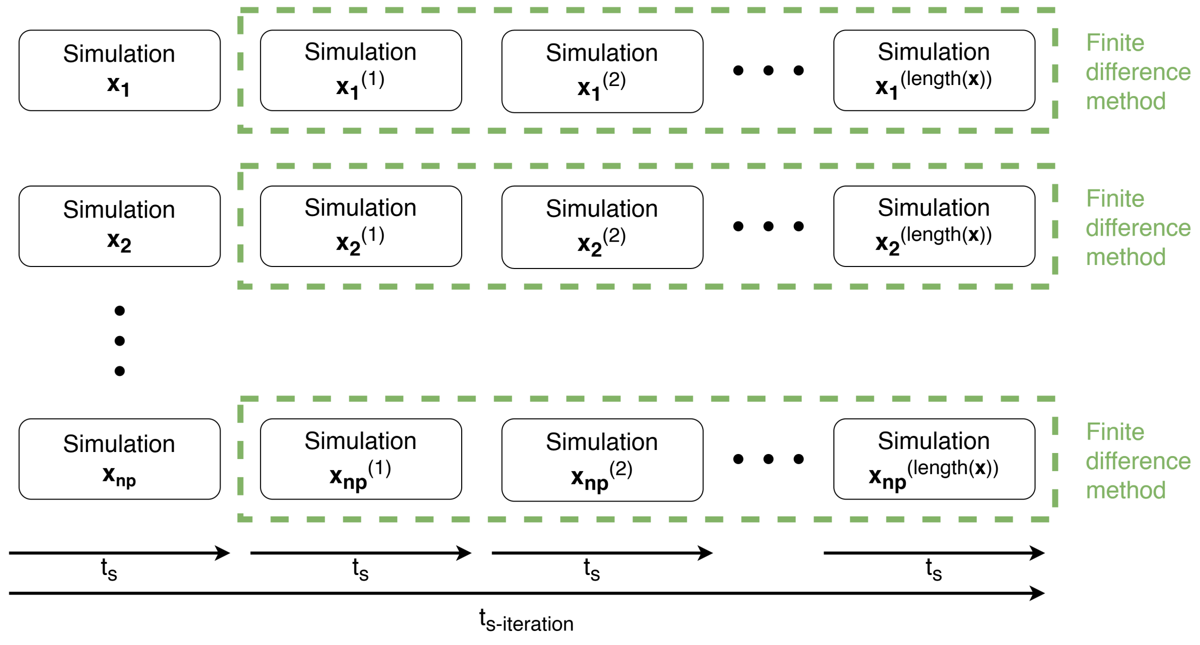

Simulation $\mathbf{x_1}$
Simulation $\mathbf{x_1}^{(1)}$
Simulation $\mathbf{x_1}^{(2)}$
Simulation $\mathbf{x_1}^{(length(\mathbf{x}))}$
Finite difference method
Simulation $\mathbf{x_2}$
Simulation $\mathbf{x_2}^{(1)}$
Simulation $\mathbf{x_2}^{(2)}$
Simulation $\mathbf{x_2}^{(length(\mathbf{x}))}$
Finite difference method
Simulation $\mathbf{x_{np}}$
Simulation $\mathbf{x_{np}}^{(1)}$
Simulation $\mathbf{x_{np}}^{(2)}$
Simulation $\mathbf{x_{np}}^{(length(\mathbf{x}))}$
Finite difference method
t_S
t_S
t_S
t_S
$t_{S\text{-iteration}}$

Figure 3.12: One iteration of the parallelized L-BFGS-B algorithm using the new method. Each row represents an independent instance of the algorithm.

$$t_{s-iteration} = t_s \cdot (1 + length(\mathbf{x})) + \delta_{t-core} \quad [\mathrm{s}]. \tag{3.31}$$

3.2.7. Genetic Algorithm

The final algorithm implemented in this thesis is the Genetic Algorithm (GA). The Genetic Algorithm is an algorithm that is based on the principles of natural selection and evolution. Similar to the PSO algorithm of Section 3.2.5, a population of potential solutions is initialized randomly and evaluated at the start of the GA algorithm. However, the search strategy differs from PSO. Instead of exploring the space by updating all population members directly, GA progresses its exploration by selecting a subset of the population members with a high fitness, called ”parents”. A new generation, called ”offspring”, is generated by applying crossover to the parents and applying mutation to the newly formed members. Crossover combines segments of two parent solutions to form new solutions with a predefined probability. Mutation introduces a probability-based random variation to some of the new solutions to encourage exploration of the search space and to prevent premature convergence. A new population is selected combining a number of best-performing solutions from the previous population (elitism) with the newly generated offspring. Elitism ensures that the best solutions are always preserved across populations. The evolutionary process is repeated until the stopping criterion is met, i.e., the maximum number of iterations is reached or convergence is achieved. During the process all candidate solutions and their fitness are logged for analysis purposes. The pseudo-code for GA is provided in Algorithm 3.

Algorithm 3 GA pseudo-code

```
1: procedure GA($f$, $n_p$, $d$, $maxIter$, $p_c$, $p_m$)
2:     Initialize population $P = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_{n_p}}\}$ with individuals in $d$ dimensions
3:     Evaluate objective function $f(\mathbf{x})$ for each individual $\mathbf{x} \in P$
4:     Set iteration counter $k \leftarrow 0$
5:     while $k < maxIter$ AND (not converged) do
6:         Select mating pool $M$ from $P$
7:         Initialize offspring population $O \leftarrow \emptyset$
8:         for each pair $(\mathbf{x_a}, \mathbf{x_b})$ in $M$ do
9:             Generate random number $r \sim \mathcal{U}(0, 1)$
10:            if $r < p_c$ then
11:                Apply crossover to $(x_a, x_b)$ to produce $(\mathbf{o_1}, \mathbf{o_2})$
12:            else
13:                Set $(\mathbf{o_1}, \mathbf{o_2}) \leftarrow (\mathbf{x_a}, \mathbf{x_b})$
14:            Mutate $\mathbf{o_1}$ and $\mathbf{o_2}$ with probability $p_m$
15:            Add $\mathbf{o_1}$ and $\mathbf{o_2}$ to $O$
16:        Evaluate $f(\mathbf{x})$ for each $\mathbf{x} \in O$
17:        Select new population $P$ from current $P \cup O$ (e.g., elitism)
18:        $k \leftarrow k + 1$
19:    return best individual in $P$
```

f: objective function
n_p: population size
d: dimensions (number of decision variables)
$maxIter$: maximum number of generations
p_c: crossover probability
p_m: mutation probability
P: current population
k: iteration counter $\mathbf{x_i}$: solution in a population
M: mating pool
O: offspring population
$(\mathbf{x_a}, \mathbf{x_b})$: mating pair
$(\mathbf{o_1}, \mathbf{o_2})$: offspring pair
r: random number

The Genetic Algorithm was implemented by using *Python*’s *pymoo* package (version *0.6.1.3*) [42]. This is the same package that was used for the implementation of PSO.

Similar to PSO, the algorithm was parallelized by evaluating each solution of the same generation in parallel. The total simulation time $t_{s-total}$ of GA is therefore also represented by Equation 3.24 and the parallelization is also illustrated as in Figure 3.10. However, it must be kept in mind that for GA the evaluations within the same row are not necessarily dependent on each other, but rather individual evaluations.

3.3. Evaluation scenarios

To assess the optimization framework's performance and generalizability, three distinct fictional scenarios that reflect real-world situations were employed. In the first scenario a hydrogen production facility was modeled. To maximize system efficiency and cost, the buffer size was optimized. The second developed scenario represented a small neighborhood energy hub, of which continuous operational parameters costs were optimized. The third scenario represented an electric vehicle charging station. For this scenario the charging schedule was optimized, resulting in a discrete planning problem. The selection of these scenarios and their associated optimization problems covered the hydrogen and electricity domains, the design, operational, and planning domains, and the continuous and discrete domains. The scenarios are described in more detail in the following sections.

3.3.1. Hydrogen production facility scenario

The hydrogen system was composed of five models:

1. Electrolyzer;
2. Buffer (100 kg capacity);
3. Compressor;
4. Demand;
5. Hydrogen controller.

The electrolyzer is responsible for the production of hydrogen gas from electricity. After production, the hydrogen gas is temporarily stored in a buffer, which operation is managed by a controller. The compressor ensures that the hydrogen is at the right pressure for transport, which is modeled by a demand model. The resulting system is illustrated in Figure 3.13. Given that the controller is not managing the demand and supply of the system, the system must be designed to handle potential imbalances. The imbalance between supply and demand is solved in two ways. First, the excess hydrogen is stored in the buffer and released when there is a demand surplus. However, in case of a production surplus while the buffer has reached its full capacity, the excess is flared. This causes a loss of potential earnings and should be avoided as much as possible. The scenario was simulated for 1 week in time steps of 15 min.

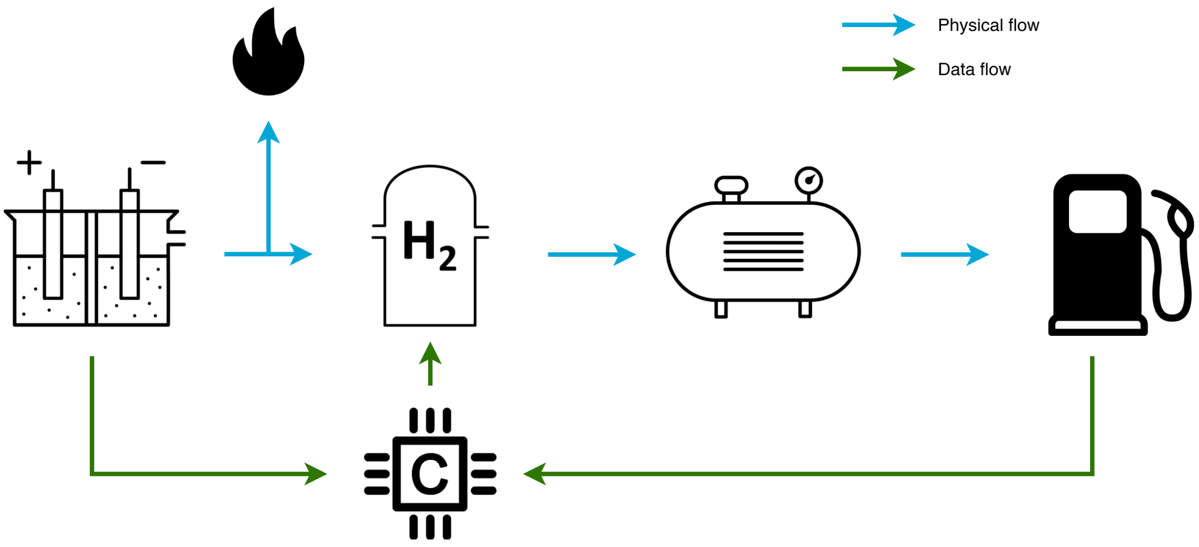


Figure 3.13: The hydrogen production facility scenario consisting of an electrolyzer, a buffer, a compressor, a demand, and a buffer controller.

The supply and demand data were artificially created using random processes. To imitate a pattern that is representative of an electrolyzer, the production data was generated while taking into account the following characteristics:

- Based on normal distributions, varying per time of day, the production rate is determined for each timestep;
- The electrolyzer has a base production of 0.75 kg/15min.

The demand pattern is generated according to the following characteristics:

- Based on normal Gaussian distributions, the demand rate is around 2 times higher on weekdays, and there is almost no demand outside of peak hours;
- The demand is scaled to achieve a total production that is equal to 108% of the total demand in a month.

The resulting production and demand patterns are illustrated in Figure 3.14.

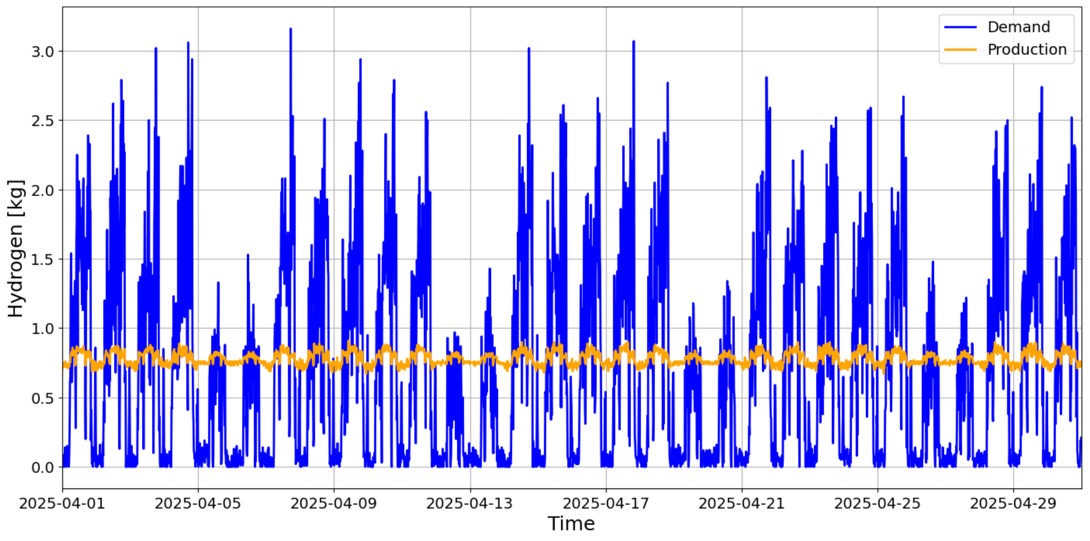


Figure 3.14: The generated hydrogen production and demand data patterns.

The imbalance that can be observed in the previous figure and the requirement to reduce flaring cause the need for an efficient setup. The most important parameter that is related to these system requirements is the buffer capacity, also referred to as buffer size. In theory, an infinitely large buffer size is optimal, as this allows a large bandwidth of stored hydrogen, making it easier to meet the demand. Furthermore, a large buffer size makes it possible to store more hydrogen, avoiding the necessity of flaring the excess production. However, the investment cost of hydrogen buffers is directly related to their size, contradicting the theoretical optimum. To summarize, the buffer size must be kept minimal to avoid high investment costs, but large enough to be able to optimally make use of the buffer. Because of this duality, a system design optimization problem arose.

In the case of this scenario, the optimization problem consists of a single decision variable, namely the buffer size. The buffer size can take on any continuous value, and therefore the problem is considered a single-variable continuous optimization problem. The cost function for this problem consists of two parts: the buffer size x itself and the utilization of the buffer capacity. To take the utilization of the capacity into account, the number of time steps the buffer state of charge reaches 0% or 100% in the simulated time frame is summed. This sum is defined as the number of SoC violations $n_{SoC,vio}$ and is number is determined by Equation 3.32.

$$n_{SoC,vio} = \sum_{t=1}^{T} \mathbf{1}_{\{SoC_t \leq 0\}} + \mathbf{1}_{\{SoC_t \geq 100\}}. \tag{3.32}$$

To combine these two parts of the cost function, a large penalty that is proportional to $n_{SoC,vio}$ is added for each SOC violation to buffer size x. This penalty is specified by λ, and is an order of magnitude larger than the buffer size, specifically 1000. The resulting objective function is defined in Equation 3.33.

$$\min_{x \in \mathcal{X}} \quad x + \lambda \cdot n_{SoC,vio}. \tag{3.33}$$

3.3.2. Energy hub scenario

The energy hub system consisted of the following models:

1. Five household demands;
2. Five PV models (60 m^2 each);
3. Central battery (100 kWh);
4. Central grid connection;
5. Battery controller.

The five households in the neighborhood of this scenario are supplied by five centrally connected PV modules. The excess power of the PV modules is either fed back into the grid or stored in a central battery. When the PV modules do not provide enough power to meet the demand of the households, power is either extracted from the grid connection or from the battery. In this scenario the assumption is made that each individual household is interested in collaborating with the neighbors to achieve solid power reliability and collective financial efficiency. The use of a central battery and controller consolidates this collaboration, as all households influence the operation of these models. The scenario was simulated in time steps of 15 min, and the total time simulated was 1 week. The schematic representation of the system is illustrated in Figure 3.15.

