MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

Optimizing the Energy Profiles of a Building's Apartments to Support Decision-Making in Demand Response Programs

LICENSE THESIS

Graduate: **Zoltan VARGA**

Supervisor: **Conf. Dr. Eng. Bianca Cristina POP**

2024

Contents

Chapter 1. Introduction - Project Context

As electronic devices found their way into our homes and became an indispensable part of our lives, the energy demand of residential buildings increased. This increase in consumption places a significant burden on the grid. It is an issue especially during peak hours when besides the residential buildings, the commercial sector has a high demand as well.

The electrical grid in our nation, and worldwide was built several decades ago, when demands as nowadays were inconceivable. Despite the effort to modernize the infrastructure, some limitations still apply in the delivery of power to consumers. Since an electrical fallout on a regional, and even on a local level would have a significant impact on the lives of thousands of people, electricity generation and distribution companies and authorities have the responsibility of mitigating this risk. Meeting the demand of consumers at any time of the day can be achieved by several methods such as turning on power generation plants, buying energy from neighboring regions, or utilizing renewable energy sources. All the options mentioned before bring significant shortcomings such as increased costs, and availability issues. As recent practices show, consumers can contribute to the reliability of the network by participating in demand response programs.

As suggested by Shewale et al in their overview [1], residential users consume around 30-40% of the total energy produced in the world. By involving them in programs that aim to reduce the stress from the grid, a more sustainable, and less costly power system can be achieved, without extending the power generation capacity.

Demand response (DR) programs were developed to introduce the consumer side to the reduction of the load from the grid. The initiative is taken by electrical energy distribution companies which are in charge of delivering an amount of electrical energy in line with the consumption. The concept behind such strategies is that each small contribution to the reduction of stress can add up, and be impactful. Consumers are rewarded for participating in such programs since they need to change their consumption habits temporarily.

There are two approaches to demand response programs, according to [2] and as illustrated in Figure 1.1. A technique consists of remotely controlling the energy consumer devices of users that explicitly signed up for it. Thus the operator sends a signal to the connected devices to shut down or reduce power usage in case of an increased overall consumption. As illustrated by Shewale et al in [1], to implement such a scenario a smart grid needs to be implemented on top of the existing grid. This means the installation of sensors and communication devices, which enables the real-time data gathering, pattern recognition, and management of electricity consumers. This may not be an issue in recently expanded cities, where buildings were constructed not long ago, however, it certainly raises challenges in urban areas with a considerable historical heritage. **In these kinds of neighborhoods consisting of old buildings the costs and the impact of introducing smart meters, and controllers might outweigh the benefits.** The other

approach to demand response programs is differentiated pricing for peak and off-peak hours. This is effective since both residential and commercial users chase to reduce their operational costs, thus they adapt to the changing prices over a day as a response to the fluctuating costs defined by the operator.

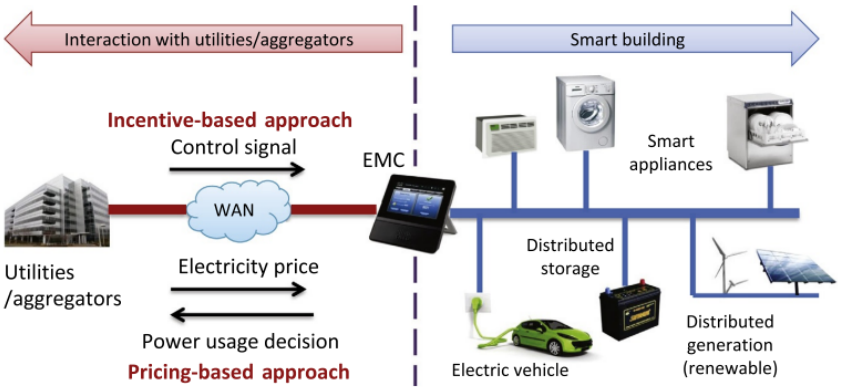


Figure 1.1: Incentive-based approach vs. Pricing-based approach in DR programs [2]

There are multiple possibilities to reduce the burden of the grid during peak hours. When looking at the curve which shows the consumption in relation to time, the following scenarios can be identified, according to [1] and illustrated in Figure 1.2: peak clipping, valley filling, load shifting, load reduction, load growth and flexible redistribution of load.

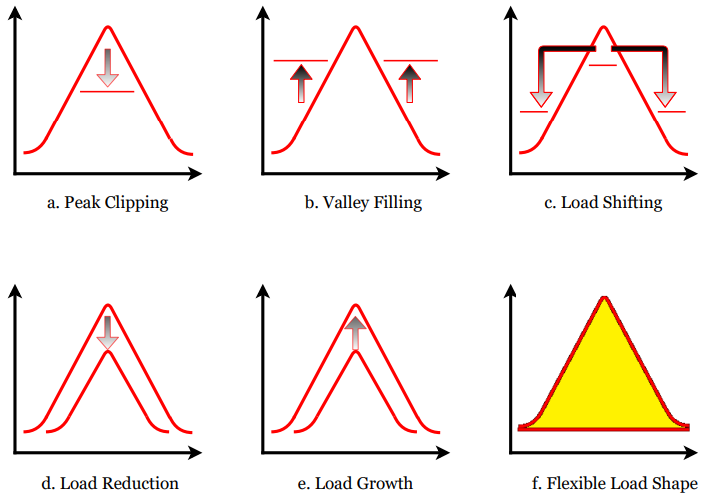


Figure 1.2: Demand side management techniques [1]

To further emphasize the importance of demand response programs, we can mention the state of California as an example. Without the implementation of a demand response program, a major electricity fallout would have been inevitable. **In the summer of 2023, the North American Electricity Reliability Corporation predicted the energy reserves of the state during peak hours to be at -11.9%. Similar to 2022, without DR programs California would have experienced a major energy fall-out in the middle of one of the hottest summers, which would have meant for residents the inability to moderate indoor temperatures using air conditioners. [3]**

As it was mentioned before, old residential buildings represent a significant hardship when developing demand response programs in historic neighborhoods of cities. Due

to the infeasibility of installing smart meters and intelligent devices, other approaches have to be considered. **In the current work, it will be presented a solution for involving such buildings in DR programs. It will offer a tool for utility companies to set an overall target consumption pattern for buildings, based on which a schedule will be created for electricity-consuming devices. The solution should also take into consideration the usage preferences of each participant for each of their devices.**

The current document is divided as follows: chapter 2 will detail the set objectives while in chapter 3 existing solutions will be studied, after that in chapter 4 the solution developed is laid on theoretical grounds. In chapter 5 will be presented the design of the solution, as well as implementation details regarding the scheduling module developed, and the application which integrates it. Towards the end of the document, in chapter 6 the results of the algorithm will be evaluated based on test scenarios, and in chapter 7 instructions regarding the deployment and usage of the application are described. Finally, in chapter 8 conclusions will be drawn.

Chapter 2. Project Objectives

This section of the document describes the clear objectives that should be achieved by the end of this thesis.

2.1. Main objective

The main goal of the project is to elaborate a strategy that enhances the consumption patterns of residential buildings included in demand response programs. More precisely it is desired to develop an algorithm that makes a usage schedule for electricity consumer devices from a residential building. In the first step, the solution given will try to approximate the target total energy consumption over a day of the building. As a further improvement, the proposed solution should adjust the schedule in such a way as to minimize the impact on the comfort of residents. To further enhance the features offered by the application, it should identify a minimum subset of apartments that need to alter their consumption patterns for the entire building to approximate the target consumption curve.

2.2. Secondary objectives

1. **Study existing solutions in the field of load scheduling**

   As a first step, an extensive analysis of existing solutions in the domain of load scheduling implemented within demand response programs is needed. Solutions for maintaining comfort in residential buildings while doing scheduling have to be studied as well.

   As a result, besides making a clear picture of the domain, insight will be gained on what are the best practices applied in this field, and which algorithms are best suitable for solving the problem. By making a comparison between the articles in the domain shortcomings will be identified among different approaches, and the best methods can then be applied in this work, despite the risk of remaining average.

2. **Study the state of the art meta-heuristic algorithms**

   Since meta-heuristic algorithms are best suited for this optimization problem it is required to gain an overview of them. To achieve this it is needed to study various types, and design variants. By doing this, the advantages and pitfalls of each will be discovered, so an informed decision about which one to choose can be made.