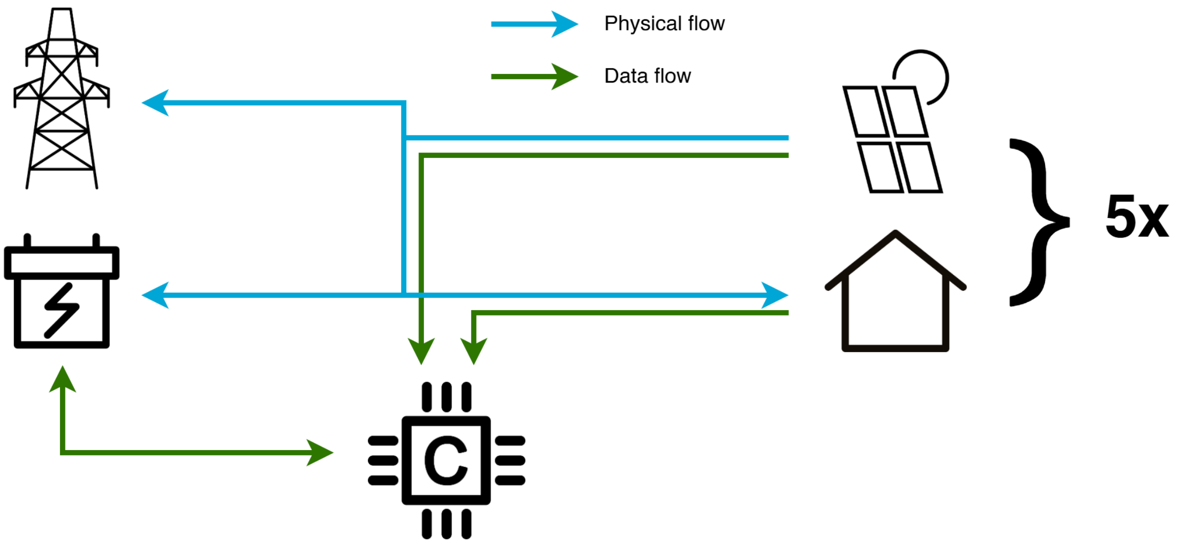


Figure 3.15: The neighborhood scenario consisting of five households, five solar systems, a central battery, a grid connection, and a battery controller.

To implement a realistic scenario, an existing dataset for household demand [44] was used. The dataset contains missing values, which were filled by extrapolating values from previous data points from the same dataset. Furthermore, since the dataset has an accuracy of 1 min while the scenario requires a 15 min-accuracy, the dataset was converted to the correct resolution by averaging the data points for every 15 min interval. Five distinct demand patterns were generated, by randomizing the data four times using a normal distribution (standard deviation = 5%). A dataset containing irradiance data (GHI, DHI, DNI), temperature, and solar positional data (height, azimuth angle, fill factor) was used for the PV models [8]. The five PV systems were assumed to be of the same size, namely 60 m^2, resulting in a total area of 300 m^2. The total demand and solar generation of the neighborhood over the course of a week is plotted in Figure 3.16.

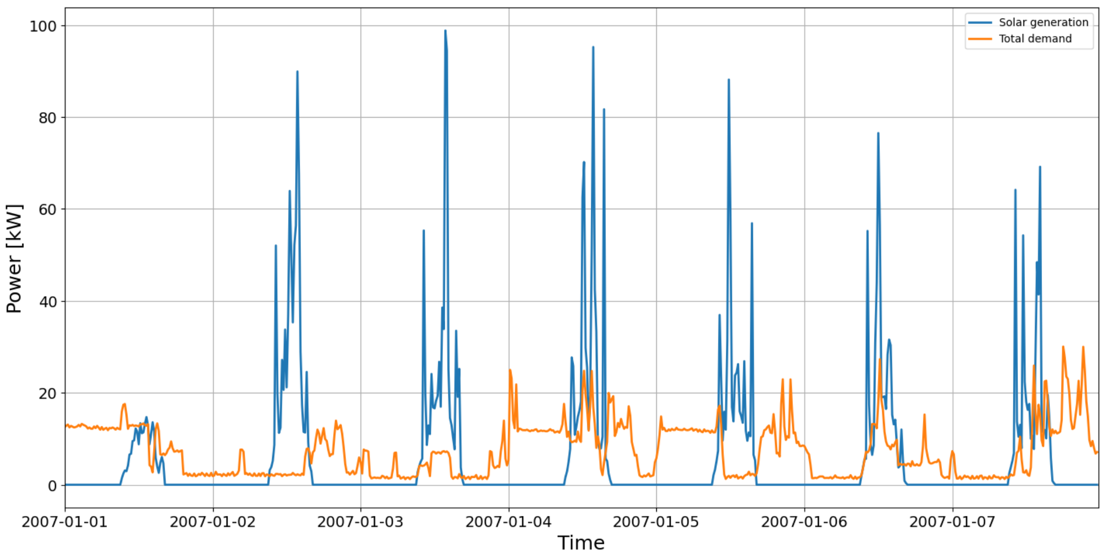


Figure 3.16: The neighborhood demand and solar power generation over the course of a week.

Similarly to the hydrogen scenario (Section 3.3.1), there is a mismatch between energy production and demand. Again, the energy storage component of the system and its control are the two most important components of the system. This time however, the problem is approached from an operational perspective. To do this, the control of the battery was programmed to be dependent on the intraday electricity market, in which the electricity price can vary every 15 min. The intraday market prices this resolution were obtained from a publicly available dataset provided by the Dutch TSO TenneT [45]. The control based on this dataset was done by using two parameters:

- **Upper threshold** θ_u **[€/kWh]**: In case the market price is higher than this threshold, feeding the surplus energy back to the grid is prioritized over charging the battery;
- **Lower threshold** θ_l **[€/kWh]**: In case the market price is lower than this threshold, the energy shortage is resolved by pulling electricity from the grid, rather than discharging the battery.

The market price dataset consists of two subsets: shortage price $\mathcal{P}_{short}$ and surplus price $\mathcal{P}_{surp}$. The shortage price indicates the price the grid user is required to pay to the TSO when it extracts energy from the grid. The surplus price defines the price that the TSO is required to pay to the grid user when the user delivers energy back into the grid. The two price subsets are defined by Equation 3.34 and Equation 3.35 respectively, in which p_t is the settlement price at 15 min-interval t, and T is the last simulated interval.

$$\mathcal{P}_{short} = \{p_t^{short}\}_{t=1}^{T} \quad [\text{€/kWh}], \tag{3.34}$$

$$\mathcal{P}_{surp} = \{p_t^{surp}\}_{t=1}^{T} \quad [\text{€/kWh}]. \tag{3.35}$$

During the simulation of this scenario, the power flow between the grid and the neighborhood P_{grid} is monitored for each timestep. The flow is determined by the decisions made by the controller which are mainly dependent on the other models in the system. These dependencies are complex, and thus P_{grid} is obtained by simulating the system. The controller decisions are dependent on θ_u and θ_l. As the optimization of this scenario is approached from an operational perspective dependent on these parameters, the grid power is defined as $P_{grid}(\theta_u, \theta_l)$. Depending on the direction of the grid power flow at timestep t, i.e., whether $P_{grid,t}(\theta_u, \theta_l)$ is positive or negative, the neighborhood either receives p_t^{short} or p_t^{surp}. Consequently, the total cost C_{total} can be obtained by taking the sum of all transactions in the simulated time frame. The total cost C_{total} can then be optimized with respect to the two thresholds θ_u and θ_l. This represents the total combined price the members of the neighborhood must either pay or receive at the end of a week. The resulting cost function is defined by Equation 3.36, which is subject to the constraints defined by Equation 3.37 and 3.38.

$$\min_{\theta_u, \theta_l \in \Theta} \quad C_{total} = \sum_{t=1}^{T} \begin{cases} -P_{grid,t}(\theta_u, \theta_l) \cdot p_t^{\text{surp}}. & \text{if } P_{grid,t}(\theta_u, \theta_l) > 0, \\ -P_{grid,t}(\theta_u, \theta_l) \cdot p_t^{\text{short}}. & \text{if } P_{grid,t}(\theta_u, \theta_l) \leq 0, \end{cases} \tag{3.36}$$

$$\text{s.t.} \quad \min(\mathcal{P}_{\text{short}}) \leq \theta_l \leq \max(\mathcal{P}_{\text{short}}), \tag{3.37}$$

$$\min(\mathcal{P}_{\text{surp}}) \leq \theta_u \leq \max(\mathcal{P}_{\text{surp}}). \tag{3.38}$$

3.3.3. Electric vehicle (EV) charging station

The third scenario that was developed for this thesis represented a charging station for electric vehicles. This energy system consisted of the following models:

- Five EV models (60 kWh battery capacity);
- PV model (240 m^2);
- wind turbine model (300 W rated power);
- charging controller.

The charging station featured 5 charging bays. When one or more EVs are present at the station, they are available to be charged with power obtained from the PV installation, the wind turbine, and the grid. Based on this presence, the charging controller determines when the charging cycle of each EV is started. The simulations of this system represented a single day and had a time resolution of 15 min. The schematic representation of the system is provided in Figure 3.17.

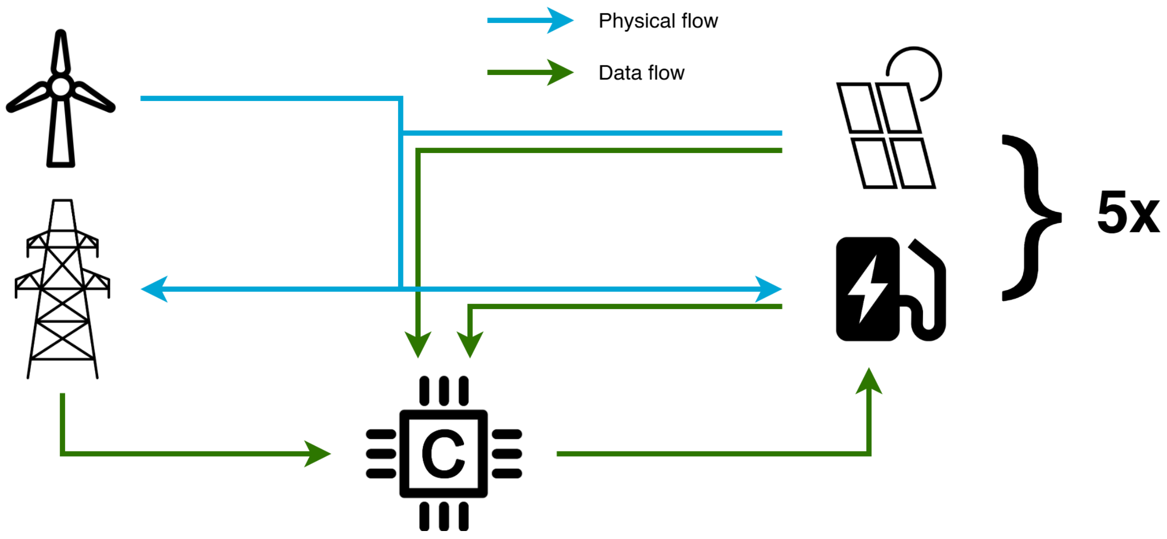


Figure 3.17: The schematic representation of the EV charging station scenario implemented in the Illuminator. The blue arrows indicate physical flows, while the green arrows represent data flows between models.

In this scenario the EV models were assumed to have a constant and equal charging profile of 11 kW. Furthermore, the EV batteries were set to have a capacity of 60 kWh. As a consequence, each charging cycle takes approximately 5.5 hours ($t_{charging} \approx 5.5$ h). The EVs charging availability is based on the generated presence schedule illustrated in Figure 3.18. This schedule can be represented by the arrival times $\mathbf{t_{arrival}}$ and departure times $\mathbf{t_{departure}}$.

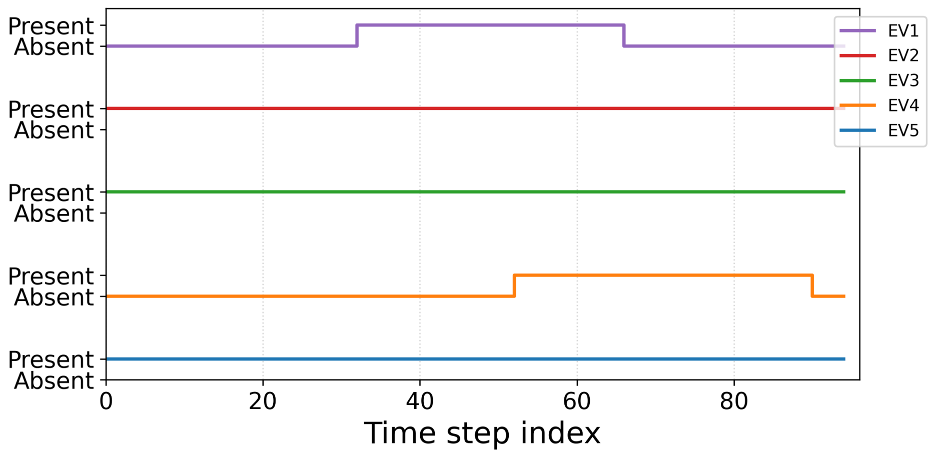


Figure 3.18: Presence of the EVs at the charging station.

The same solar irradiance data as in the previous scenario was used, but for this scenario a single model with a total area of 240 m^2 was implemented. The wind turbine model was set to have a diameter of 2 m and a rated power of 300 W.

For the operation of the charging station, of which the loads can be substantial, it is important to consider the strain on the grid connection. The reason for this is to mitigate the risk of grid congestion, which, as explained in the introduction chapter, is a growing challenge associated with the energy transition.

To reduce the maximum load on the grid connection, the charging schedule of the five electric vehicles can be optimized. By optimally scheduling the starting times of the charging cycles, the maximum power drawn or fed back into the grid can be minimized. The decision variables for this problem are therefore the starting time-steps τ_1^0, τ_2^0, τ_3^0, τ_4^0, and τ_5^0, vectorized as $\tau^{\mathbf{0}}$. The problem's search space is constrained by the EV presence pattern presented in Figure 3.18 minus the charging time $t_{charging}$, as the EVs need to be fully charged before departure. The cost function is thus defined as in Equation 3.39, subject to the constraint of Equation 3.40.

$$\min_{\tau^{\mathbf{0}} \in \mathcal{T}} \quad P_{grid,peak} = \max(\{P_{grid,t}(\tau^{\mathbf{0}})\}_{t=1}^{T}) \tag{3.39}$$

$$\text{s.t.} \quad \mathbf{t_{arrival}} \leq \tau^{\mathbf{0}} \leq \mathbf{t_{arrival}} - t_{charging} \tag{3.40}$$

4 Results

This chapter presents the results of the experiments performed on the scenarios from Section 3.3. For each scenario, the problem is reintroduced, after which the three algorithms are applied to the scenario.

To test the consistency of each algorithm, each algorithm is was applied three times to each scenario, each time using a different randomized set of initial solutions. Furthermore, the gradient-based algorithm (parallel L-BFGS-B) was applied an additional time with a set of initial solutions distributed equidistantly over the search space. The best resulting run of each algorithm was used to produce the plots for each algorithm application.

The maximum number of iterations was set to 100 for all cases. For PSO and GA the tolerances that indicate convergence were set to 0.001 for both the function and the variables. The *period* variable, which determines for how many generations the tolerances must be met, was set to 5. The number of parallel processes was set to 9. Therefore, the population-based algorithms used a population size of 9 and the parallelized gradient-based algorithm consisted of 9 parallel instances. For PSO the initial weight was set to 0.9 and the two coefficients were both set to 1.5. However, adaptive weights were enabled in the *pymoo* package, meaning that the weights changed dynamically over time.

4.1. Hydrogen production facility scenario

Scenario 1, as described in Section 3.3.1, involved a hydrogen system consisting of an electrolyzer, a buffer, a compressor, and a refueling station. The problem related to this scenario aimed to minimize the buffer size. Therefore, the optimal solution corresponded to the lowest cost.

In the following sections the results of the three algorithms are presented and briefly compared. In the figures that illustrate the explored solutions in the search space, the decimal logarithm of the cost is used. This is because the penalty applied to the cost function is significantly larger than the buffer size, making the figures less understandable if the standard cost is used.

4.1.1. PSO

First, the PSO algorithm was applied to scenario 1. The solutions that were explored by the PSO algorithm are illustrated in Figure 4.2. From this figure, it can be observed that the buffer size at which the best cost is obtained is around 420 kg. Furthermore, the figure shows that the most evaluated solutions are equal to or larger than this found optimum. Additionally, it can be observed that the search space contains a discontinuity close to this optimum, which can be explained by the use of the penalty. In other words, buffer sizes lower than the optimum receive a penalty for being too small.

Table 4.1 presents the results of these three runs with different initial solution distributions. In this table, the number of generations, the run time, the solution, the lowest cost, and the gap to the best-found solution of each run are listed. It can be observed that the solutions are close to 420 kg, and the best-found solution is 419.91 kg. In all runs, the cost had the same value as the solution, as in these cases no penalty was applied. The algorithm was consistent among runs as the largest gap was

0.223%. The run time for each run is similar, but the run with ID 42 was the quickest (282.9 s). It can be concluded that all runs had similar performance in terms of run time and solution quality.

Table 4.1: Simulation optimization results for the PSO algorithm on scenario 1. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Solution [kg]	Cost [kg]	Gap [%]
42	9	**282.9**	420.32	420.32	0.097
33	9	324.6	419.91	**419.91**	**0**
1	9	337.2	420.85	420.85	0.223

4.1.2. Parallel gradient-based algorithm

The second algorithm that is applied to this scenario is the parallel gradient-based algorithm (parallel L-BFGS-B). The solutions explored by the best run of this algorithm are illustrated in Figure 4.3. From the figure it can be observed that the explored solutions are spread over the search space, with a dense cluster around the 420 kg mark. Except for this, the explored solutions are similar to the ones of PSO.

From Table 4.2 it can be observed that the best solution found by this algorithm is 419.95 kg. The algorithm shows some inconsistency as the gap is between 0.010% and 0.526%. The solutions are however still close to the optimal. The quickest run had a run time of 1380.2 s but had the worst solution of the four runs.

Table 4.2: Simulation optimization results for the parallel L-BFGS-B algorithm on scenario 1. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Solution [kg]	Cost [kg]	Gap [%]
-	34	1899.1	420.73	420.73	0.196
42	32	1678.2	419.95	**419.95**	**0.010**
33	32	1805.6	421.42	421.42	0.359
1	22	**1380.2**	422.12	422.12	0.526

4.1.3. GA

The last algorithm that was applied to scenario 1 was GA. In Figure 4.4 it can be observed that GA mainly explored solutions close to optimal value, while having a less distributed exploration space. In fact, almost no solutions above 475 kg were explored.

Table 4.3 concludes that the best solution found by the GA is 419.92. Although the differences between the three runs are small, this run was also the quickest, with a run time of 362.8 s. The gap showed a small variability in the calculated gaps and the solution with the best gap had a gap of 0.003%.

Table 4.3: Simulation optimization results for the GA algorithm on scenario 1. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Solution [kg]	Cost [kg]	Gap [%]
42	12	378.1	420.03	420.03	0.029
33	12	398.5	421.13	421.13	0.291
1	11	**362.8**	419.92	**419.92**	**0.003**

4.1.4. Comparison of the algorithms

In this section, the performances of the three algorithms that were applied to the optimization problem of scenario 1 are compared.

Figure 4.1 shows the convergence curves of the three algorithms. It can be observed that the gradient-based-algorithm (parallel L-BFGS-B) required substantially more iterations than the other algorithms to achieve convergence. Additionally, it was terminated many iterations later. It can also be observed that the global best cost of PSO was found after 4 iterations, while only 1 iteration was required to make an improvement to the initial solution set. Furthermore, the figure shows that the PSO terminates first, and is followed two iterations later by GA.

The characteristics of the application of each algorithm to the problem of this scenario are listed in Table 4.12. In this table, it can be observed that PSO shares its best average cost with GA. Although the difference is small, the shorter average time of PSO makes it the best performing algorithm to this problem. The gradient-based algorithm showed to have the worst performance indicators compared to the other two algorithms. Although the average gap was low, it was a factor 2 larger compared to the other two algorithms.

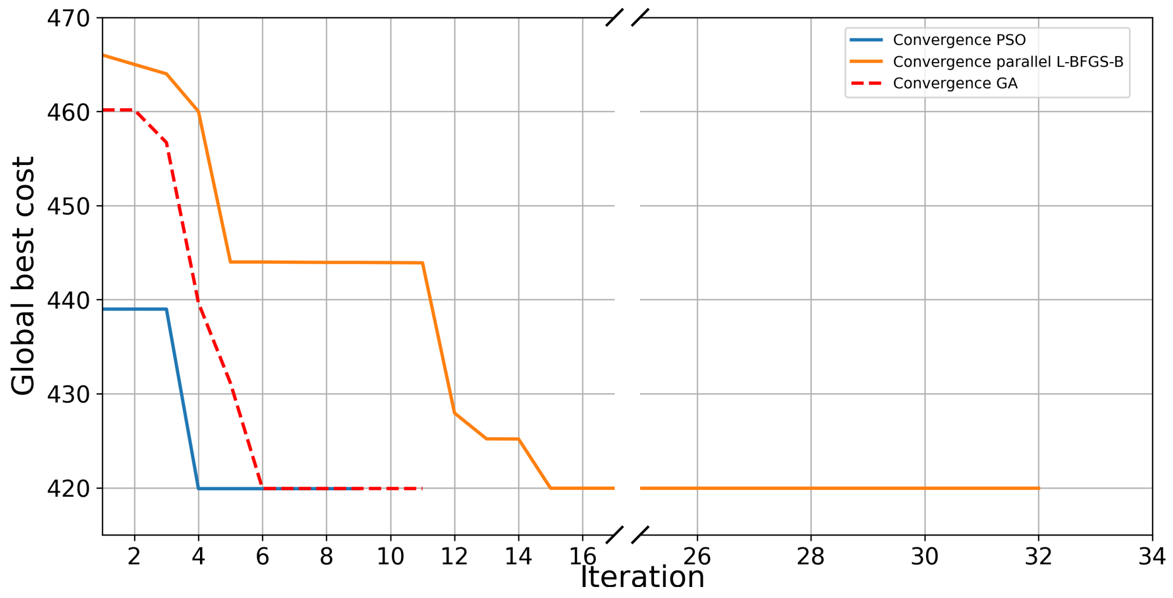


Figure 4.1: Convergence of all three algorithms applied to scenario 1.

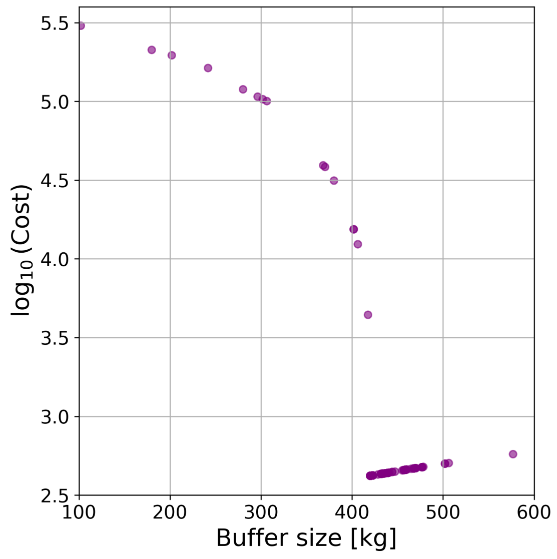


Figure 4.2: Explored search space of the best run of the PSO algorithm for buffer size optimization.

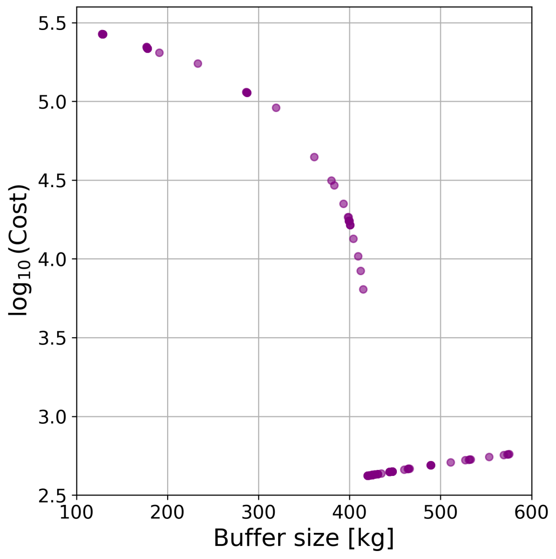


Figure 4.3: Explored search space of the best run of the L-BFGS-B algorithm for buffer size optimization.

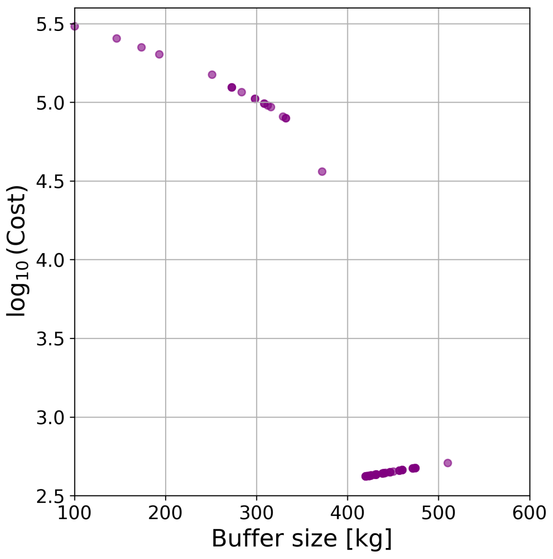


Figure 4.4: Explored search space of the best run of the GA for buffer size optimization.

Table 4.4: Simulation optimization results of all algorithms applied on scenario 1. The shortest average run time, the best average cost, the absolute best cost, and the smallest average gap are highlighted.

Algorithm	**Avg. iteration**	**Avg. time** [s]	**Avg. cost** [kg]	**Best cost** [kg]	**Avg. gap** [%]
PSO	9	**314.88**	**420.36**	**419.91**	**0.107**
L-BFGS-B	30	1690.78	421.05	419.95	0.273
GA	11.67	379.77	**420.36**	419.92	**0.107**

4.1.5. Optimization result

To show the result of the optimization, the scenario is run two times, one time with a large buffer size of 500 kg, and one time a buffer size that is found by the best run of PSO (419.91 kg). The state of charge of these simulations are Illustrated in Figure 4.5 and Figure 4.6.

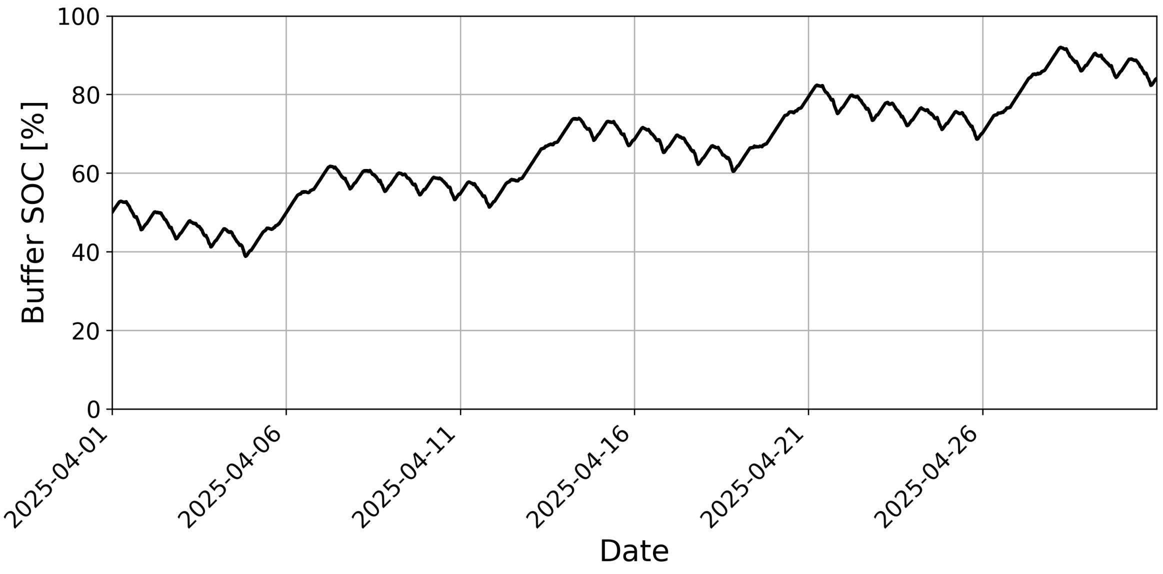


Figure 4.5: A simulation of scenario 1 in which the buffer size is not optimized (500 kg).

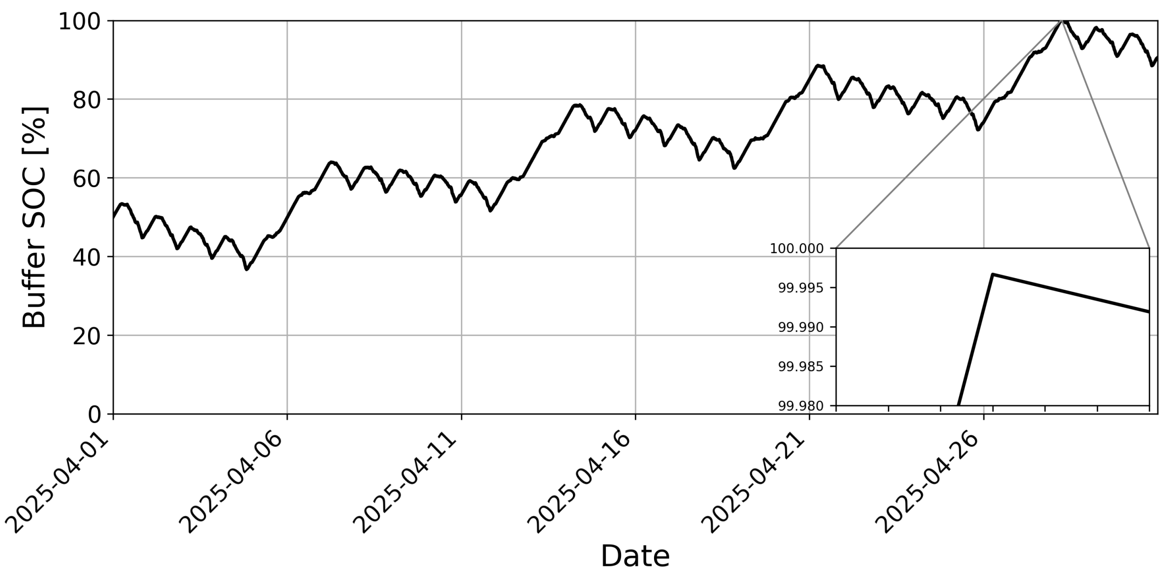


Figure 4.6: A simulation of scenario 1 in which the buffer size is optimized (419.91 kg).

Figure 4.5 shows that the state of charge does not reach 100% by a lot. From this it can be deduced that the buffer size can be decreased. In fact, in the optimal case illustrated in Figure 4.6 it can be observed that a state of charge of 100% is almost reached, implying a more optimal utilization of the buffer.

4.2. Energy hub scenario

As explained in Section 3.3.2, scenario 2 represents the grid in a small neighborhood, containing five households, five PV installations, a central battery, and a central grid connection. The cost function of the optimization problem associated with this scenario represents the total cost for the use of the grid over the course of a week. The objective is to minimize the cost. A negative cost implies that the neighborhood earns money from selling energy back to the grid and buying energy at an advantageous price. Therefore, a low or even negative total cost is favorable. The cost function is highly dependent on two variables that indicate at which settlement price the energy should be bought or sold back from the grid. Since two variables are optimized, the figures illustrating the explored solutions in the search space have three dimensions. The two axes each represent a variable, while the cost is indicated by a color.