3. **Formally define the optimization problem**

   After sufficient insight was gained on the topic, the next step is to define the given problem using rigorous notation. As the problem falls into the category of optimization problems a mathematical description has to be provided describing all its details. This will capture the representation of a residential building, the function that has to be optimized, the constraints that apply to the optimization process, and the structure of the output provided by the algorithm.

4. **Develop the scheduling algorithm**

   Having the best-performing approaches from the domain, the solution will be built based on them, while following the formalization of the problem. Also, the general architecture of the system will be designed in this process, before starting the implementation.

   After selecting the suitable meta-heuristic algorithm during the analysis section, it is necessary to map the problem onto it by making modifications. At this stage, the following main customizations can be identified: transform the input data such that the algorithm can perform operations on it in the least amount of time, implement the objective function that has to be minimized during the optimization process, also which encapsulate all goals and, as a last step of the customization, define the constraint functions of the optimization.

5. **Develop the method for selecting the minimum number of apartments to modify their consumption patterns**

   After the scheduling algorithm was developed it should be integrated as a subsystem of another algorithm that can identify a minimum subset of apartments from the residential building that have to modify their usage schedule such that the consumption of the whole building (consumption resulting from included and not included apartments) to approximate the target curve.

   A well-thought method has to be designed to be able to try out the most promising combination of apartments and integrate the scheduling algorithm such that the whole process runs in a reasonable amount of time.

6. **Develop an experimental prototype that integrates the proposed methods**

   For the scheduling solution to be adopted by energy distribution companies the provided solutions have to be incorporated into a web-based application that can be deployed in the cloud. The integration will provide a simplified and intuitive interface for the solutions. It has to provide functionalities such as uploading the data of multiple residential buildings, running the optimization in multiple scenarios, downloading the schedule, and evaluating the schedule produced.

7. **Evaluate the performance of the proposed methods and fine-tune them**

   After finalizing the implementation it is essential to evaluate the solution on datasets and compare it to the schedule provided by other implementations. By measuring the performance of the strategies further improvements can be proposed.

   As an initial step to this objective, a dataset reflecting a real usage scenario has to be created. After that, the scheduling algorithm will be tested on this dataset. To

be able to assess the quality of the methods, expressive evaluation metrics have to be developed.

Since multiple configuration parameters will be exposed in the development phase, due to the nature of the problem, during evaluation these should be fine-tuned to obtain the best performance, based on the metrics defined. At this stage, if the results are not satisfactory the implementation should be further refined. As a last step, it is necessary to compare the solution provided to existing solutions from the domain. Thus, the contribution to the domain can be evaluated, and a sense of where the provided solution positions itself compared to other approaches can be gained.

Chapter 3. Bibliographic Research

Several research papers were analyzed to gain a clear picture of the research field to which the current thesis is contributing. In this chapter, an overview will be presented of what has already been done in the field of scheduling algorithms for appliances from residential buildings.

D. H. Muhsen et al. [4] addresses the problem of reducing energy consumption in peak hours, thus improving operational costs, while reducing customer inconvenience. The proposed solution is divided into two levels. In the first level, a Multi-Objective Henry Gas Optimization (MHGSO) Algorithm is used to obtain the best starting time for operating each electricity-consuming device during the day such that to minimize the objectives while adhering to constraints. The generated candidate solutions are passed to the second level. On the second level, a Multi-Criteria Decision-making algorithm will rank the candidate solutions by computing the criteria weight (the criteria are the objectives) using the Entropy Weighting Method (EWM), and taking the best solution using the Technique for Order Preferences by Similarity to the Ideal Solution (TOPSIS), based on the weights of the criteria. The objective function is constructed from energy cost, end-user inconvenience, and peak demand. The inconvenience is measured as the difference between the preferred usage interval and the scheduled usage interval expressed in hours. The peak demand objective focuses on reducing the maximum consumption during a day, thus making the consumption curve as flat as possible. Unlike in the currently proposed solution, devices are not categorized, all of them are considered non-interruptable, and deferrable. Unlike in the proposed solution, the objectives are not normalized in this paper. Like in the solution proposed a minimum usage hour is considered for operating appliances.

Z. Luo et al. [5] tackle the problem of scheduling electricity-consuming devices from a building for the day ahead. They identify the problem of oversimplifying occupant behavior in related works. This oversimplification achieved with pre-defined schedules can lead to a significant impact on cost-effectiveness in case of a slight behavior change. The proposed solution consists of integrating a stochastic model using the Metropolis Hasting algorithm based on the Markov chain Monte Carlo method to simulate the behavior of tenants. The behavior of occupants can influence the consumption pattern of inflexible loads, and weather can influence the usage of ACs for the day ahead, also the model can predict what configuration parameters might be set by users in the upcoming day (turn-on moment, turn-on time for each device, temperature setpoint, deferrable load flexibility window). So predicting these can contribute to the stability of the schedule as there is no available data referring to preferences from users for the next day, and a rigid assumption would surely lead to an impact on comfort. The resulting model produces input data for the scheduling algorithm. Based on this model, the algorithm will then make an optimal scheduling for flexible loads from the residential building, while taking into account the likely behavior of tenants resulting from the stochastic model. The

data gathering for training the stochastic model means the usage of smart devices, however, after training the usage of this method doesn't require real-time input from devices connected to a network, thus being suitable for usage in old residential buildings. For solving the multi-objective non-linear optimization problem the Non-dominated Sorting Genetic Algorithm (NSGA-III) is used. The Technique for Order Preference by Similarity to an Ideal Solution (TOPSIS) method is used to select from the solutions lying on the Pareto front. The objective function proposed is complex, as it has many components. It comprises of daily electricity costs based on consumption, dissatisfaction of tenants, the average ramping index, and a sustainability factor: the daily CO_2 emissions. The average ramping index shows how abruptly consumption changes from one hour to another, and the CO_2 emission is computed based on the estimated emissions related to the generation and distribution of power brought from the grid. The dissatisfaction of tenants consists of the thermal comfort, and shifting loads away from preference. The discomfort associated with thermostatically controlled loads is computed based on the difference between the preferred temperature intervals given by occupants, and the range established by the algorithm. Likewise, the discomfort of shifting loads is given by the number of hours a flexible load is shifted compared to the preferred values. Two main categories of devices are considered by this solution: thermostatically controlled loads, and flexible loads. Thermostatically controlled loads are represented by air-conditioning systems which activate on the command of a thermostat. For flexible loads, they consider appliances that can shift the time interval they are used at, but they cannot interrupt their operation. Since deferrable loads can have different power consumption based on the cycle they are executing, this solution considers the different power consumption of each devices in different stages of their operation. The performance of the algorithm wasn't compared to any existing solution. Unlike in the solution proposed by this work, the schedule is made for the day ahead which has as requirement the prediction of consumption patterns. Thus only the scheduling algorithm can be compared. Both methods use a meta-heuristic algorithm to solve the problem at hand. However, the solution presented by the paper considers only non-interruptable devices as flexible loads, but it defines different consumption ratings for the specific cycles of each. Another difference is in the approach of defining comfort: they are considering ACs so taking into account thermal comfort is mandatory. When it comes to dissatisfaction generated by shifting flexible loads, this solution measures discomfort by the amount of shift from the preferred intervals.

A. Shewale et al. in the paper [6] are aiming to optimize energy consumption costs of household appliances. The solution to this problem consists of a scheduling algorithm for appliances that helps minimize energy cost and peak-to-average ratio. The optimization is based on initial scheduling, the power rating specification for each device, and the time duration of using each appliance. The data is provided by residential users. Since the electricity cost is an important factor for the outcome of the scheduling, the authors propose two pricing schemes that are used to define the prices over a day: time of use pricing (at some time intervals prices might be lower than others, depending on the load) and critical peak pricing (for peak time interval the cost of consumption is very high). While the effectiveness of the algorithm is explicitly expressed for homes equipped with smart appliances, since the output is a schedule it can be used for residential buildings without such smart infrastructure. The authors of the paper deploy an optimization algorithm from the field of combinatorial optimization named the knapsack problem that aims to find the configuration of putting elements into bins such as to maximize value.