4.2.1. PSO

The first algorithm that was applied to the optimization problem of scenario 2 is the PSO algorithm. In Figure 4.8 it can be observed that the optimal solution of the problem found by the PSO algorithm is the central part of the search space. In fact, a cluster of solutions with a low cost-value can be observed in the middle of the figure. In the figure it can also be observed that the solutions of which the lower threshold is between €0/kWh and €0.25/kWh have a higher cost compared to the other solutions. From this it can be concluded that the solution quality is dominated by the lower threshold variable.

The contribution of each variable and the dominance of the lower threshold variable is visible in Figure B.2 and B.3 in Appendix B.2. In fact, it can be observed that a larger standard deviation is present in the cost of the upper price variable, compared to the lower price variable.

The PSO algorithm was applied three times to the optimization problem of scenario 2. The results obtained from these three runs are presented in Table 4.5. From this table it can be concluded that the run with ID 42 yielded the lowest cost, namely €−60.51. The solution that resulted in this cost has an upper threshold of €0.117/kWh and a lower threshold of €0.111/kWh. The calculated gaps of the runs are low. The number of iterations and the total time that was required to obtain the solutions are similar for each of the three runs. The quickest run time was 829.9 s.

Table 4.5: Simulation optimization results for the PSO algorithm on scenario 2. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Solution [€ /kWh]	Cost[€]	Gap [%]
42	20	**829.9**	0.117, 0.111	**-60.51**	**0**
33	21	1074.2	0.084, 0.127	-60.14	0.612
1	19	944.7	0.123, 0.106	-60.22	0.478

4.2.2. Parallel gradient-based algorithm

After the PSO algorithm was applied to the second scenario, the parallelized L-BFGS-B algorithm was implemented and evaluated. The step size ϵ of the finite difference algorithm was set to 0.1. The resulting search space exploration plot is illustrated in Figure 4.9. The first thing that can be noticed, is that a diagonal line of solutions appears across the search space plot. This line can be explained by the distribution of initial solutions in the best performing run of this algorithm. In fact, in this run the initial solutions were distributed equidistantly across the search space. All these initial solutions were between the upper and lower bounds of the search spaces, determined by the maximum and minimum of the used price dataset [45]. The optimal solution found by this algorithm is found at the center of the plot. For conciseness, the plots of the individual costs of the solutions values are attached in Appendix B.2 and can be seen in Figure B.4 and Figure B.5.

The parallel L-BFGS-B algorithm was applied four times to the optimization problem of scenario 2. The results for each run are listed in Table 4.6. From this table it becomes clear that the best result was obtained when the initial solutions were spread equally over the search space, resulting in a cost of € −54.99. This cost was found for an upper threshold of € 0.123/kWh and a lower threshold of € 0.044/kWh required the largest number of iterations and also had the longest run time. The quickest run had a run time of 923.3 s but resulted in the second worst cost. The gaps calculated for these runs were large, with the largest gap being 66.985% and the smallest being 9.129%.

Table 4.6: Simulation optimization results for the L-BFGS-B algorithm on scenario 2. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Solution [€ /kWh]	Cost[€]	Gap[%]
-	45	3127.1	0.123, 0.044	**-54.99**	**9.129**
42	26	1807.8	0.607, 0.333	-19.98	66.985
33	18	2645.9	0.627, 0.074	-42.75	29.356
1	11	**923.3**	-0.054, 0.383	-24.67	59.236

4.2.3. GA

The final algorithm that was applied to the second scenario was GA. The plot for the explored solutions in the search space is illustrated in Figure 4.10. From this figure it can be observed that there is a high density of solutions at the center of the plot.

As for the other algorithms applied to this scenario, the cost of the individual explored parameters can be found in Appendix B.2 (Figure B.6 and B.7).

The three runs of the GA on scenario 2 resulted in the characteristics listed in Table 4.7. The best performing run in terms of cost required 19 iterations and took the least amount of time. With a threshold of € 0.117/kWh and a lower threshold of € 0.103/kWh, the cost was € −60.19. Although the costs found in each run were close to each other, and thus also their gaps, the run times showed a larger variance.

Table 4.7: Simulation optimization results for the GA algorithm on scenario 2. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Solution [€/kWh]	Cost [€]	Gap [%]
42	32	1758.3	0.079, 0.125	-60.09	0.699
33	26	820.4	0.089, 0.126	-60.16	0.575
1	19	**615.7**	0.117, 0.103	**-60.19**	**0.521**

4.2.4. Comparison of the algorithms

This section compares the results obtained by applying each of the three algorithm to the optimization problem related to scenario 2.

By comparing the convergence curves of the three algorithms in Figure 4.7, it can be observed that the gradient-based algorithm required more iterations to achieve a stable result, compared to the other two algorithms. In fact, the gradient-based algorithm required 25 iterations to converge, and terminated at iteration 45. The GA and the PSO algorithm only required 14 and 15 iterations respectively, and both terminated 5 iterations later. Additionally, it can be observed that the GA made a large improvement regarding the global best solution in the first three generations. Conversely, the convergence rates of the other two algorithms were slower in the initial phase.

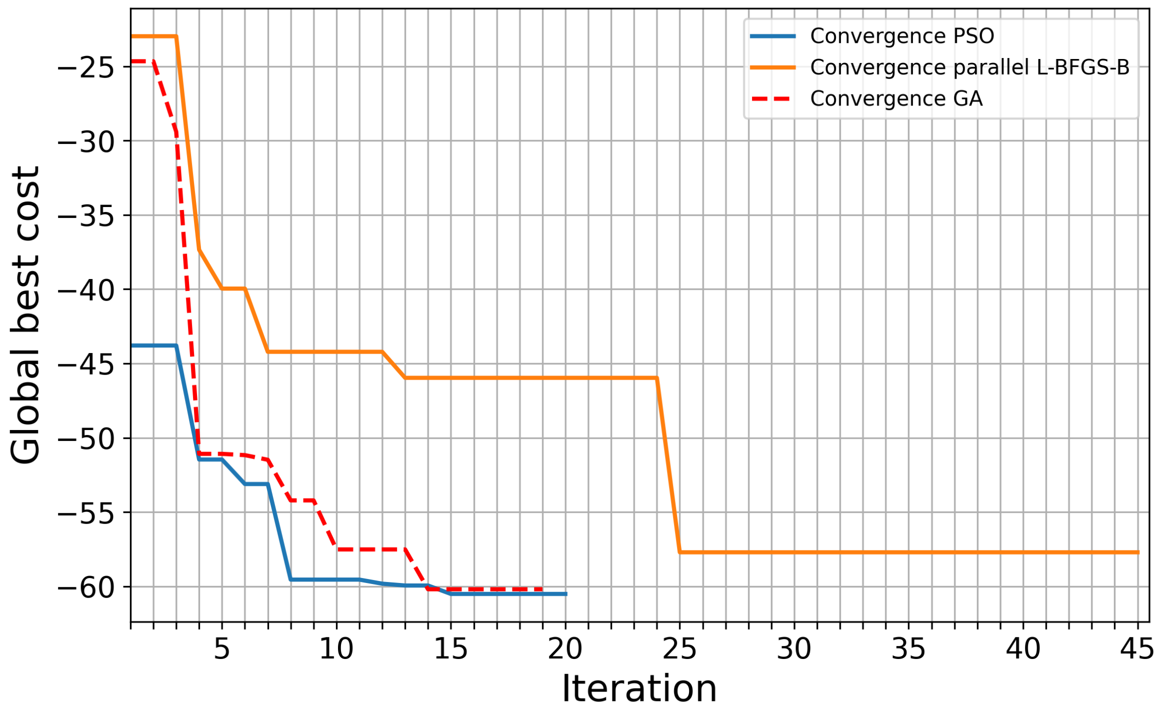


Figure 4.7: Convergence of all three algorithms applied to scenario 2.

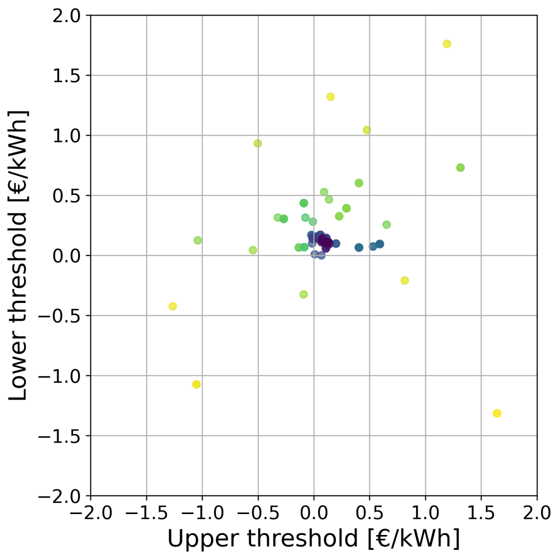


Figure 4.8: Explored search space of the best run of the PSO algorithm for threshold optimization.

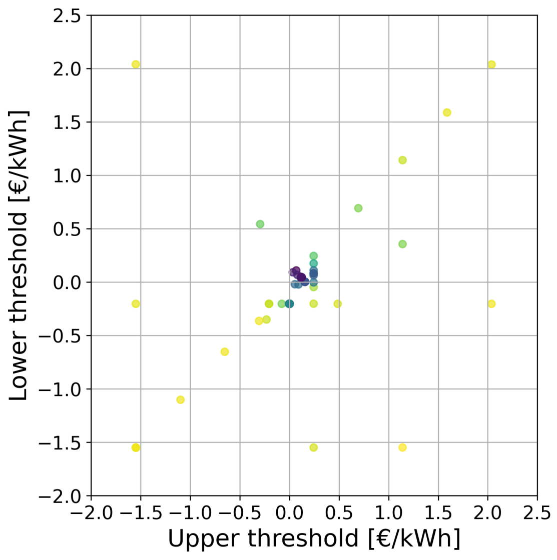


Figure 4.9: Explored search space of the best run of the L-BFGS-B algorithm for threshold optimization.

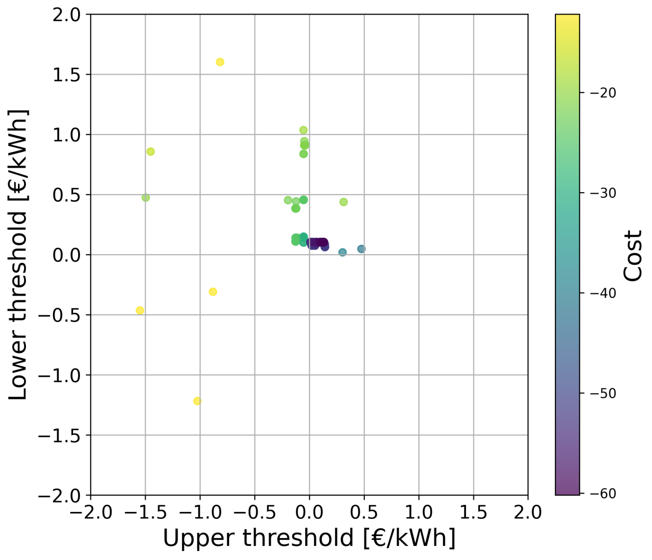


Figure 4.10: Explored search space of the best run of the GA for threshold optimization.

In Table 4.8, the characteristics of the applied algorithms are listed. It can immediately be observed that the PSO algorithm performed the best in terms of time and solution quality. In fact, the PSO algorithm had the best average iteration count, run time, and cost, while also achieving the best overall cost. While the performance of the GA is relatively close to that of the PSO algorithm, the run time and quality of the parallel L-BFGS-B stands out. The average time is significantly higher and the average cost is almost a factor 2 worse than the cost obtained by the other algorithms. This is reflected in the average gap, which is a factor 100 larger. In fact, it seems that the algorithm was not able to achieve a solution close to the true optimum.

Table 4.8: Simulation optimization results of all algorithms applied on scenario 2. The shortest average run time, the best average cost, the absolute best cost, and the smallest average gap are highlighted.

Algorithm	Avg. iteration	Avg. time [s]	Avg. cost [€]	Best cost [€]	Avg. gap [%]
PSO	20	**949.60**	**-60.29**	**-60.51**	**0.363**
L-BFGS-B	25	2126.03	-35.59	-54.99	41.177
GA	25.67	1064.77	-60.15	-60.19	0.598

4.2.5. Optimization result

To evaluate whether the optimization of this scenario was successful, a comparison is made between a simulation with suboptimal and optimized thresholds. The suboptimal case is depicted in Figure 4.11, while Figure 4.12 depicts the optimized case. The figures are structured in the same way. The first plot illustrates the power flows of the demand, grid usage, and PV generation. The second plot illustrates the variable prices and the thresholds applied to the case. Finally, the third plot illustrates the state of charge of the central battery throughout the week as a consequence of the thresholds. It is important to note that the grid power usage is negative in the figure when power is pulled from the grid, while it is positive when the neighborhood feeds power into the grid.

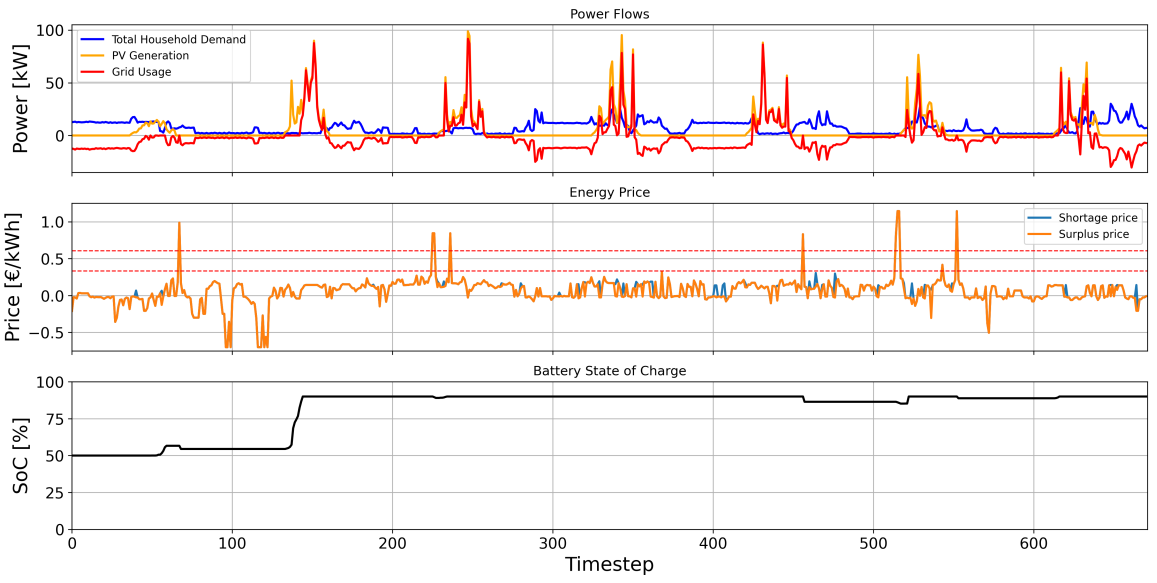


Figure 4.11: A simulation of scenario 2 in which the control thresholds are not optimized (θ_u = €0.607/kWh and θ_l =€0.333/kWh).

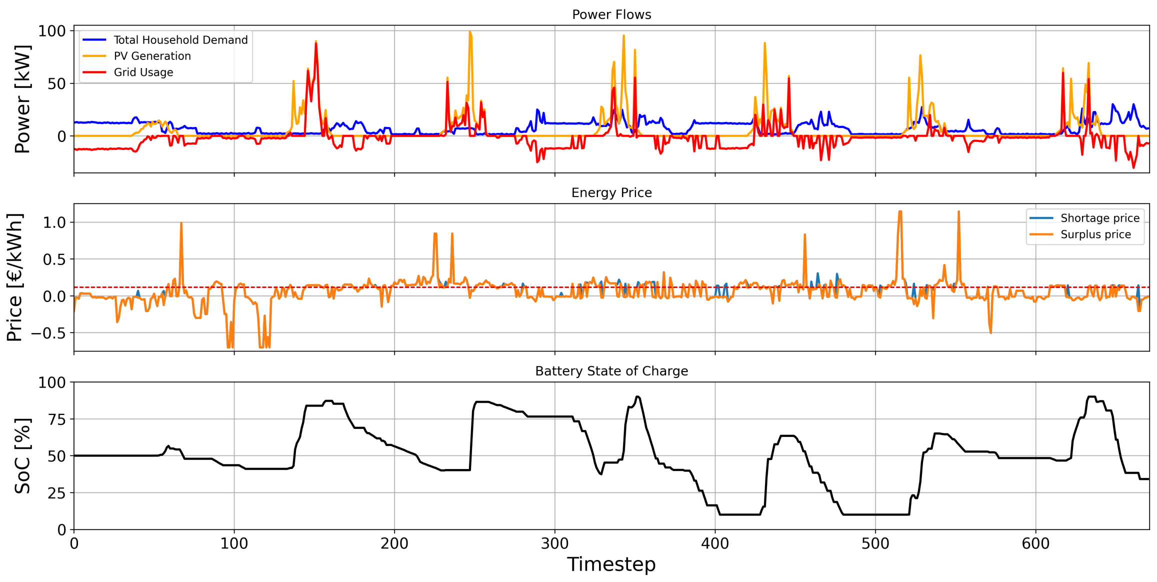


Figure 4.12: A simulation of scenario 2 in which the control thresholds are optimized (θ_u = €0.117/kWh and θ_l =€0.111/kWh).

By comparing the two figures it becomes clear that by optimizing the thresholds, the battery behavior becomes significantly more dynamic. In fact, in the suboptimal case the battery is almost always fully charged, meaning that it is not optimally used to minimize the total energy cost of the neighborhood. This is confirmed by summing the total costs. For the suboptimal case the neighborhood makes a profit of €19.98, while in the optimized case this is €60.51, a factor 3 improvement.

4.3. Electric vehicle (EV) charging station

The last scenario modeled an electric vehicle (EV) charging station equipped with a PV system, a small wind turbine, a grid connection, and five EV charging points. The optimization problem pertaining to the system of this scenario targets to minimize the peak power going through the grid connection, and thus to reduce the strain on the grid connection. The cost found by the algorithm represents the peak grid power and is therefore preferred to be low.

To do this, the charging schedule is determined by searching for the optimal combination of starting time steps for the charging cycle of each EV. The starting time steps are constrained by the arrival and departure times of the EVs and the requirement that each EV must be fully charged before departure.

Each solution to this problem is a combination of five time steps, one for each EV charging cycle. The explored solutions are illustrated using plots of which the x-axis represents the EV index, and y-axis indicates the charging start time step. Starting points belonging to the same solution are connected by lines.

4.3.1. PSO

First the PSO algorithm was applied to this problem. Figure 4.14 illustrates the solutions that were explored by the PSO algorithm. It can be observed that a clear range of feasible solutions was found. In Table 4.9 it can be seen that in the lowest peak power found in all runs was 14.95 kW. The shortest run was 155.8 s. The gaps were relatively large, with the largest being 39.891%.

Table 4.9: Simulation optimization results of all runs of the PSO algorithm applied to scenario 3. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Cost[kW]	Gap[%]
42	9	**155.8**	20.91	39.891
33	11	201.4	18.02	20.544
1	21	357.3	**14.95**	**0**

4.3.2. Parallel L-BFGS-B

For the parallel gradient-based algorithm, the explored solutions are illustrated in Figure 4.15. It can be observed that the solutions show significant variation and are not clustered.

The results of all runs of this algorithm are listed in Table 4.10. The best cost found among all runs is 24.72 kW. The shortest run time was 2789.0 s. The gaps were very large, as the smallest gap was 65.420% and the largest was 97.130%.

Table 4.10: Simulation optimization results of all runs of the parallel L-BFGS-B algorithm applied to scenario 3. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Cost[kW]	Gap[%]
-	8	2991.7	29.46	97.13
42	17	3575.6	**24.72**	**65.42**
33	14	3518.8	29.02	94.15
1	9	**2789.0**	25.81	72.71

4.3.3. GA

The final applied algorithm to scenario 3 was GA. The explored results are illustrated in Figure 4.16. The figure shows a large variance in solutions. The results of the three runs of the GA algorithm are listed in Table 4.11. The best cost found was 14.95 kW. The shortest run time was 118.2 s. The gaps were relatively large and showed a large variance. The largest gap was 51.970%.

Table 4.11: Simulation optimization results of all runs of the GA applied to scenario 3. The shortest run time, the lowest cost, and the smallest gap are highlighted.

ID	Iterations	Run time [s]	Cost[kW]	Gap[%]
42	10	171.1	22.71	51.970
33	10	166.4	**14.95**	**0**
1	7	**118.2**	19.76	32.246

4.3.4. Comparison of the algorithms

In this section, the three algorithm performances on the third scenario of this thesis are compared.

In Figure 4.13 the convergence curves of the three algorithms are illustrated. It can be immediately observed that the L-BFGS-B algorithm did not converge to a similar value as the other algorithms. Additionally, the gradient-based algorithm reached convergence the quickest. In terms of run time, however, the PSO algorithm had the longest run time of the three algorithms, while the GA had the shortest.

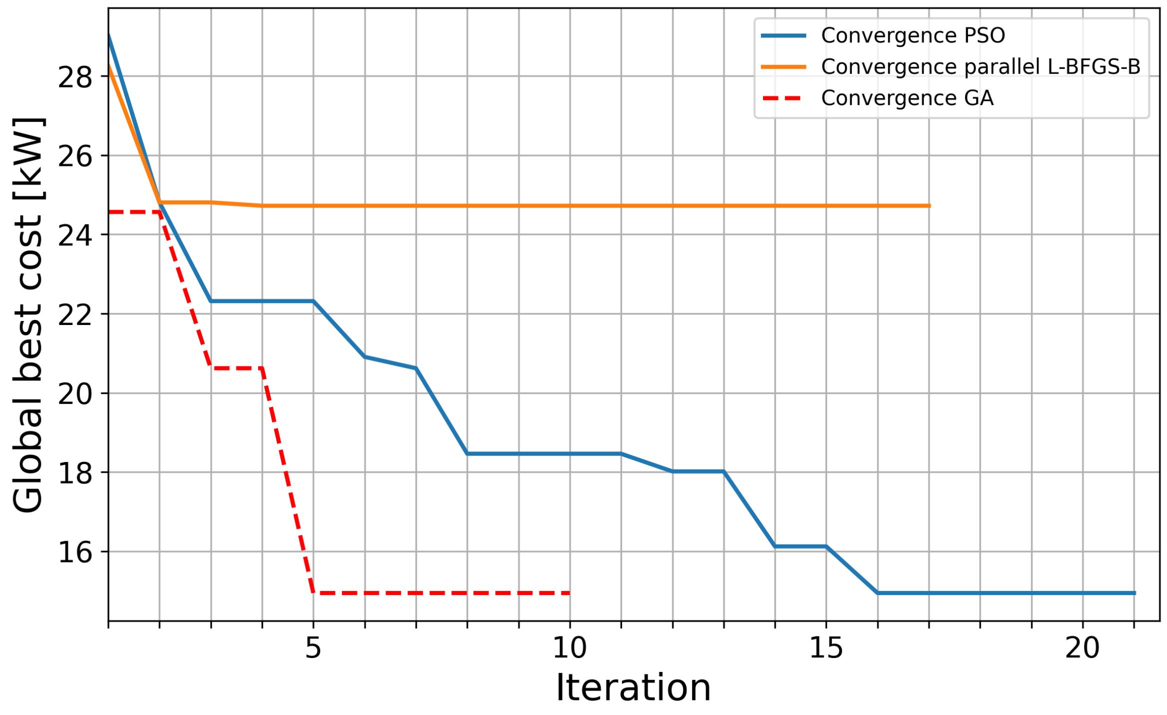


Figure 4.13: Convergence plots of the best runs of the three algorithms applied to scenario 3.

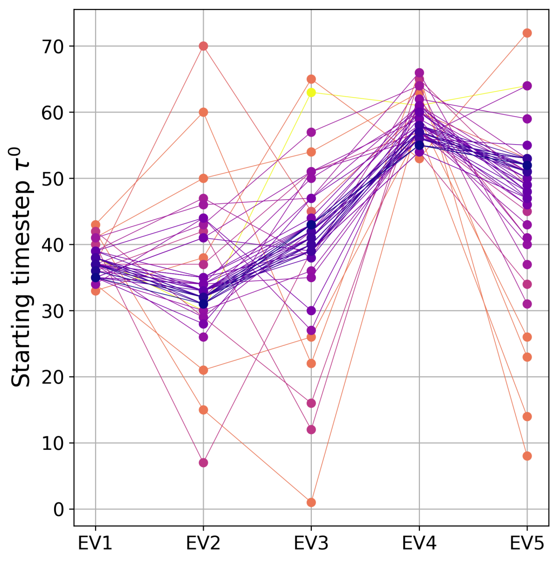


Figure 4.14: Search space explored in the best run of the PSO algorithm to find the charging start time steps for five EVs. The best combination of starting time steps has the lowest cost.

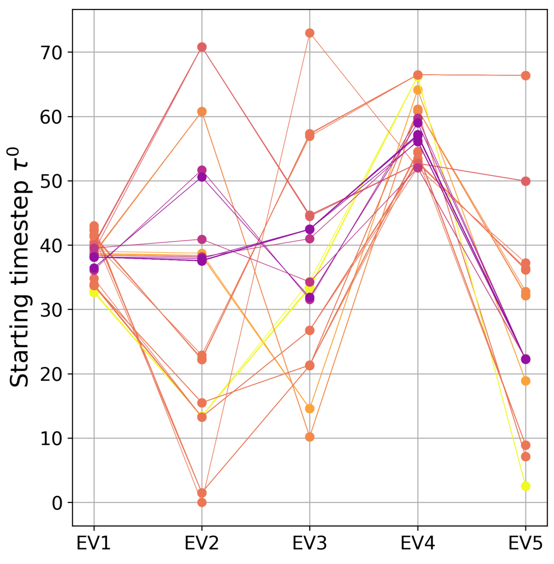


Figure 4.15: Search space explored in the best run of the parallel L-BFGS-B algorithm to find the charging start time steps for five EVs. The best combination of starting time steps has the lowest cost.

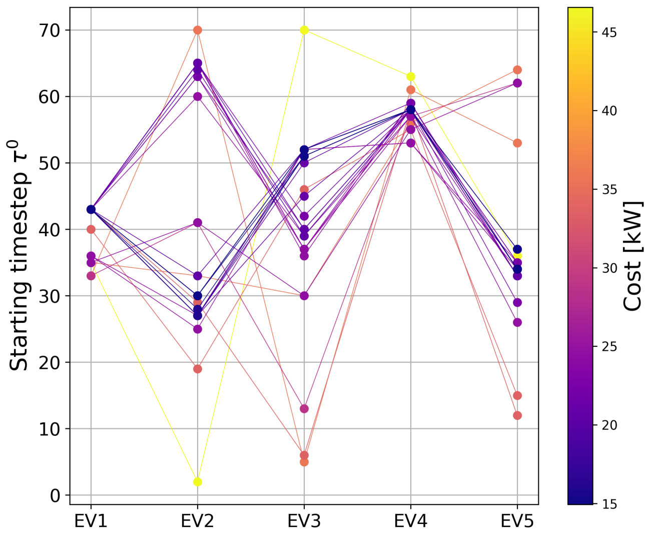


Figure 4.16: Search space explored in the best run of the GA to find the charging start time steps for five EVs. The best combination of starting time steps has the lowest cost.

Table 4.12 lists the characteristics of the three algorithms applied to this problem. Supported by the convergence plot, it can be concluded that the parallel L-BFGS-B did not find a competitive solution. In fact, it has the worst performance on all aspects. Instead, the PSO algorithm achieved the best average cost, and shares the best cost with GA. Additionally, it can be seen that the average gap of PSO and GA were similar, while the average gap of the gradient-based algorithm was significantly higher. The best average time obtained by the GA. From this it is concluded that the PSO algorithm performed best on solution quality, but in terms of speed the GA was better.

Table 4.12: Simulation optimization results of all algorithms applied on scenario 3. The shortest average run time, the best average cost, the absolute best cost, and the smallest average gap are highlighted.

Algorithm	Avg. iteration	Avg. time [s]	Avg.cost [kW]	Best cost [kW]	Avg. gap [%]
PSO	13.67	238.2	**17.96**	**14.95**	**20.145**
L-BFGS-B	13	3362.1	27.73	24.72	85.565
GA	9	**151.88**	19.14	**14.95**	28.072

4.3.5. Optimization result

To show the result of the optimization performed on the third scenario, two schedules are compared. The first schedule is a suboptimal schedule, while the second schedule is obtained by the best run of the PSO algorithm. The comparison can be observed in Figure 4.17. The first plot in this figure illustrates the RES generation and the grid usage for both the suboptimal and the optimized schedule. It is important to note that for a negative grid usage power is extracted from the grid, while for a positive grid usage, power is fed into the grid. The second and third plot represent a suboptimal and optimized charging schedule, respectively. Each EV charging cycle is represented by a distinct color.

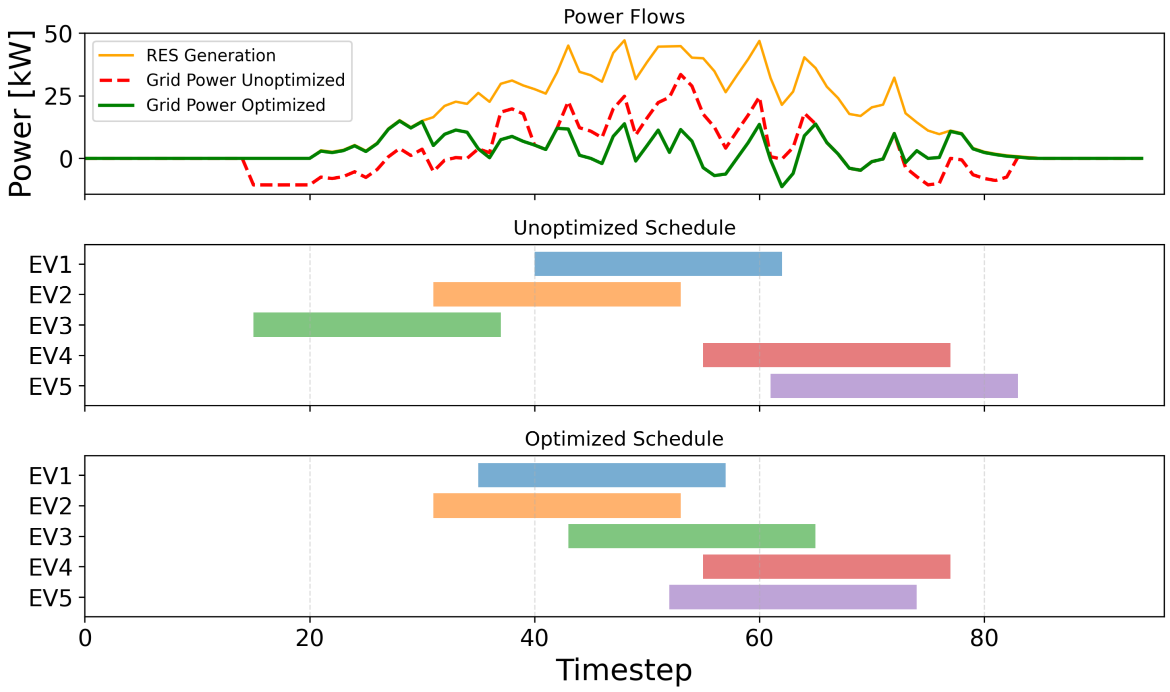


Figure 4.17: A comparison between a random suboptimal EV charging schedule and the schedule found by the best run of PSO.

In this figure, it can be observed that the magnitude of the largest grid peak is lower in the optimized schedule compared to the peak in the suboptimal schedule. In fact, in the suboptimal schedule the largest peak is 14.9 kW, while in the optimized schedule this is 33.5 kW. Furthermore, it can be observed that in the optimized schedule, there is a larger overlap between EV charging cycles, and that the cycles are more centered around the middle of the day.