Each hour of the day is modelled as a bin, the value of each item is consumption cost of each device, and the weight is given by the device's power rating. Two implementations of this algorithm are used: one for serial scheduling (devices are non-interruptable) and one for parallel scheduling (devices are interruptable). The objective function consists of the electricity cost over a day for each device (computed from the energy price, the average consumption, and the status of each device at each hour), and a component of peak-to-average ratio. The proposed solution doesn't consider comfort. Devices are categorized as shiftable and non-shiftable. Shiftable devices are further characterized as interruptable or non-interruptable depending on whether they are using serial scheduling or parallel scheduling of devices. The algorithm is compared to an existing solution that is also based on knapsack technique. It was shown, that the proposed solution outperforms the existing one in both energy cost and peak-to-average ratio. The best results were obtained in the case the operation of shiftable devices could be interrupted. Despite that the solution in this paper uses dynamic programming instead of meta-heuristic algorithms, its approach to device categorization is similar to the scheduling algorithm proposed by this thesis. Namely, the presented solution considers appliances to be non-shiftable or shiftable, the latter having been further decomposed into interruptable or non-interruptable.

The problem tackled by Y. Wei et al. [7] is regarding the stress reduction of the electrical grid in rural areas where newly installed air-conditioning systems represent an unexpectedly high-risk factor for the already fragile grid. The authors of the work solve this problem by aggregating neighboring AC equipment with the same specification using so-called Load Aggregator entities that will deploy the scheduling algorithm for the ACs of their clients. The solution has two parts, one for load scheduling and the other for day-ahead load forecasting using neural networks. The Power Company queries the forecasted consumption values for the day ahead from the model trained on historical data. The forecast is then fed into the system of the aggregator which will then elaborate a preferential pricing strategy and a compensation price paid by the aggregator to participating households. This new pricing and the target consumption values represent the input parameters of the scheduling algorithm. ACs are then controlled remotely, either by turning them on or off, or by adjusting their thermostat. For this solution to work AC devices, or their thermostat needs to be connected to a central network, to manage them remotely. Thus making them incompatible with old residential buildings. For the scheduling part, the authors approached the problem with an Integer Linear Programming algorithm and a Genetic Algorithm. During the testing phase, the Genetic Algorithm converged faster to a solution, while the fitness values remained similar for the two algorithms. Besides having a better electricity price if ACs are used outside peak hours, participants also get compensation based on the extent they are reducing their power consumption (a large difference in consumption means more compensation, but a higher impact on the comfort of the tenant as well). Thus the objective function is expressed as a minimization of the compensation given by the load aggregator. This is a slightly different approach, than the ones encountered so far, however it achieves its objectives. Comfort is comprised of thermal comfort, which is treated as a constraint. If the thermal comfort values (which are directly related to temperatures) are out of a range, then the solution is invalid. Only AC devices are considered. Based on previous classifications, these types of electricity consumers can be categorized as deferrable and interruptable devices.

L. Guanghua et al.[8] tackle the problem of lowering consumption peaks and reducing consumer energy costs while maintaining user comfort. The solution has several

parts. Residential consumers input their device's data, and usage constraints through a user interface, then the energy scheduler creates a schedule for the devices entered into the system obeying both utility and user constraints. The result is then communicated to each device via the network. The solution proposed by the authors of this paper considers renewable energy sources and energy storage devices as tools for reducing the operational costs of residential houses. The solution provided relies heavily on the usage of Advanced Metering Infrastructure (AMI), thus making it incompatible with old residential buildings. AMI provides real-time measurement and communication with home appliances, making remote control possible. Despite this, by providing a schedule for appliances, and taking out the remote control component, it can be deployed in old residential buildings. The algorithm used for solving the problem is the Crossover Mutation Enhanced Wind-driven Optimization (CMEWDO). The WDO algorithm consists of initializing wind parcels on the search space, each representing a possible solution, then moving these parcels based on velocity and direction to explore new candidate solutions. It is enhanced with crossover and mutation operations (similar to genetic algorithms) to achieve better exploration of the search space. These operations are applied to each search agent after each iteration. The objective function is the weighted sum of the comfort metric (expressed in percentage), the price of the electricity bill (obtained by adding the product of the total power consumption and its price for each hour), and the Peak-to-Average demand ratio (measures the ratio of peak demand to average demand). Comfort is defined as the time difference between the preferred usage pattern and the scheduled usage interval. It is expressed as a percentage of the number of hours delayed from the maximum waiting time it can have. The types of devices considered during the optimization process are interruptable devices (functioning can be interrupted or delayed), non-interruptable appliances (once they are turned on, they must complete their operation), power regulating appliances (these are smart devices that can adapt their power consumption), critical appliances (these devices cannot change their operation schedule). The solution is compared with the Whale optimization algorithm (WOA), Enhanced differential evolutionary algorithm WDO (EDE), and Wind-driven optimization (WDO). It was shown that in all three of the objectives, this solution performed the best reduction of the electricity bills (by 26.21%) while having an average delay of just 1.87 hours for interruptable and non-interruptable devices. The Peak-to-average ratio was 37.7%, obtaining a much smoother consumption curve. Similar to the algorithm presented by the current thesis, the solution of this paper uses meta-heuristic algorithms, it normalizes the values of the objective function's components and has a similar approach to categorizing appliances, but, it introduces a new category: non-interruptable devices.

A. A. Dashtaki et al. [9] tackles the problem of lowering the electric energy costs of smart homes, by optimizing consumption patterns of appliances and ensuring user privacy. The problem is solved using a multi-objective optimizer to reduce costs and maintain user comfort of smart homes with devices being able to both consume and inject electricity from the grid. The solution is constructed using a meta-heuristic called the Bacterial Foraging Optimization Algorithm (BFA), and the Improved Tabu Search Algorithm (ITS). Three different candidate algorithms are evaluated, one for each optimization algorithm used and one that combines them. Unfortunately, it is not detailed how the two algorithms are combined. The objective function is defined as the multiplication of the cost of operating the smart home and the peak-to-average ratio. The cost considers the revenue from injecting electricity into a grid using storage devices, the revenue from participating in the demand response program, and the cost of the energy

consumed. Although it is mentioned as an objective, the objective function doesn't have a component measuring the comfort of users. The following device types are considered: non-deferable devices, interruptable devices, and non-interruptable devices.

Z. Chen et al. [10] address the problem of high electricity demand in the evening hours, situation which jeopardizes the potential of photovoltaics since they cannot counteract evening consumption by producing energy during dusk. The proposed solution for this problem is based on a scheduling algorithm implemented using the Non-dominated Sorting Genetic Algorithm (NSGA-II). The solution is based on the following input data: electricity price, user schedule preference, and user interference (meaning that if the user changes his comfort intervals, the algorithm reevaluates the schedule). Appliances are classified into four classes: interruptable and deferrable, non-interruptable and deferrable, non-interruptable and non-deferrable, and air-conditioners. Each has its specific operational constraints. The authors start scheduling in stages meaning that each type of device is treated in a different stage of the optimization. Since non-interruptable and non-deferrable devices cannot be adjusted, they are scheduled first, after which the remaining device types are scheduled. As there is no need for real-time monitoring and controlling in this solution, it would be suitable for old residential buildings. The objective function consists of the electricity cost and the discomfort produced by the schedule. The comfort objective is computed separately for each type of device: for deferrable and non-interruptable appliances the discomfort is given by the difference in hours between the scheduled time and the minimum allowed start time. For AC loads the comfort metric is given by the widely used predictive mean vote (PMV) index. It is insightful to note that at evaluation the authors constructed besides a weekday model, a weekend model as well.