4.3.6. Parallelization result

The EV charging station scenario was also optimized with sequential versions of the population-based algorithms. To ensure a fair comparison, both PSO and GA were initialized with the same initial solutions as used in their fastest parallel runs (run 42 for PSO and run 1 for GA). As a consequence, the same number of generations were needed and the same solution was found.

Table 4.13 shows the run time for the sequential and parallel implementations of PSO and GA. It can be observed that the parallel implementation of PSO is 4.7 times faster than its sequential version. For GA the parallel implementation is 4.8 faster than its sequential counterpart.

Table 4.13: Parallel and sequential run times for PSO and GA.

Algorithm	Sequential run time [s]	Parallel run time [s]
PSO	735.08	**155.81**
GA	571.96	**118.18**

5 Discussion

In this chapter of the thesis, the results presented in Chapter 4 are discussed and analyzed. First, the research questions posed in the introduction chapter of this thesis (Chapter 1) are answered. After this, additional remarks are presented.

5.1. Research question

In this section, the thesis is discussed from the point of view of the research questions proposed in the introduction. First the sub-questions related to simulation optimization of energy systems are answered, followed by the main research question on this topic.

1.1 *What are the main technical prerequisites and challenges for combining optimization algorithms with simulation models?*

The use of simulation models for the optimization of energy systems involves several key challenges and considerations that must be addressed to ensure effective and reliable outcomes. As pointed out in Section 3.2.1, one of the primary challenges in the implementation of simulation-based optimization is the computational time required to run detailed simulations. Since each algorithm iteration requires at least one simulation, it is important to use an algorithm that finds an optimum within a limited number of iterations. To further minimize run times in existing algorithms, parallel computing was implemented as described in Section 3.2.5, Section 3.2.6, and Section 3.2.7. The potential of this method became evident when the theoretical simulation time equations of the sequential methods were compared to their corresponding parallelized versions. The parallel execution of independent iterations resulted in a significant reduction of total run time. In fact, as Section 4.3.6 pointed out, the parallel versions of PSO and GA were almost 5 times faster than their respective original versions (4.7 and 4.8 times respectively).

Nevertheless, the results obtained in the previous chapter confirmed that the optimization of a system by using simulation optimization can be time costly. In fact, the shortest run time for the single-variable continuous problem of scenario 1, was 282.9 s. For the two-variable continuous problem of scenario 2, which aimed to optimize the operation of a system, the minimal run time was 615.7 s. For the last problem, which was dependent on five discrete decision variables related to the planning of the system, at least 118.2 s were needed to find a solution.

Although in general the total run time is proportional to the number of iterations, it can be observed that this is not always the case. For example, when the PSO algorithm was applied to scenario 1 in Section 4.1.1, the number of generations was equal in each run, while the simulation times slightly varied. This can be caused by the fact that each run started with different starting solutions, resulting in different particle trajectories before reaching an optimum. Furthermore, external conditions, such as unrelated CPU usage during the runs, have an influence as well.

Aside from the aspects of time and number of iterations, there is a challenge regarding the objective and search space for simulation optimization problems. As was explained in Section 3.2.1, the problems that are solved with simulation optimization are black-box optimization problems. Because of this, there is an uncertainty about the characteristics of the search space. Due to this, it can be difficult to select the most effective algorithm, especially for complex problems. To overcome this challenge, it is useful to have a generalized approach, and to classify the problem at hand as much as possible. For the EV charging station problem, for example, it was known beforehand that the search space was discrete, as the solution would entail a set of discrete time steps. An algorithm that is known to be ineffective on this type of search space can therefore already be excluded.

To summarize, the main challenges that arise when combining optimization algorithms with simulation models are the following:

- Large computation times due to the need for a simulation for each iteration;
- Uncertainty in the search space and objective of the problem because of simulation dependability (black-box problem).

The following prerequisites are necessary for simulation optimization:

- The number of iterations, or rather the number of subsequent simulations, must be kept at a minimum. An effective way of doing this is parallelization of independent iterations;
- Classifying the problem as much as possible beforehand and adapting the approach accordingly.

1.2 *Which optimization algorithms are applicable to simulation optimization in energy system modeling, and how do they differ from one another?*

By investigating three different optimization problems in this thesis, it was observed that there is a variety of possible problem types in this domain. The first problem was based on the design of the system, the second problem aimed to optimize the operation of a system, and the last problem addressed the optimization of the planning of a system. Furthermore, the third scenario involved a discrete problem, while the others were continuous. Also, the number of variables differed among the three problems. Considering this diversity, it is important to analyze how the different algorithms performed on each problem type.

The three algorithms evaluated in this thesis use different approaches to find optima. PSO and GA use evolutionary exploration methods, while L-BFGS-B is gradient-based. PSO employs a population of particles that change their direction and speed depending on personal and group performance. Because the particles retain their own identity across generations, they remain the same individual entities. In GA, the exploration is based on genetic evolution, and by using recombination and mutation, each generation consists of new entities. Both exploration methods make it possible to escape local optima and increase the likelihood of finding the global optimum. L-BFGS-B, on the other hand, makes use of gradient properties by using the finite difference method. The implemented algorithm uses instances of local optimization by starting from different initial solutions. When an individual instance finds an optimum, the instance stops its exploration. The quality of the initial solutions strongly influences whether the global optimum is found in the gradient-based algorithm. This is confirmed by the results obtained in Section 4.2.2, where four runs of the algorithm with different starting points were executed. Among the different runs there is a large variance in the cost. In this scenario the calculated gaps were 9.129%, 66.985%, 29.356% and 59.236%, indicating a large variance among runs with different starting points. This variance was less noticeable in the other algorithms.

To compare the convergence properties of each applied algorithm, Figure 4.1, Figure 4.7, and Figure 4.13 can be used. In the first two figures, PSO and GA started to converge after a similar number of generations, namely after around 5 and 15 generations, respectively. The average run times from Table 4.4, 4.8, confirm that the speed of the two algorithms are similar in these scenarios, but PSO is slightly quicker. In the third scenario, GA had a shorter run time compared to PSO, namely 151.9 s versus 238.2 s. From this data it is inferred that, in terms of time efficiency, PSO performs slightly better than GA in the continuous problems, while GA is significantly faster in the discrete problem. L-BFGS-B was consistently slower in all cases. In fact, compared to the two population-based algorithms, L-BFGS-B was at least a factor 2 slower.

By observing the values from Table 4.4, it can be seen that PSO and GA had very similar results in terms of average cost (both 420.36 kg), best cost (419.91 kg and 419.92 kg), and average gap (both 0.107%. The difference is that, on average, PSO required less run time. L-BFGS-B performed significantly worse on all fronts, although it found a similar best cost and had a small average gap. In the second scenario (Table 4.8), the same pattern was observed. The PSO algorithm and the GA had similar performance in terms of best cost and average cost, although the PSO performed slightly better. PSO had an average gap of 0.363%, while GA had an average gap of 0.598%. Again, the parallelized L-BFGS-B algorithm performed worse. In fact, it was not able to find an optimum that was competitive with the optima found by the other algorithms. For the third scenario (Table 4.12), both PSO and GA found the same best cost (14.95 kW). However, their average cost varied slightly, which was also reflected in the average gap. PSO had an average gap of 20.145% and GA had an average gap of 28.072%. On the other hand, as mentioned earlier, GA required less time. The parallel L-BFGS-B had the worst cost performance. In fact, it did not find a competitive solution. This was as expected, because it was known beforehand that it is not a suitable algorithm for discrete problems.

Between the explored algorithms in this thesis, the PSO algorithm resulted in the best average cost and the best overall cost for all problems. Therefore, the PSO algorithm is the preferred algorithm for simulation optimization. GA was comparable on this aspect but yielded slightly worse cost. However, on the discrete problem, GA performed better time-wise while obtaining an equally good solution. If this time characteristic is prioritized, GA can be considered the best algorithm for discrete optimization. Furthermore, it is proved that the parallelized L-BFGS-B algorithm works for simple optimization problems such as the one presented in scenario 1, but that it is not competitive compared to the other two algorithms on any aspect.

1.3 *How generalizable is a simulation optimization framework to various energy system problem domains?*

Based on the experiments performed with the selected algorithms it can be concluded that the Particle Swarm Optimization (PSO) algorithm generally performed better than the other algorithms on the presented set of problems. Therefore, PSO is the preferred algorithm for a generalized simulation optimization framework. A potential distinction can be made for discrete problems, as the time performance of GA on the discrete EV charging station problem was better than that of PSO. However, to confirm this distinction, PSO and GA should be applied to additional discrete problems. Examples of these are: other scheduling problems, such as load shedding or unit commitment decisions, and problems regarding component selection from a set of predefined components.

Since no correlation was found between the algorithm performances and the energy carrier types treated in the scenarios, simulation optimization is confirmed to be generalizable for different energy carriers. The framework has been evaluated on the distinct hydrogen and electricity domains, but is expected to perform equally for energy domains such as natural gas or heating systems. It must be noted that the experiments performed in this thesis did not cover multi-carrier systems such as a combination of hydrogen and electricity. However, it is expected that the optimization framework works on these types of systems as well.

A distinction was made between design, operational, and planning problems in energy systems. However, no difference in performance was found in the conducted experiments that can be correlated to this distinction. To draw a conclusion regarding this distinction, more problems must be evaluated. However, it is proven that the optimization framework is generalizable on the problems presented in this thesis.

To conclude, a simulation optimization framework is well generalizable among the different domains treated in this thesis. The best algorithm in terms of generalization was found to be PSO.

1.4 *How can multi-carrier energy systems be effectively modeled to enable meaningful insights?*

In this thesis two energy carriers were presented: electricity and hydrogen. As The Illuminator lacked full capability of simulating hydrogen systems, several additional component models were developed. This was done by using a top-down approach and by considering each component as a black box. The newly created models can be subdivided into the five categories presented in Section 2.2: production, conversion, storage, transport, and utilization. From this, the analogy between electrical and hydrogen component types became apparent. With the use of an appropriate abstraction level, the behavior of real-life components was approximated. The results of some of these models are observable in Section 4.1, which shows how the models are working together as a system that is optimizable by the created framework.

The simulation results presented in this thesis, together with the description of the created models, therefore show how energy systems with varying carriers can be effectively modeled. Especially in combination with the newly implemented optimization framework, meaningful insights could be generated regarding designing, operating, and planning multi-carrier energy systems. Although this thesis did not actively present multi-carrier energy systems, the results can be extended to scenarios in which models pertaining to the electrical and hydrogen domains can be combined. This functionality is readily available, since all models of The Illuminator, including the newly created hydrogen models, are created at the same level of abstraction. Furthermore, all inputs and outputs use the same units. Therefore, no difference in performance is expected when models belonging to different carriers are directly connected to each other. Examples of systems that can currently be simulated and optimized are hydrogen production facilities that use electrical systems to produce hydrogen. Another example is the use of hydrogen as energy storage in energy hubs. A combination of the scenarios presented in this thesis can be a reasonable starting point. The exploration of these types of systems can provide valuable insights into multi-carrier energy systems, contributing to the energy transition.

1. ***How can simulation optimization support the development of improved configurations of renewable energy systems?***

As energy system complexity increases due to an increasing share of renewable energy sources, the design and operation of energy systems becomes more challenging. While the use of simulation facilitates the design, operation, and planning of energy systems, it does not inherently aid the search for optimality in terms of efficiency, cost, and reliability. By combining simulation with optimization techniques, it becomes possible to find optimal simulation parameters, and with this support the development of energy systems.

Three scenarios, each with a different objective, were modeled and optimized in this thesis. The first scenario involved the optimization of the size of the buffer, while ensuring system efficiency and cost minimization. The second scenario involved optimization of the operational strategy, and the final scenario involved planning optimization. The diversity of the scenarios and the success of the algorithms proved that simulation optimization can be applied to a variety of energy system problem types.

The ability to manage continuous and discrete variables confirms the potential of simulation optimization. The best performing algorithms, PSO and GA, showed reliable performance in system optimization with complex search spaces, a common characteristic of systems that consist of many interdependent components.

A notable challenge in simulation optimization is computation time. The implementation of parallelization in this thesis helped to minimize this time, creating prospects for the application of simulation optimization on energy systems.

5.2. Further remarks

As mentioned in Section 3.3, the evaluation scenarios of this thesis were fictional scenarios that represent potential scenarios as accurately as possible. However, there are certain improvements that can be made to further increase the realism of the scenarios. For example, in the first scenario, specifically in Figure

4.5 and 4.6, it can be predicted that if the simulation time frame had continued with the same production and demand pattern, the buffer would have been too small to prevent hydrogen flaring. The reason for this is that in this scenario the total production was significantly higher than the total demand. Additionally, the system had a low level of flexibility in terms of demand and production control. In the second scenario, the assumption was made that all neighbors connected to the energy hub shared the interest of maximizing the total profit. To make the scenario more accurate, intra-hub trading could be implemented, such that the individual profits could be optimized. Moreover, to increase the profit, trading with the energy stored in the battery could also be implemented. In the third scenario, the assumption was made that an EV can only charge in one uniform charging stint. In real life however, this is not an obligation. Furthermore, for this scenario it must be noted that the starting timesteps of EV2, EV3, and EV5 are interchangeable. The reason for this is that these vehicles are present at the charging station during the same period of time. This does not become clear in the figures 4.14, 4.15, and 4.16, because of the search strategy of the algorithms. In fact, the algorithms do not explore all possible solutions but rather persuade an optimal solution. As a consequence, there is no guarantee that the optima found in this thesis are the true optimal solutions.

6 Conclusion

6.1. Conclusion

This thesis investigated the potential of simulation optimization to support the design, operation, and planning of energy systems. To execute this investigation, The Illuminator simulation software was used. To extend its capabilities, new (hydrogen) models were developed to be able to explore different energy carrier domains. Additionally, an optimization framework was integrated with The Illuminator, which enabled optimization by using simulation performed by this software. To evaluate the technique, three distinct scenarios were created.

Three optimization algorithms were implemented: Particle Swarm Optimization (PSO), a gradient-based algorithm (L-BFGS-B), and a Genetic Algorithm (GA). As these algorithms were not directly compatible with the simulation software, a framework was built to feed algorithm-generated decision variables into the simulation, and to evaluate the output of a simulation. It was found that both the simulation time and the number of simulations played a significant role in the total run time of simulation optimization. This finding is consistent with literature. Parallelization was applied to the algorithms in an attempt to reduce run times. For PSO and GA this allowed solutions within the same generation to be evaluated simultaneously. Indeed, for these two algorithms, the total runtime was reduced by almost a factor 5. As L-BFGS-B was found to require multiple runs at various starting points to find a global optimum, parallelization was used to execute these runs simultaneously to reduce the total run time as well.

The three evaluation scenarios were diverse and covered various domains in different categories. The first scenario involved a hydrogen production facility that required the optimization of the buffer size. The scenario covered a hydrogen-based system design problem with a continuous variable. The second scenario involved a residential energy hub. Its purpose was to optimize two operational parameters that determined the use of the electricity grid and its financial impact. Hence, the optimization problem covered an electricity-based system operation problem with continuous variables. The last scenario involved a charging station for electric vehicles (EV) that was powered by RES and a grid connection. For this scenario, the goal was to optimize the charging schedule (measured in discrete time steps) of the EVs to minimize the peak power from and towards the grid. The problem therefore covered an electricity-based system planning problem with discrete variables.

PSO was found to be the overall best performing algorithm, performing well across all domains, i.e., continuous, discrete, hydrogen, electricity, and all system problems, i.e., system design, operation, and planning. Only for the third scenario GA outperformed PSO in runtime. The L-BFGS-B performed worst in all scenarios. Although it found a viable solution in the first scenario, it was not able to find a competitive solution for the other two scenarios.

From these findings, it was concluded that PSO is the most suitable algorithm for the optimization problems presented in this thesis. Moreover, the results show that simulation optimization is a valuable technique that can support and improve the development of renewable energy systems, despite its computational demands. Future work should focus on the experimentation of the developed framework on real-life systems, validation, and further computation speed improvements.