Y. Liu et al. [11] try to come up with a solution for the problem of optimizing electric energy bills while improving the comfort of the users. They propose a multi-objective optimization method to make a trade-off between the two objectives. In their solution, they build a hybrid meta-heuristic-based algorithm. Thanks to its fast convergence the Multi-objective Particle Swarm Optimization (MOPSO) algorithm is used as a basis, which is enhanced with the genetic operator of NSGA-II to ensure diversity. To choose a solution from the Pareto-optimal solution set the TOPSIS method is used. They consider the following types of devices: controllable and non-controllable devices. Controllable devices can be further decomposed into non-interruptable devices, interruptable devices, and power-shiftable devices. The objective function consists of the consumption price and the comfort fitness value. The comfort penalty for each type of device is computed as follows: for non-interruptable devices, the penalty is given by the ratio between the difference in scheduled starting time and preferred starting time and the maximum number of hours it can be delayed. Similar to the solution proposed by the thesis, for interruptable devices the authors propose a parameter defining the minimum number of hours a device should be used. For interruptable devices the comfort fitness is defined as the average delay time of scheduled time slots. The discomfort created by power-shiftable devices, such as air conditioning systems, is measured as the deviation of the actual temperature from the ideal range. The total discomfort is then fed through the sigmoid function to obtain a normalized value for the comfort objective.

To conclude this chapter a comparative analysis will be made between the studied solutions and the approach taken by this thesis. All the consulted papers had as their main objective the cost reduction of electricity by shifting consumption. The solution proposed by this thesis however considers as its objective to approximate a target con-

sumption curve set by the electricity distributor. Despite the apparent difference, this approach can be transformed into a cost-reduction objective in one step. As Table 3.1 summarizes, only 2 papers adapt an algorithm other than a meta-heuristic one. As underlined by the literature, meta-heuristic algorithms are well suited for multi-objective optimization problems. Despite researchers showing a clear preference for evolution-based algorithms for solving these types of tasks, the solution of the current thesis leverages the abilities of swarm-based algorithms. When looking at the objective functions used one can observe that the cost of electricity consumed over the day was the most used one. Since almost all the papers cited set cost reduction as objective, it is inherent to have costs quantified in their solution's objective function. The nature of the problem solved by this thesis asks for a function that minimizes the distance between a target and an actual consumption curve, thus Euclidean distance is considered as the objective function. It is insightful to examine the choices of representing discomfort produced by a schedule. The majority of papers included the delay between the preferred starting time and actual scheduled time of the operation of an appliance as discomfort factor. It can be argued that this approach restricts the possibility of scheduling a device to only once a day. The method presented by this thesis proposes to have a value representing discomfort for each hour for each device, thus being able to schedule a device multiple times a day. It can be deduced that classifying devices in just two classes might not capture well reality, therefore in this aspect, the approach taken by the current thesis might be flawed compared to other methodologies. It was identified, that besides the currently presented approach, only the paper made by L. Guanghua et al. [8] takes into account the magnitude difference of the values given by different objective functions. By combining them unnormalized, an irrelevant importance is introduced between the objectives (an objective function having greater values will count more in the final fitness, than others). To overcome this imbalance, we bring all values in the range $[0, 100]$.

Reference	**Objective**	**Meta-heuristic-based**	**Suitable for old buildings**	**Is comfort considered**	**Number of classes used to classify appliances**
[5]	Cost reduction based on day-ahead prediction	X		YES	2
[7]	Cost reduction based on day-ahead prediction			YES	1
[4]	Cost reduction	X	X	YES	1
[6]	Cost reduction		X	NO	3
[8]	Cost reduction	X		YES	4
[9]	Cost reduction	X	X	NO	3
[10]	Cost reduction	X	X	YES	4
[11]	Cost reduction	X	X	YES	4
This approach	**Curve approximation**	**X**	**X**	**YES**	**2**

Table 3.1: Comparing the studied papers with the solution proposed by this thesis

Chapter 4. Analysis and Theoretical Foundation

In this chapter, the theoretical ground will be laid for the solution proposed. Also, multiple implementation possibilities will be analyzed that are presented by the literature. Several definitions, and concepts will be presented, so the reader will gain an understanding of their meaning and also can see the reasoning behind choices made.

4.1. Theoretical foundation

In this part, domain-specific concepts will be presented, together with a high-level overview of the algorithms used and the mathematical description of the problem.

4.1.1. Demand response strategies

The subject of demand response (DR) strategies has been extensively studied in the specialty literature. There are various methods proposed to involve the demand side of the electrical grid to lift off the load generated during peak hours.

As shown by research in the field, there is no doubt about the potential that electricity-consuming appliances hold in reducing energy consumption during peak hours. In the study made by Pothitou et al [12], it was shown that in 2012, in the UK, 23% of household electricity use was produced by Information Communication Technology devices proportion which has only increased since then. The potential of these devices is underlined by [13], as these types of devices can often store electricity, and the charge periods can easily be shifted within a time frame.

Pallonetto et al [14] highlight the benefits of demand response programs. Besides the perks that can be directly obtained by users, such as energy cost reduction, they also highlight the social and environmental impact of such strategies. Making the grid more reliable means less discomfort for the community in the region, and no crises generated by power outages. DR strategies facilitate also the inclusion of renewable energy sources, as a purposefully shifted consumption means greater predictability that translates into better utilization of green energy sources.

As it was mentioned in the previous chapters, demand response programs (DRP) can be divided into two categories: pricing-based and incentive-based. The study in [15] has shown that incentive-based programs contributed to 93% of load reduction in 2008 in the US.

The paper [13] admits the benefits of DRPs in residential buildings, and after defining what a demand response program is it discusses various methods that can improve the energy efficiency and flexibility of involved buildings. It was shown by [13] that buildings having installed renewable energy sources, storage systems, and HVAC systems are the most flexible and can fully harvest the benefits of such programs. The most flexibility in both summer and winter seasons was obtained by installing such types of systems. The

same paper mentions that optimization algorithms that solve problems associated with DRPs have to solve the energy cost reduction in a reasonable time. Flexibility time windows are defined by the same work as time frames in which appliances can be operated freely without imposing discomfort on the user.

4.1.2. Comfort in demand response strategies

According to the authors of the paper [16], thermal comfort often oppresses financial benefits in the eyes of occupants, i.e. building occupants are keen to spend more on energy bills to maintain their comfort level. According to the same paper, thermal comfort evaluation metrics can be classified into the following four categories: analytical, adaptive, practical-experimental, and bioclimatological.

Another paper that discusses and tries to measure comfort is written by S.N. Bragagnolo et al. [17]. Their solution is tailored for an imaging center in Argentina and uses a genetic algorithm to solve the optimization problem. Besides considering electrical costs, they also keep thermal comfort in mind when operating the air conditioning system.

The thermal comfort in this paper is defined based on the ISO 7730 standard and uses the Predictive mean vote metric. PMV is a tool for indicating the average thermal discomfort a population would feel under given conditions. The values on the metric are on a seven-point scale, where 0 means neutral, negative means cold, and positive means hot [18]. The mathematical model behind considers the heat balance of the human body, which changes depending on the air temperature, mean radiant temperature, air velocity, humidity, metabolic rate, etc.

The objective function defined in this paper (4.1) consists of the sum of the electrical energy cost and the thermal cost multiplied by a weight which shows the *trade-off between comfort and price*. The electrical energy cost (C_{EE}) considers the price established by the energy supplier and the total demand. The thermal cost (C_{th}) (4.2) is the absolute deviation of the PMV for a building in a given period, with the possibility to weigh the sensation of each room separately. Furthermore, the projection of total demand (P_l) includes both the consumption prediction of uncontrollable loads and the consumption prediction of controllable loads. The researchers are interested in minimizing the objective function. The problem is constrained by the maximum consumption of devices, and the allowed comfort values for each room.

$$f_{obj}(P_{ac}) = C_{EE}(P_{ac}) + w * C_{th}(P_{ac}) \tag{4.1}$$

$$C_{th}(T_i) = \sum_{k=1}^{n} \sum_{j=1}^{m} c_j |PMV_j(T_{ij}(k))| \tag{4.2}$$

4.1.3. Optimization problems

Optimization problems provide a formal description of problems in which one or more measurable factors need to be improved.

As the given definition states, it is mandatory to have a measurable feature, which then can be modified in a manner that the new outcome of the feature minimizes the distance from a target value. In other words, the solution to an optimization problem will always try to find the best solution from a set of solutions. The set of solutions can

be finite if we are speaking about discrete optimization problems, or infinite in the case of continuous optimization problems.

In an ideal situation, fine-tuned features can take any value. In such a case it is always possible to reach the target value, thus find the best solution to the problem. However, in practice, the values are constrained. Moreover, a single feature can have multiple constraints. As a result, the target value can only be approximated by the solution, and not reached. An optimization algorithm deals with finding a configuration of parameters that best approximates the ideal solution. The distance between the ideal solution and the current solution is given by the fitness function.

As stated in Sven Leyffer in his lectures [19], optimization problems can be formalized in the following manner:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) && (4.3)\\ \text{subject to} \quad & l_c \leq c(x) \leq u_c && (4.4)\\ & l_A \leq A^T x \leq u_A && (4.5)\\ & l_x \leq x \leq u_x && (4.6)\\ & x \in \mathcal{X} && (4.7) \end{aligned}$$

Where $f(x)$ is the fitness function, which takes as input x, the current configuration. $\mathcal{X} \subset \mathbb{R}^n$ imposes structural restriction on x. $l_c, l_A, l_x, u_c, u_A, u_x$ are bounds for the input configuration.

These problems can be classified considering the type of variables, constraints, and functions involved in their definition. Problems can be unconstrained, bound-constrained, linearly constrained, equality constrained, etc. Based on the variable type we can have continuous variables, discrete variables, state and control variables, and random variables. When looking at the cardinality of the set of objective functions two types can be identified: single-objective function optimization problems, and multi-objective optimization problems.

In our case, by abstracting from the problem statement, and adopting a high-level view, we can say that it is expected to create an algorithm that considers multiple objectives while obeying constraints. Matching the definition of optimization algorithms. The objectives consist of minimizing the distance from a target energy consumption curve and minimizing the impact on tenant's comfort. From the bibliographic research, we concluded that it is good practice to have different categories of electricity consumer devices to accurately capture reality. Consequently, the constraints are given by the specifics of each category. The two categories implemented are non-deferrable devices (devices which cannot modify their consumption patterns from the initially given one), and deferrable devices. We also introduced the possibility to set the minimum number of hours a device has to be used in case it is deferrable. This will be another constraint for the algorithm.

Bi-level optimization problems solve complex optimization tasks by organizing algorithms into a layered structure. The pipeline of methods is suitable for finding a solution by combining the strengths of each algorithm in a certain domain.

Sinha et. al. [20] provide a formalization of bi-level problems shown in figure 4.1. It can be seen that the two levels have distinct search spaces. The optimization from the upper level sends as a parameter its best solution to the optimization on the

lower level. Based on the received solution, the algorithm from that level elaborates a solution based on its objectives. The combination of the optimal upper and lower-level solutions is elected as the overall answer. According to the same paper, these types of architectures are employed in various applications such as transportation, network optimization, environmental economics, machine learning, and so on.

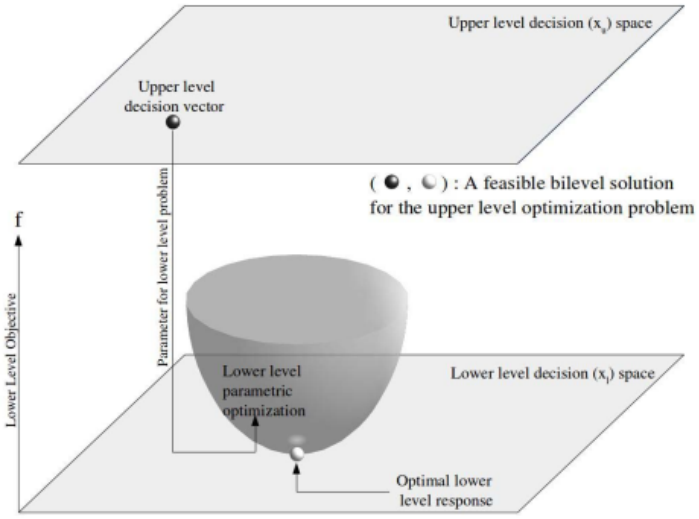


Figure 4.1: Illustration of bi-level problems' search space [20]

J.F. Camacho-Vallejo et. al. [21], in their comprehensive review, conclude that the most frequently encountered configuration of such multi-layered optimization solutions using meta-heuristic algorithms are architectures where the one on the top layer solves a part of the problem according to its objectives, then the obtained partial solution is further refined by the algorithm on the lower level, based on its particular objectives. This is called the nested approach, which is a computationally demanding architecture. According to the same study evolution algorithms are the most used ones.

4.1.4. Meta-heuristics in optimization problems

Minimizing or maximizing objective functions is not a trivial problem, since such functions are often not differentiable, nor continuous. Having in mind this, in [22] it is observed that there is a need for algorithms, that can find a Pareto Optimal solution for problems with multiple objectives, from a solution space which is given by the cartesian product of all the allowed parameter values. In such cases, meta-heuristic algorithms are the best choice.

Metaheuristic algorithms provide a framework for a generic set of problems, and which give an approximate solution to optimization problems, within a reasonable time. They are employed when the exact solution is too computationally demanding to find, and it is not vital to have an exact solution to the problem. These kinds of algorithms start with an initial solution, and continuously adapt the parameters of the solution to get close to the target values, i.e.: minimize the cost function. They will stop when the approximation is acceptable. They also are generic enough to be applied to a wide range of problems, only by tuning the input parameters.

Yang in [23] identifies two types of such algorithms: deterministic and stochastic algorithms. The steps taken by Deterministic algorithms can be reproduced since they

are based on a well-defined procedure, and a specific mathematical model is constructed for them. Conversely, stochastic algorithms involve randomness, thus they can not be easily reproduced, but due to this fact, they might get closer to the solution by chance. Running them in more iterations will lead to different results.

These types of algorithms usually combine two major properties: degree of exploration and degree of exploitation. The two properties combined in a balanced way will ensure the discovery of the globally optimal solution. Considering this, Yang [23] states that we can divide metaheuristic algorithms into trajectory-based algorithms, which have a higher degree of exploration, and population-based algorithms, which exploit the entities generated during the algorithm. For example, among population-based algorithms, we can name the particle swarm algorithm, which relies on the shared knowledge of the population, and among trajectory-based algorithms, we can name the simulated annealing, which moves a single entity through the search space in search of the global solution.

In the following, I will briefly present examples of meta-heuristic algorithms taken from F. Fausto's work [24] on gathering them in a single paper. Most of the scientists developing such algorithms took inspiration from nature (this kind of meta-heuristics are called bio-inspired algorithms), and they try to mimic the behavior of complex natural systems. Such algorithms can be classified as evolution-based, swarm-based, physics-based, and human-based.

The search starts with a population initialized randomly in the search space. At each iteration, the new position is computed based on the previous position and some criteria. The selection of an agent from the candidate set, and the definition of movement of each agent is specific to the type of algorithm we are speaking of. Next, we will present a few types of meta-heuristic algorithms

1. Evolution-based methods

   (a) Differential evolution

   The differential evolution algorithm applies at each iteration a set of mutations of the previous solutions. Individuals (candidates) are selected at random and to their value it is added a so-called differential weight whose differential variation is controlled by another 2 randomly chosen individuals. In this way, a new solution is generated by crossing the particularities of 3 randomly selected candidates from the previous set. In addition, a trial solution is used (which either received the value of the mutant, or the original value, based on a crossover rate). The results of the trial solutions are compared to the original solutions, and if they improve the fitness, then they are kept.

2. Swarm-based methods

   (a) Ant colony algorithm

   The ant colony algorithm was developed to find the optimal paths in a graph. Consequently, the method starts with N ants that are moving through the graph. Based on the current node, each ant will decide on which arc to continue its journey. This decision is taken based on a function depending on the pheromone level of an arc (how many times was that arc traversed by ants, and how long ago), and the distance of that arc (cost). In this way, each ant traverses several paths from the graph, but during backtracking, they release pheromones which will fade over time, marking how many times the path was taken by an ant.

(b) Bat algorithm

The bat algorithm models the movement of a swarm of bats towards the prey. In this algorithm, each bat is initialized with a random location, from which using its echolocation system navigates through the search space. The echolocation consists of the emitted frequency, loudness, and pulse emission rate. The frequency is specific to each bat, and it is updated at each iteration depending on some randomness. The loudness and pulse emission rates are modified based on predefined parameters. The position of the bat is updated considering its previous location, the location of the best solution from the swarm, and its signal's frequency. This algorithm introduces a local search mechanism. Specifically, a randomly chosen entity from the best solutions will perform a random walk at each iteration.