6.2. Recommendations

Based on the results of the experiments conducted and the simulation optimization methods applied, a number of recommendations for future research is proposed.

To assess the potential of simulation-based algorithms and the performance of GA and PSO, it is advisable to experiment with scenarios that cover a larger time frame or to make a compilation of multiple smaller time frames. For example, the energy hub scenario could be simulated for a single day or week and use these results to initiate a consecutive simulation. By implementing this chain of simulation, the price thresholds can become more dynamic and may be used to control the hub for a longer period of time.

Another recommendation for future research is the validation of the developed framework by using real-life systems and experimental setups for which the optimal values are known. This way, the true accuracy of the optimization framework can be found, as in this thesis the performance was measured by comparing each result to the best-found results, making it a relative metric.

Additionally, it can be valuable to experiment with various algorithm parameters. Using the optimal parameter for a specific problem may result in higher accuracies and perhaps shorter computation times. These optimal optimization parameters can for example be found by performing a high number of runs with different parameters, and by investigating their effect.

From the results of the applied algorithms it was concluded that PSO is the most suitable algorithm to implement for a general simulation optimization framework. GA showed good speed performance on a discrete problem. For future research, the capability of this algorithm on more complex discrete problems must be further investigated. Suggestions for scenarios are: other system planning algorithms or problems in which the optimal component combination is required. An example of the latter category is a problem in which a combination of components must be made from a set of existing components with slightly varying characteristics.

Finally, an important research topic is the further improvement of run time. An example of an improvement is the memory handling of simulation files. Currently, each simulation requires the creation of a separate configuration file and a separate output file. Ideally, this is not needed and is replaced by overwriting operations. Additionally, it can be interesting to investigate the influence of hardware specifications of the simulation system. An example is the use of more computation cores or even the use of multiple processors contemporaneously. Effectively this could lead to an increase of the population size for PSO and GA, which in its turn could reduce the number of generations needed to find a viable solution.

References

[1] Energy Agency International (IEA), “World Energy Outlook 2024,” p. 103, Oct. 2024. [Online]. Available: https://www.iea.org/reports/world-energy-outlook-2024, (accessed on May 16, 2025).

[2] D. Maradin, L. Cerović, and T. Mjeda, “Economic Effects of Renewable Energy Technologies,” *Naše gospodarstvo/Our economy*, vol. 63, no. 2, pp. 49–59, 2017. doi: 10.1515/ngoe-2017-0012.

[3] S. P. Filippov and A. B. Yaroslavtsev, “Hydrogen energy: development prospects and materials,” *Russian Chemical Reviews*, vol. 90, no. 6, pp. 627–643, Jun. 2021, issn: 0036-021X. doi: 10.1070/RCR5014.

[4] M. A. Rosen and S. Koohi-Fayegh, “The prospects for hydrogen as an energy carrier: an overview of hydrogen energy and hydrogen energy systems,” *Energy, Ecology and Environment*, vol. 1, no. 1, pp. 10–29, Feb. 2016, issn: 23638338. doi: 10.1007/s40974-016-0005-z.

[5] A. Fu, R. Saini, R. Koornneef, A. van der Meer, P. Palensky, and M. Cvetkovic, *The Illuminator*. [Online]. Available: https://github.com/Illuminator-team/Illuminator, (accessed on Mar. 25, 2025).

[6] A. Fu, R. Saini, R. Koornneef, A. van der Meer, P. Palensky, and M. Cvetković, “The illuminator: An open source energy system integration development kit,” *2023 IEEE Belgrade PowerTech*, pp. 1–5, 2023. doi: 10.1109/PowerTech55446.2023.10202816.

[7] L. Hrčka, P. Važan, and Z. Šutová, “Basic Overview of Simulation Optimization,” *Research Papers Faculty of Materials Science and Technology Slovak University of Technology*, vol. 22, no. 341, pp. 11–16, Dec. 2014. doi: 10.2478/RPUT-2014-0001.

[8] Meteonorm, *Climate*. [Online]. Available: https://meteonorm.com/climate/, (accessed on Jul. 15, 2025).

[9] H. Jongen, K. Meer, and E. Triesch, *Optimization Theory*. New York: Springer, 2004. doi: 10.1007/b130886.

[10] N. Guide, *Optimization problem types*. [Online]. Available: https://neos-guide.org/guide/types/, (accessed on Mar. 27, 2025).

[11] A. Roberts Wayne and D.E. Varberg, *Convex Functions*. New York: Academic Press Inc, 1974, vol. 57.

[12] K. H. Rahi, H. K. Singh, and T. Ray, “Partial Evaluation Strategies for Expensive Evolutionary Constrained Optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 6, pp. 1103–1117, Dec. 2021, issn: 19410026. doi: 10.1109/TEVC.2021.3078486.

[13] K. Terayama, M. Sumita, R. Tamura, and K. Tsuda, “Black-Box Optimization for Automated Discovery,” *Accounts of Chemical Research*, vol. 54, no. 6, pp. 1334–1346, Mar. 2021, issn: 15204898. doi: 10.1021/acs.accounts.0c00713.

[14] I. Bajaj, A. Arora, and M. M. Hasan, “Black-Box Optimization: Methods and Applications,” *Springer Optimization and Its Applications*, vol. 170, pp. 35–65, Jan. 2021, issn: 19316836. doi: 10.1007/978-3-030-66515-9.

[15] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. United States: Dover Publications, Jan. 1998.

[16] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization,” *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, Sep. 2003, issn: 03600300. doi: 10.1145/937503.937505.

[17] R. Olaechea, D. Rayside, J. Guo, and K. Czarnecki, “Comparison of exact and approximate multi-objective optimization for software product lines,” *ACM International Conference Proceeding Series*, vol. 1, pp. 92–101, Sep. 2014. doi: 10.1145/2648511.2648521.

[18] Y. Wang, “Review on greedy algorithm,” *Theoretical and Natural Science*, vol. 14, no. 1, pp. 233–239, Nov. 2023, issn: 2753-8826. doi: 10.54254/2753-8818/14/20241041.

[19] S. Desale, A. Rasool, S. Andhale, and P. Rane, “Heuristic and Meta-Heuristic Algorithms and Their Relevance to the Real World: A Survey,” *International Journal of Computer Engineering in Research Trends*, vol. 2, pp. 296–304, 2015, issn: 2349-7084.

[20] V. Shoarinezhad, S. Wieprecht, and S. Haun, “Comparison of Local and Global Optimization Methods for Calibration of a 3D Morphodynamic Model of a Curved Channel,” *Water 2020, Vol. 12, Page 1333*, vol. 12, no. 5, p. 1333, May 2020, issn: 2073-4441. doi: 10.3390/W12051333.

[21] B. Hartke, “Global optimization,” *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 1, no. 6, pp. 879–887, Nov. 2011, issn: 1759-0884. doi: 10.1002/WCMS.70.

[22] M. Cavazzuti, “Deterministic Optimization,” *Optimization Methods*, pp. 77–102, Sep. 2013. doi: https://doi.org/10.1007/978-3-642-31187-1.

[23] J. Haddock and J. Mittenthal, “Simulation optimization using simulated annealing,” *Computers & Industrial Engineering*, vol. 22, no. 4, pp. 387–395, Oct. 1992, issn: 0360-8352. doi: 10.1016/0360-8352(92)90014-B.

[24] M. H. Alrefaei and A. H. Diabat, “A simulated annealing technique for multi-objective simulation optimization,” *Applied Mathematics and Computation*, vol. 215, no. 8, pp. 3029–3035, Dec. 2009, issn: 0096-3003. doi: 10.1016/J.AMC.2009.09.051.

[25] K. Amine, “Multiobjective Simulated Annealing: Principles and Algorithm Variants,” *Advances in Operations Research*, vol. 2019, no. 1, p. 8 134 674, Jan. 2019, issn: 1687-9155. doi: 10.1155/2019/8134674.

[26] J. A. Joines, K. A. Thoney, R. E. King, and M. G. Kay, “Supply chain multi-objective simulation optimization,” *4th International Industrial Simulation Conference 2006, ISC 2006*, pp. 377–383, 2006. doi: 10.1109/WSC.2002.1166395.

[27] B. Dengiz and C. Alabas, “Simulation optimization using tabu search,” *Winter Simulation Conference Proceedings*, vol. 1, pp. 805–810, 2000, issn: 02750708. doi: 10.1109/WSC.2000.899877.

[28] A. Sedki and D. Ouazar, “Swarm intelligence for groundwater management optimization,” *Journal of Hydroinformatics*, vol. 13, no. 3, pp. 520–532, Jul. 2011, issn: 1464-7141. doi: 10.2166/HYDRO.2010.163.

[29] H. Abbasi, A. Afshar, and M. R. Jalali, “Ant-colony-based simulation–optimization modeling for the design of a forced water pipeline system considering the effects of dynamic pressures,” *Journal of Hydroinformatics*, vol. 12, no. 2, pp. 212–224, Mar. 2010, issn: 1464-7141. doi: 10.2166/HYDRO.2009.147.

[30] R. F. Adebiyi, K. A. Abubilal, M. B. Mu’azu, and B. H. Adebiyi, “Intelligent Systems and Applications,” *Intelligent Systems and Applications*, vol. 8, pp. 68–74, 2018. doi: 10.5815/ijisa.2018.08.06.

[31] European Forum for Reciprocating Compressors, *Tutorial: Introduction to Thermodynamics.* [Online]. Available: https://www.recip.org/tutorial-introduction-to-thermodynamics/, (accessed on Apr. 16, 2025).

[32] A. M. Elberry, J. Thakur, A. Santasalo-Aarnio, and M. Larmi, “Large-scale compressed hydrogen storage as part of renewable electricity storage systems,” *International Journal of Hydrogen Energy*, vol. 46, no. 29, pp. 15 671–15 690, Apr. 2021, issn: 0360-3199. doi: 10.1016/J.IJHYDENE.2021.02.080.

[33] A. Raj, I. A. Larsson, A. L. Ljung, *et al.*, “Evaluating hydrogen gas transport in pipelines: Current state of numerical and experimental methodologies,” *International Journal of Hydrogen Energy*, vol. 67, pp. 136–149, May 2024, issn: 0360-3199. doi: 10.1016/J.IJHYDENE.2024.04.140.

[34] G. Papaevangelou, C. Evangelides, and C. Tzimopoulos, “A new explicit relation for the friction coefficient in the Darcy-Weisbach equation,” *Proceedings of the Tenth Conference on Protection and Restoration of the Environment: PRE10*, Jul. 2010. [Online]. Available: https://tinyurl.com/yr63kz87.

[35] H. Kanesugi, K. Ohyama, H. Fujiwara, and S. Nishimura, "High-pressure hydrogen permeability model for crystalline polymers," *International Journal of Hydrogen Energy*, vol. 48, no. 2, pp. 723–739, Jan. 2023, issn: 0360-3199. doi: 10.1016/J.IJHYDENE.2022.09.205.

[36] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury, "Simulation optimization: a review of algorithms and applications," *4OR*, vol. 12, no. 4, pp. 301–333, Nov. 2014, issn: 16142411. doi: 10.1007/s10288-014-0275-2.

[37] K. Chandrasekar and N. V. Ramana, "Performance Comparison of GA, DE, PSO and SA Approaches in Enhancement of Total Transfer Capability using FACTS Devices," *Journal of Electrical Engineering & Technology*, vol. 7, no. 4, pp. 493–500, 2012. doi: 10.5370/JEET.2012.7.4.493.

[38] S. Alam, X. Zhao, I. K. Niazi, M. S. Ayub, and M. A. Khan, "A comparative analysis of global optimization algorithms for surface electromyographic signal onset detection," *Decision Analytics Journal*, vol. 8, p. 100 294, Sep. 2023, issn: 2772-6622. doi: 10.1016/J.DAJOUR.2023.100294.

[39] H. Wang and J. Gong, "A comparative study of GA, PSO and SCE algorithms for estimating kinetics of biomass pyrolysis," *Emergency Management Science and Technology*, vol. 2023, p. 9, doi: 10.48130/EMST-2023-0009.

[40] M. Wetter and J. Wright, "A comparison of deterministic and probabilistic optimization algorithms for nonsmooth simulation-based optimization," *Building and Environment*, vol. 39, no. 8, pp. 989–999, Aug. 2004, issn: 0360-1323. doi: 10.1016/J.BUILDENV.2004.01.022.

[41] Vahid Beiranvand, Warren Hare, and Yves Lucet, "Best practices for comparing optimization algorithms," *Optimization and Engineering*, vol. 18, no. 4, pp. 815–848, Dec. 2017, issn: 15732924. doi: 10.1007/s11081-017-9366-1.

[42] J. Blanc and k. Deb, "pymoo: Multi-Objective Optimization in Python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020. doi: 10.1109/ACCESS.2020.2990567.

[43] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Mar. 2020, issn: 15487105. doi: 10.1038/S41592-019-0686-2.

[44] G. Hebrail and A. Berard, *Individual Household Electric Power Consumption*, 2006. doi: 10.24432/C58K54.

[45] TenneT, *Settlement Prices*, 2023. [Online]. Available: https://www.tennet.eu/nl-en/node/3479, (accessed on Jun. 20, 2025).

[46] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

[47] L. M. Valle-Falcones, C. Grima-Olmedo, L. F. Mazadiego-Martínez, A. Hurtado-Bezos, S. Eguilior-Díaz, and R. Rodríguez-Pons, "Green Hydrogen Storage in an Underground Cavern: A Case Study in Salt Diapir of Spain," *Applied Sciences 2022, Vol. 12, Page 6081*, vol. 12, no. 12, p. 6081, Jun. 2022, issn: 2076-3417. doi: 10.3390/APP12126081.

A Detailed Figures

A.1. Background Chapter

Table A.1 is a more detailed version of Table 2.2 from Section 2.3.2. In this table the types each algorithm belongs to is listed. Additionally, the table provides a short summary of the operation of each algorithm.

Table A.1: Optimization algorithms used in simulation optimization literature (detailed).