(c) Particle swarm optimization

The Particle Swarm Optimization algorithm is inspired by the behavior of flocking birds, which together are searching for food. Each bird is characterized by its current position, its best position yet, and its velocity. Each bird has access to the position of the global best position reached within its neighborhood. At each iteration, each agent's position is updated based on the previously enumerated parameters, as well as arrays of random coefficients.

3. Physics-based methods

(a) Simulated annealing

This method was made based on the heat treatment of solids to change their physical properties. Initially, a starting state is defined, and a cooling schedule. This schedule will contain all the temperatures during the transition (the values of these temperatures are monotonically decreasing). Besides the temperature, it is also defined for how many iterations those temperatures will be applied. After the initial step, by iterations, a new solution is generated by following a particular criterion around the current best solution. Then the costs of the two values (the current best solution and the generated solution) are compared, and if the new solution improves the outcome, it will be taken as the next best solution.

4. Human-based methods

(a) Harmony search

This algorithm mimics the way musicians create harmony by combining different pitches. The main structure of this algorithm is the harmony memory. This contains initially a random configuration of parameters, and over time will include the best solutions found yet. During the execution of the algorithm, a new candidate solution is generated using both exploration and exploitation. The new solution is based on a randomly selected solution from the memory map, the probability of selecting a certain solution is given by its random coefficient. If the fitness of the newly found solution is better, than the worst one's in the harmony memory, then it is replaced.

As one can deduce, each meta-heuristic algorithm is suitable for a specific range of problems. For instance, genetic algorithms are giving good results for combinatorial

problems in which the goal is to select a suitable subset from a set of possibilities. Another example would be the ant colony algorithm, which is used for searching for the best path between nodes in a graph. It can be applied in network design problems, or different nuances of routing problems.

In the case of our problem, we decided to solve it using a combination of meta-heuristic algorithms organized in a hierarchical structure. Further on it will be analyzed the feasibility of using a Genetic algorithm for the upper-level optimization, and the Harris Hawks meta-heuristic for the lower level.

4.1.5. Genetic algorithm

The genetic algorithm method is part of the evolution-based meta-heuristic class. It was inspired by the process of transmitting genes over reproduction processes from nature. The algorithm is described by D. Whitley [25] as follows. The solution to a problem is represented in a binary string-like data structure that models chromosomes. The combination of two agent's strings is the base operation that changes information between the two in the hope of achieving a superior solution.

The implementation of this algorithm is straightforward as it has a few steps which can be easily understood. The following are the steps of the algorithm: selection, crossover, and mutation.

In each iteration, there exist two populations: the current population and the intermediate population. Agents from the current population will be transferred into the next iteration's population by transiting the intermediate population after having applied the specific genetic operators.

The selection phase, as described in the paper [25], is done in a stochastic manner. There can be multiple approaches for selection such as roulette selection, tournament, or random. Each method makes the selection based on the fitness of individuals and a probability distribution. Agents with better fitness will have a higher chance of being selected to be in the intermediate population.

The crossover operation is applied to the solution of individuals from the intermediate population. The crossover operation randomly chooses two individuals from the intermediate population and interchanges their solution's bits. There are multiple ways of interchanging the bits, such as arithmetic crossover, one-point crossover, and multi-point crossover. They vary in the number of crossovers that are being made. Figure 4.2 illustrates how single-point crossover works; a randomly selected point in the bit string is selected and all the bits following are interchanged.

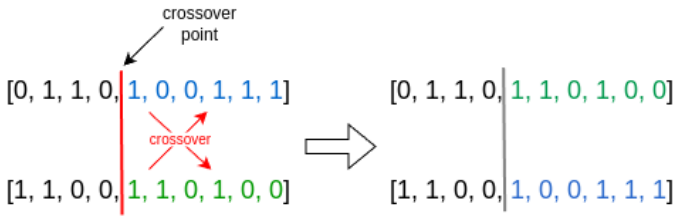


Figure 4.2: Showcasing how single-point crossover works

The mutation operation is applied to the results of the crossover. The goal of this operation is to slightly alter the solutions of an intermediate population to explore more of the search space. By applying this operator each bit of the solution string is or

swapped with another bit from the same string, or it will be set to a random boolean value, depending on the strategy applied.

Then the resulting agents from the intermediate population move to the next epoch's population.

4.1.6. Harris Hawks meta-heuristic

The algorithm was defined by A.A. Heidari, S. Mirjalili, H. Faris et al. in the paper [26]. The Harris Hawks optimization algorithm is inspired by the outstandingly collaborative hunting of Harris Hawks and belongs to the swarm-based algorithms category. The hunting method of Harris Hawks is outstanding due to their ability to gather the members of the same family as a swarm to hunt for food. The ritual is started by dispersing onto high trees within the realm and occasionally diving close to ground level to search for hiding prey. When the prey is localized they will coordinate to exhaust and confuse it by performing "surprise pounces". Meaning that hunters will cooperate to attack the prey from different angles. In the optimization algorithm, the Harris Hawks are search agents containing the candidate solutions of the problem, and the best solution of each iteration is seen as the prey. There are several stages, which will be described below.

1. Exploration phase

   In this phase, the search agents have to discover new positions within the search space hoping to find the global optimum. Two strategies are applied depending on a random variable with equal chances. The two strategies consist of the agents moving toward another agent selected at random or moving toward the best solution found yet. This is done to explore the search space as much as possible and to avoid being stuck in a local optima. Through the extensive search, they ensure that when the transition to the next phase is made some of the agents will discover, or are close to the ideal solution. This behavior can be described by the equation (4.8).

$$X(t+1) = \begin{cases} X_{rand}(t) - r_1|X_{rand}(t) - 2r_2X(t)|, & q \geq 0.5 \\ (X_{rabbit}(t) - X_m(t)) - r_3(LB + r_4(UB - LB)), & q < 0.5 \end{cases} \tag{4.8}$$

   Where X_{rand} is the position of the randomly selected agent, X_{rabbit} is the position of the agent with a solution resulting in the best fitness, $X(t)$ is the position of the current agent in the current step and X_m is the average position of the search agents.

   Random variables improve the diversity of the model. Despite that the current agent is moved into the vicinity of the selected random agent, the randomness ensures that new space is explored around them.

   In case it moves towards the best solution found yet, the algorithm first computes the mean of the population. The mean will be the average position of the agents (the average of their optimization parameters at each position). This way the new position of the agent will be around the best position, but shifted in the direction where the majority of the agents are. This way the search agents will move as a conglomerate towards the best agent.

2. Transition from exploration to exploitation

   The transition from the exploration phase to the exploitation phase is assured by the variable E, called the energy of the target solution.

   The decision based on which phase the algorithm should do operations is determined using the energy variable, called E. Initially, when the energy variable is large, exploration is done. Over time, as iterations pass, the energy will decrease, and a transition toward exploitation is obtained. The transition is modeled by the function (4.9).

$$E = 2E_0(1 - \frac{t}{T}) \tag{4.9}$$

   Where T is the total number of iterations, t is the current iteration number, while E_0 randomly changes at each iteration in the range $(-1, 1)$. One can observe, that as time passes, we multiply the random energy with smaller and smaller values, which will lead to the gradual decrease of the energy.

3. Exploitation phase

   As the number of epochs increases the energy value decreases. This triggers the exploitation phase, in which the best solutions found are refined, rather than generating new solutions.

   Several cases are considered in this phase: the first category of cases is given by the "behavior" of the target solution. Depending on a random number (r) the best solution can be approached directly by other search agents or by simulating a random movement. The other set of cases is given by the energy value(|E|): the values of the optimization parameters of an agent can be computed based only on the position of the agent containing the best solution found so far (target agent), or by moving the target agent in a random direction.

   As a result of these scenarios, we will have 4 cases:

   (a) Soft besiege ($|E| > 0.5,\ r > 0.5$)
   This is the case when the so-called target still has plenty of energy, but is not able to dodge the attacks. The new solution of the current agent is computed as the parameter-wise difference between the best and current agent, from which is subtracted a value modeling a sudden random movement, damped by the energy factor, see Equation (4.11). Where the difference between the solutions is computed as shown in Equation 4.10.

$$\Delta X(t) = X_{rabbit} - X(t) \tag{4.10}$$

$$X(t+1) = \Delta X(t) - E|JX_{rabbit}(t) - X(t)| \tag{4.11}$$

   The strength of the random movement J is computed based on Equation 4.12. On one hand, the random movement ensures that a larger space is covered during the search. On the other hand, it also guarantees the agent to escape from local optima.

$$J = 2 * (1 - r) \tag{4.12}$$

(b) Soft besiege with progressive rapid dives ($|E| > 0.5,\ r < 0.5$)

In this strategy, the agents will get close to the best solution found so far by doing one or two types of random movements, depending on which yields a solution with better fitness. To model the movement of the agent trying to reduce the distance from the target agent the levy flight concept is used (random walk, where the steps are based on lengths that follow a probability distribution). This being said, the next steps of the attackers are made by deciding to follow a path described by the equation 4.14 or adding to that movement the LF-based leapfrog movement. The decision is taken based on which next step will reduce the fitness of the current position as seen in 4.15. The function Z will give the next step containing a zig-zag movement as seen in (4.13), where S is a random vector of dimension 1xD (where D is the number of optimization parameters), and LF is the levy flight function.

$$Z = Y + S \times LF(D) \tag{4.13}$$

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X(t)| \tag{4.14}$$

$$X(t+1) = \begin{cases} Y, & \text{if } F(Y) < F(X(t)) \\ Z, & \text{if } F(Z) < F(X(t)) \end{cases} \tag{4.15}$$

(c) Hard besiege with progressive rapid dives ($|E| < 0.5,\ r < 0.5$)

This is almost the last stage of approaching the best solution. In this phase, search agents will reduce their average distance from the target agent. This ensures that the population will start forming a cluster by moving towards each other, and also moving towards the agent having the best solution. The base equation (4.16) is very similar to the previous step, in this case, the damped distance is not computed from the current solution, but from the average position of the whole population. In this case, all the agents contribute to the final solution.

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X_m(t)| \tag{4.16}$$

The equation giving the random movement is the same as 4.13, and the decision of choosing the position with or without the random movement is made the same way as in 4.15.

(d) Hard besiege ($|E| < 0.5,\ r > 0.5$)

This strategy is employed when the energy level is low. The new solution is computed only based on the distance of the parameters of the best solution found yet and a new position resulting from the distance of the current solution and the best solution damped by the energy factor. This solution will give a solution very close to the best solution. Thus exploring the search space near to the best solution. The equation (4.17) describes this movement.

$$X(T+1) = X_{rabbit}(t) - E|\Delta X(t)| \tag{4.17}$$

4.1.7. Justifying the decision of using these algorithms

- **Genetic Algorithm**

  For the upper-level optimization, a simplistic algorithm was needed that can inherently model taking subsets from a given set. For this task, a Genetic Algorithm is an obvious choice. It is a well-documented method that has proven its capabilities of exploring the search space by trying different combinations of parameters in a guided fashion. Our problem maps perfectly onto it since the inclusion of apartments can be encoded into a boolean bit-string of the length equal to the number of apartments.

- **Harris Hawks optimization algorithm (HHO)**

  By examining the scheduling problem statement, we identified the fact that it can be modeled as an optimization problem. Further on, due to the problem's multi-objective nature, and the high dimension of the solution space we examined the feasibility of meta-heuristic algorithms for obtaining a Pareto-optimal solution by balancing the two objectives. The dimension of the solution space is given by the large number of parameters that have to be adjusted. Although the definition of a parameter is included in the chapter providing a formalism for the problem, for clarity we mention it briefly. A parameter of the solution specifies that a certain device is turned on or off at a given hour. Therefore, we will have parameters for all devices from the residential building for each hour of the day.

  As stated in the paper which defines the HHO algorithm [26], this approach has an outstanding performance in solving multi-modal problems. Based on the qualitative tests made in the same paper, it was shown that given a parameter space with multiple optimum values the HHO was able to converge to the best solutions without being stuck in local optima. In Figure 4.3 we can see that despite having multiple local optima, the algorithm was able to converge towards the global optimum, thus gradually decreasing the fitness value of the best solutions found over the iterations. It can be also observed that after the 100^{th} iteration when the transition from exploration to exploitation is made, the average fitness decreases drastically while the algorithm focuses on the best solution found so far, and tries to explore the neighborhood of that solution.

  The authors of the paper further emphasize the ability of the algorithm to solve complex optimization problems by comparing it to other meta-heuristic algorithms such as Genetic Algorithms, Particle Swarm Optimization, Cuckoo Search, Bat Algorithms, and so on. They conclude, that while the performance of the mentioned optimization algorithms decreases with the increase in the parameter number, the Harris Hawks maintains its performance while still being able to balance exploration and exploitation, no matter the dimension of the search space. It was also shown, that HHO produces superior results when solving constrained problems, and is also able to explore the search space effectively.

  That being said, it is the best fit for our problem, in which we deal with a constrained problem, having multiple possible solutions. Furthermore, the large number of parameters we have makes us more confident in the usage of this meta-heuristic algorithm.

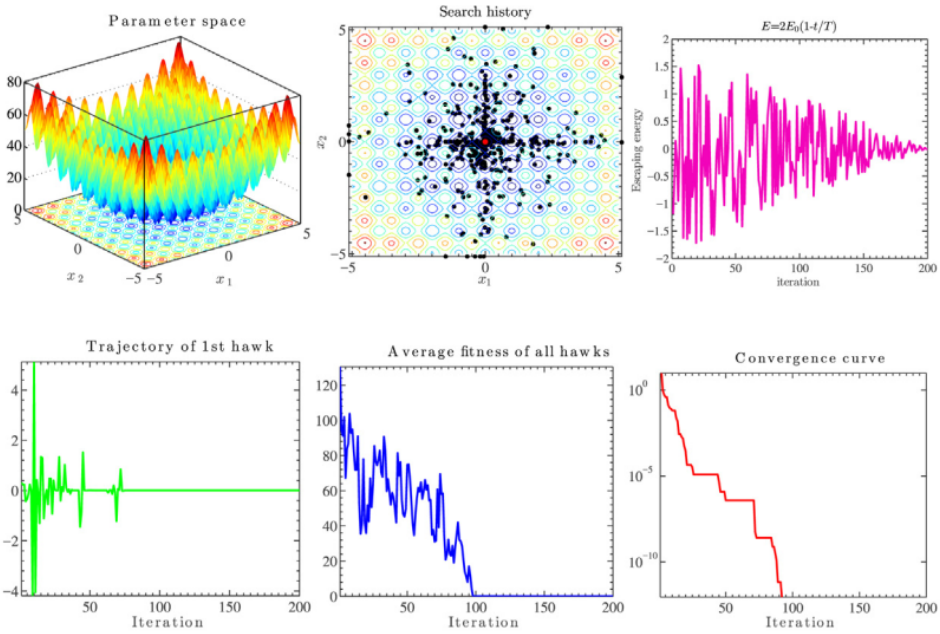


Figure 4.3: Visualization of the metric history for a search performed in a multi-modal space produced by a function with 2 parameters[26]

4.2. Analysis

In this section, the requirements of the application will be analyzed, and also a use-case description will be provided that will capture the mentioned requirements.

4.2.1. Use case analysis

In this subsection, a detailed analysis of the main use cases will be presented. In Figure 4.4 it is shown the use case diagram. It captures how the primary actor interacts with the use cases of the system. Three primary use cases were identified at the analysis stage, which will be detailed further on.