Algorithm	Type	Algorithm summary	References
Simulated annealing	• Single objective; • meta-heuristic; • stochastic; • single-solution based • continuous; • global.	Begin the iteration with an initial solution and temperature T_0. Iteratively: generate neighbor solution, evaluate the objective function, in case of a positive change accept solution, in case of negative change accept with a probability $P(T, f(x))$, reduce temperature. Repeat until $T < T_f$.	[23][1]
	• Multi-objective; • meta-heuristic; • stochastic; • single-solution based • continuous; • global.	Similar to single-objective simulated annealing. However now, non-dominating solutions found in the exploration of the domain are stored in a memory. With the correct acceptance probability, the problem converges to a set of Pareto solutions.	[25], [24][1]

Genetic algorithm	• Single-objective; • meta-heuristic; • stochastic; • population based; • continuous; • global.	Begin the iteration by initializing a randomly generated population consisting of N individuals and evaluating the objective function for each of these individuals. The best individuals are recombined based on their performance, generating a new population. The individuals of the new population are evaluated and iteration is continued until the maximum number of iterations is reached [26].	
	• Multi-objective; • meta-heuristic; • stochastic; • population based; • continuous; • global. .		[26][1]
Tabu search	• Single-objective; • meta-heuristic; • deterministic; • single-solution based; • discrete; • global search. • single-solution based.	A random initial solution is picked and selected as current best solution. Iteratively the following two steps are executed until the maximum number of iterations occur: 1) Neighboring parameter sets are generated and evaluated. 2) The best newly occurring is stored in the Tabu list. When the list is full, the oldest set is eliminated. When the iterations are complete, the best solution in the Tabu list is set to be the best solution.	[27][1]

Particle swarm	• Single-objective; • meta-heuristic; • stochastic; • population based; • continuous; • global. • population based.	Initial positions of a number of particles are generated randomly in the search space. After this, they are individually evaluated in the objective function. The results are set as the current best solution for each particle (personal best). Iteratively the particles values are updated based on their previous value and the particle velocity. The particle velocity can be defined as the magnitude and direction of the particle's movement in search space. The velocity is dependent on a reduced inertia weight (controlling the influence of the previous velocity), the personal best solution of the particle, and the global best solution, which is the best solution found by any particle in the swarm. The process is terminated when the particles are converged to a similar solution or the maximum number of iterations is reached.	[28][1] [46]
Ant colony	• Single-objective; • meta-heuristic; • stochastic; • population based; • discrete; • global.	Initially a set of ants(agents) travel through search space at random. After each step, each agent selects its new path based on the pheromone level of paths, which is dependent on the quality of the solutions found by other agents. The pheromones are 'evaporated' by time, in other words their strength is decayed, to promote diversity and prevent premature convergence. In the meantime the global best solution is tracked and the process is repeated until the termination conditions are met.	[29][1]

Artificial bee colony	• Single-objective; • meta-heuristic; • stochastic; • population based; • continuous; • global.	The algorithm is initiated with food sources (random solutions to the problem). First the employed bees explore the neighborhood of each food source and keep track of the best solution. Next, the onlooker bees choose a food source based on the fitness of the solution. Finally, scout bees neglect the food sources that have not been improved and generate a new random solution. This cycle continues until the termination condition is met.	[30][1]

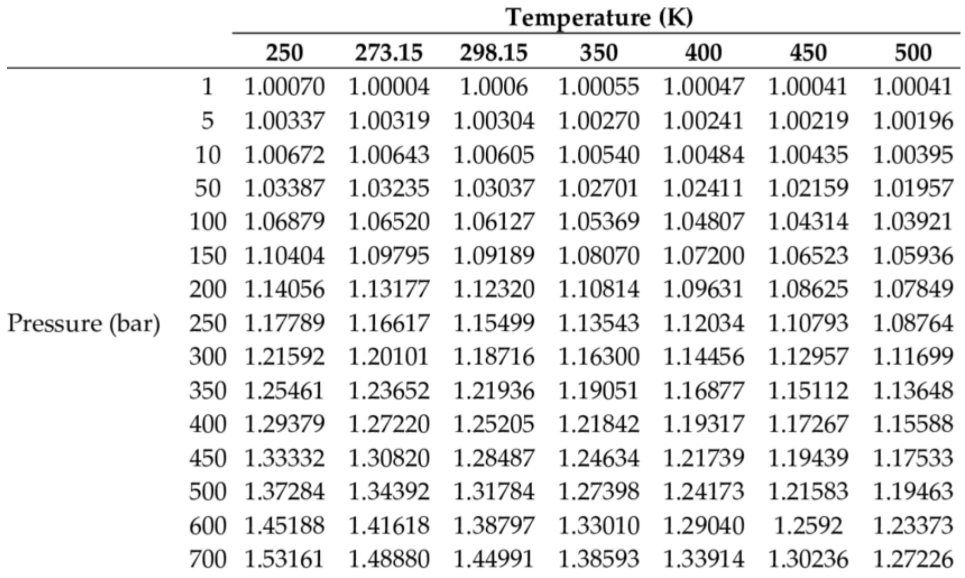
[1]Reference uses the algorithm for simulation optimization.

B
Supplementary figures and tables

This appendix provides additional supplementary figures and tables belonging to various chapters.

B.1. Methodology chapter

Figure B.1 illustrates a table that contains the Z-values (or z-factors) for a number of pressure and temperature combinations. The table is hard coded in the compressor and pipeline models described in Section 3.1.1. When the model is provided with an unknown pressure or temperature, nearest neighbor interpolation is used.



		Temperature (K)						
		250	**273.15**	**298.15**	**350**	**400**	**450**	**500**
Pressure (bar)	1	1.00070	1.00004	1.0006	1.00055	1.00047	1.00041	1.00041
	5	1.00337	1.00319	1.00304	1.00270	1.00241	1.00219	1.00196
	10	1.00672	1.00643	1.00605	1.00540	1.00484	1.00435	1.00395
	50	1.03387	1.03235	1.03037	1.02701	1.02411	1.02159	1.01957
	100	1.06879	1.06520	1.06127	1.05369	1.04807	1.04314	1.03921
	150	1.10404	1.09795	1.09189	1.08070	1.07200	1.06523	1.05936
	200	1.14056	1.13177	1.12320	1.10814	1.09631	1.08625	1.07849
	250	1.17789	1.16617	1.15499	1.13543	1.12034	1.10793	1.08764
	300	1.21592	1.20101	1.18716	1.16300	1.14456	1.12957	1.11699
	350	1.25461	1.23652	1.21936	1.19051	1.16877	1.15112	1.13648
	400	1.29379	1.27220	1.25205	1.21842	1.19317	1.17267	1.15588
	450	1.33332	1.30820	1.28487	1.24634	1.21739	1.19439	1.17533
	500	1.37284	1.34392	1.31784	1.27398	1.24173	1.21583	1.19463
	600	1.45188	1.41618	1.38797	1.33010	1.29040	1.2592	1.23373
	700	1.53161	1.48880	1.44991	1.38593	1.33914	1.30236	1.27226

Figure B.1: The Z values used for the calculation of the volumetric density of hydrogen[47].

Table B.1 provides the variables required by the compressor model and their respective units.

Table B.1: Variables used in the compressor model.

Symbol	Description	Unit
p_1	initial pressure	bar
p_2	final pressure	bar
$T1$	initial temperature	K
$\eta_{compressor}$	compressor efficiency	$-$

Table B.2 provides the variables required by the pipeline model and their respective units.

Table B.2: Variables used in the pipeline model.

Symbol	Description	Unit
L	pipe length	m
v	flow velocity	m/s
D	pipe diameter	m
ϵ	pipe roughness	m
μ	dynamic viscosity of hydrogen	Pa
P_{H_2-poly}	Permeability coefficient	$cm^3(STP) \cdot cm/(cm^2 \cdot s \cdot Pa)$
A	cross-sectional area pipe	m^2
$p_{in} - p_{out}$	pressure gradient inside-outside	$Pa \quad (1bar = 10^5 Pa)$
d	pipe wall thickness	m

Table B.3 provides the permeation coefficients for hydrogen in HDFE pipes. These coefficients are used in the pipeline model. Again, nearest neighbor interpolation is used for pressures that are not explicitly provided in the table.

Table B.3: Permeation coefficients for hydrogen in HDPE pipes [35].

Pressure $[MPa]$	**Permeability coefficient** $[cm^3 \cdot cm/cm^2 \cdot s \cdot Pa]$
10	$5.77 \cdot 10^{-8}$
30	$4.17 \cdot 10^{-8}$
50	$2.84 \cdot 10^{-8}$
70	$1.99 \cdot 10^{-8}$
90	$1.68 \cdot 10^{-8}$

Table B.4 provides the variables required by the electrolyzer model and their respective units.

Table B.4: Variables used in the electrolyzer model.

Symbol	Description	Unit
P_{in}	input power	W
η_{ez}	electrolyzer efficiency	$-$
$P_{in,rated}$	rated input power	W
$P_{in,ramp}$	max ramp input power	W

Table B.5 provides the variables required by the fuel cell model and their respective units.

Table B.5: Variables used in the fuel cell model.

Symbol	Description	Unit
$P_{fc,t}$	output power	W
η_{fc}	fuel cell efficiency	$-$
$\dot{m}_{H_2}$	hydrogen input flow	kg/s
$m_{H_2,max}$	maximum hydrogen input flow	kg/s
$\Delta P_{fc,max}$	maximum ramp up output power	$W/'time'$

Table B.6 provides the variables required by the hydrogen storage model and their respective units.

Table B.6: Variables used in the hydrogen storage model.

Symbol	Description	Unit
$Q_{H_2-sto,t}$	hydrogen charge at time t	kg
$Q_{H_2-sto,max}$	hydrogen storage capacity	kg
$\dot{Q}_{H_2-sto-in,t}$	hydrogen storage input flow	kg/s
$\dot{Q}_{H_2-sto-in,min}$	minimal hydrogen storage input flow	kg/s
$\dot{Q}_{H_2-sto-in,max}$	maximal hydrogen storage input flow	kg/s
$\dot{Q}_{H_2-sto-out,min}$	minimal hydrogen storage output flow	kg/s
$\dot{Q}_{H_2-sto-out,max}$	maximal hydrogen storage output flow	kg/s
$\eta_{H_2-sto,charge}$	hydrogen storage charge efficiency	$-$
$\eta_{H_2-sto,discharge}$	hydrogen storage discharge efficiency	$-$
$SOC_{H_2-sto,min}$	minimum state of charge	$\%$
$SOC_{H_2-sto,max}$	maximum state of charge	$\%$

B.2. Results chapter

Figure B.2 illustrates the upper threshold component of all solutions found by PSO. Figure B.3 illustrates the lower threshold components.

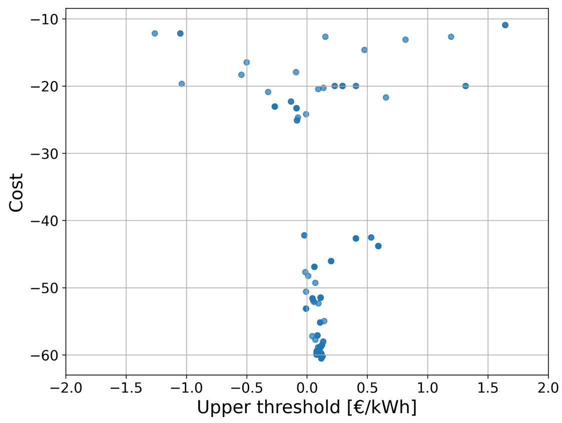


Figure B.2: PSO search space for the upper price.

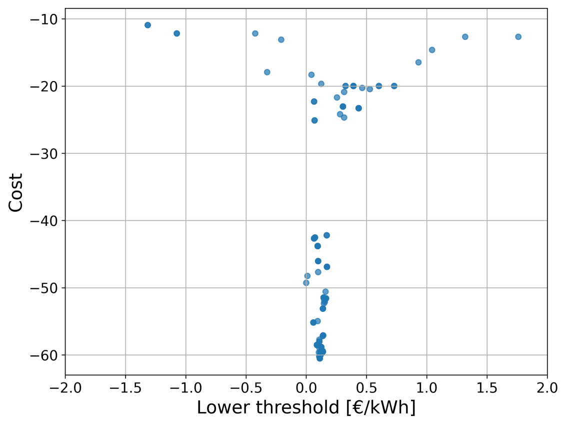


Figure B.3: PSO search space for the lower price.

Figure B.4 illustrates the upper threshold component of all solutions found by L-BFGS-B. Figure B.5 illustrates the lower threshold components.

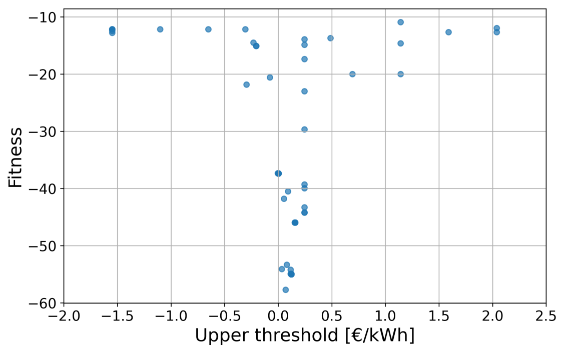


Figure B.4: Parallel L-BFGS-B search space for the upper price.

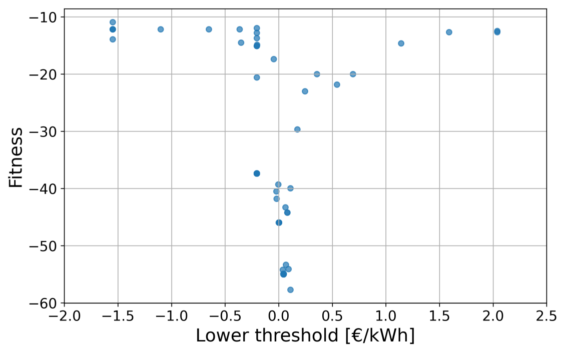


Figure B.5: Parallel L-BFGS-B search space for the lower price.

Figure B.6 illustrates the upper threshold component of all solutions found by GA. Figure B.7 illustrates the lower threshold components.

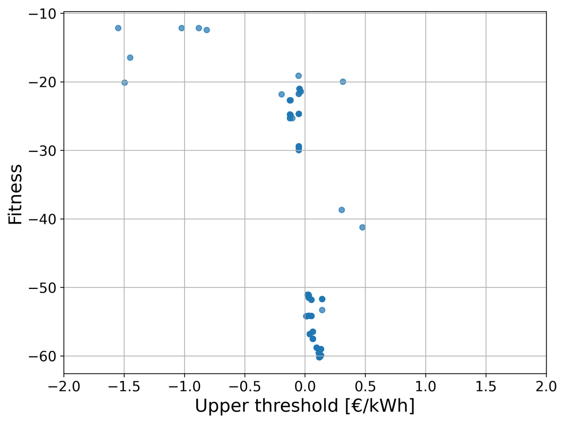


Figure B.6: GA search space for the upper price.

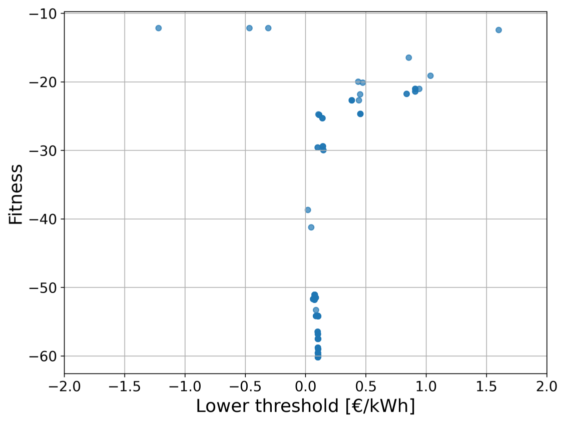


Figure B.7: GA search space for the lower price.

Figure B.8 illustrates the progression (and convergence) of the three algorithms during a number of iterations for the energy hub scenario.

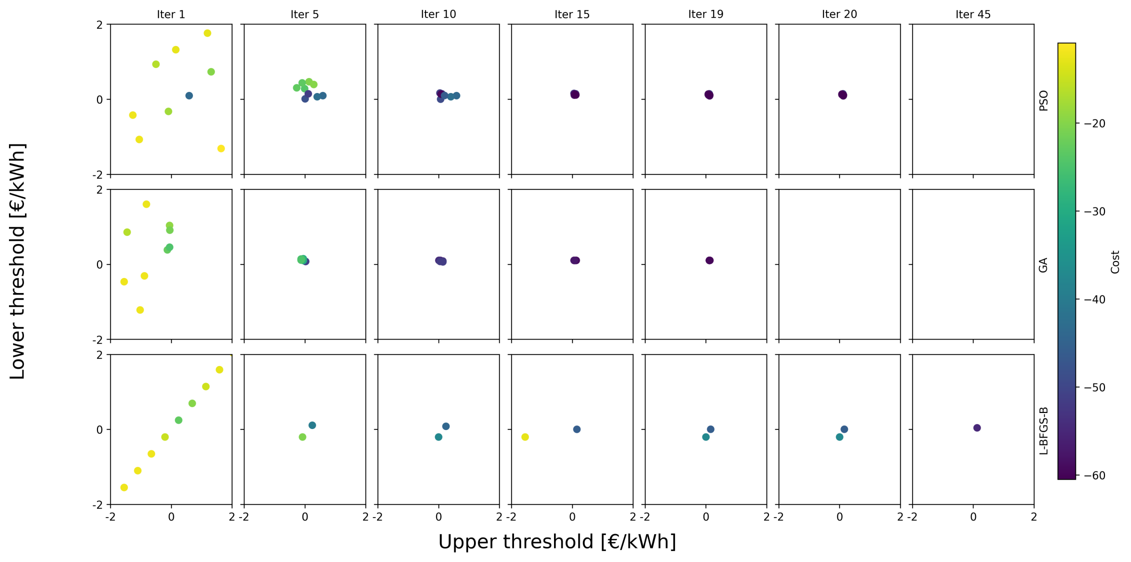


Figure B.8: Convergence of all algorithms for each generation.