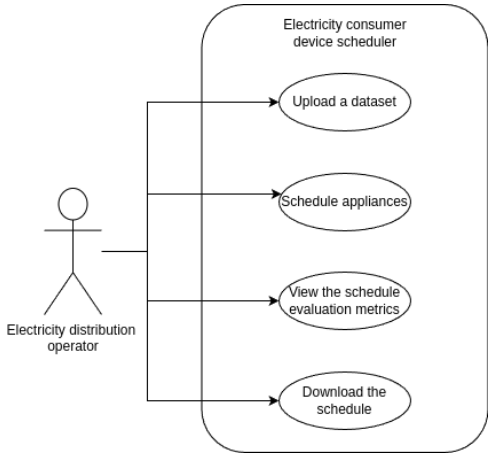


Figure 4.4: Use case diagram

1. **Title: Upload a dataset**

   - **Brief description:** The user can upload the data of a residential building.
   - **Primary actor:** The electricity distribution operator
   - **Pre-conditions:** None
   - **Post-conditions:** A new building setup is created and appears in the Operations tab, *Dataset codes* field.
   - **Basic flow:**
     1. The operator enters a dataset code (identifier).
     2. The operator uploads the device description file.
     3. The operator uploads the comfort definition file.
     4. The operator uploads the initial usage file.
     5. The operator clicks the upload button.
     6. The system processes the uploaded data.
   - **Alternative flows:**

     5a. The dataset code is already used
        1. The system shows a notification pointing to the invalid field.
        2. The operator enters another dataset code.
        3. The operator continues from step 5.

     5b. The device description file has an invalid format.
        1. The system shows a notification pointing to the invalid field.
        2. The operator uploads another device description file.
        3. The operator continues from step 5.

     5c. The comfort definition file has an invalid format.
        1. The system shows a notification pointing to the invalid field.
        2. The operator uploads another comfort definition file.
        3. The operator continues from step 5.

     5d. The initial usage file has an invalid format.
        1. The system shows a notification pointing to the invalid field.
        2. The operator uploads another initial usage file.
        3. The operator continues from step 5.

2. **Title: Schedule appliances**

   - **Brief description:** The operator uploads the necessary information and starts the scheduling for the selected dataset. The system makes a schedule for a day for every appliance.
   - **Primary actor:** The electricity distribution operator
   - **Pre-conditions:** The dataset is uploaded.
   - **Post-conditions:** The schedule and the metrics are passed to the user interface.

- **Basic flow:**
  1. The operator selects a dataset code.
  2. The operator selects DO_FIRST_LEVEL_OPTIMIZATION from the operation code list.
  3. The operator uploads the target curve file.
  4. The operator uploads the configuration file.
  5. The operator clicks the Submit Operation button.
  6. The system schedules the devices.
- **Alternative flows:**

  5a. The target curve file is invalid.
  1. The system shows a notification pointing to the invalid field.
  2. The operator uploads another target curve file.
  3. The operator continues from step 5.

  5b. The configuration file is invalid.
  1. The system shows a notification pointing to the invalid field.
  2. The operator uploads another configuration file.
  3. The operator continues from step 5.

  6a. An internal error occurred
  1. The system sends a notification that an error occurred.
  2. The operator continues from step 1.

3. **Title: View schedule evaluation metrics**

- **Brief description:** The operator views the metrics evaluating the schedule produced by scrolling down in the page.
- **Primary actor:** The electricity distribution operator
- **Pre-conditions:** The scheduling is successfully made.
- **Post-conditions:** The evaluation metrics are shown on the user interface.
- **Basic flow:**
  1. The operator scrolls down on the page.
  2. The system displays the metrics evaluating the quality of the schedule.
- **Alternative flows:** None

4. **Title: Download the schedule**

- **Brief description:** The operator downloads the schedule resulting from the algorithm being run on the dataset.
- **Primary actor:** The electricity distribution operator
- **Pre-conditions:** The scheduling is successfully made.
- **Post-conditions:** The schedule is downloaded.
- **Basic flow:**
  1. The user clicks the Download button.
  2. The system downloads the schedule.

4.2.2. Requirements

Functional requirements

- **Upload of the input data**

  The user should be able to upload all the necessary information about a residential building that will be used in the scheduling process. They will be able to give a name for the data to be able to distinguish between buildings.

- **Start scheduling for a selected dataset**

  From the application, it should be possible to start scheduling over a day for a given building. The user should be able to select the dataset for which the schedule will be made, upload the target curve which will be approximated, and also upload the configuration parameters that he wants to run the algorithm with.

- **Download the resulting schedule**

  The application should offer the possibility to save the resulting schedule for each device within each apartment over a day. The format of the saved file should be a generic format, which allows processing it with third-party applications.

- **View how well the schedule fulfills the objectives**

  The application operator will have to evaluate the quality of the result by examining the distance of the total over-the-day consumption from the target curve, and by analyzing how the schedule complies with the comfort intervals. For this, the application will have to display the values of the fitness functions, a chart with the total consumption over a day compared to the target consumption, and charts for evaluating the comfort objective.

Non-functional requirements

- **Usability**

  The system should be easy to use and will overtake all the unnecessary actions from the users. The user will need to introduce only the strictly necessary information intuitively. From the operator's point of view, the information shown will be aggregated, and its dimension reduced for increased readability. This requirement also includes the functionality of showing warning messages and guiding the user to provide valid information to the system.

4.3. Meta-heuristic based method for the optimal scheduling of devices

To be able to implement the optimization algorithm a definition must be provided of the problem. A residential building B will have AN number of apartments (Equation 4.18).

$$B = \{ap_i \mid i \in [1, AN]\} \tag{4.18}$$

Each apartment ap is identified by an id and has a set of devices D_a (Equation 4.19). The apartment a contains DN_a number of devices (Equation 4.20).

$$ap_a = (id, D_a) \tag{4.19}$$

$$D_a = \{d_{i_a} \mid i \in [1, DN_a]\} \tag{4.20}$$

Each device i from apartment a has a set of basic information to describe it (BI_{i_a}), a comfort vector (CV_{i_a}), and a current consumption pattern vector (CPV_{i_a}) (Equation 4.21).

$$d_{i_a} = (BI_{i_a},\ CV_{i_a},\ CPV_{i_a}) \tag{4.21}$$

The set of information that describes devices has as elements the ID of the device (id_{d_a}), its name ($name_{d_a}$), its consumption rating expressed in kWh ($consumption_{d_a}$), a marker whether it is deferrable ($isDeferrable_{d_a}$), and the minimum number of hours it must be turned on during a day ($minUsageHours_{d_a}$) (Equation 4.22).

$$BI_{d_a} = (id_{d_a},\ name_{d_a},\ consumption_{d_a},\ isDeferrable_{d_a},\ minUsageHours_{d_a}) \tag{4.22}$$

The comfort vector of device d from apartment a (CV_{d_a}) is a vector of 24 elements (Equation 4.23), a value for each hour of the day. And it marks how uncomfortable tenants are to use a certain device at a given hour. The variable $comfortValue_h$ can take positive integer values (Equation 4.24): 0 means the user is comfortable using the device at a certain hour, and a greater number will mark the significance of the discomfort produced.

$$CV_{d_a} = \{comfortValue_h \mid h \in \{0,\ 1,\ ...,\ 23\}\} \tag{4.23}$$

$$comfortValue_h \in \mathbb{N} \tag{4.24}$$

In Figure 4.5, multiple comfort vectors CV are shown stacked on each other for each devices of three apartments. The first value from the id touple is the apartment identifier, the second is the device indetifier.

The current consumption pattern vector CPV shows in which hours is the device d used by the tenant without scheduling (Equation 4.25). It is a boolean vector containing 1 if the appliance is turned ON at a certain hour and 0 otherwise (Equation 4.26).

$$CPV_{d_a} = \{isUsed_h \mid h \in \{0,\ 1,\ ...,\ 23\}\} \tag{4.25}$$

$$isUsed_h = \begin{cases} 0, & not\ used\ at\ hour\ h \\ 1, & used\ at\ hour\ h \end{cases} \tag{4.26}$$

id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
(1,1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,2)	0	0	2	1	0	0	0	1	2	2	2	2	2	2	2	2	2	2	2	1	0	0	0	0
(1,3)	2	2	2	2	2	2	0	1	0	2	2	2	2	2	2	2	2	2	0	1	1	0	0	1
(2,1)	2	2	2	2	2	2	2	2	2	2	1	0	1	1	0	0	1	1	0	0	0	1	2	2
(2,2)	2	2	2	2	2	2	0	0	0	2	2	2	2	2	2	2	2	2	0	2	2	0	0	1
(3,1)	2	2	2	2	2	2	2	2	2	2	0	0	0	1	0	1	0	0	0	1	2	2	2	2
(3,2)	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
(3,3)	2	2	2	2	2	2	2	2	1	0	0	0	1	1	2	0	0	0	0	1	2	2	2	2

Figure 4.5: Comfort matrix layout

id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
(1,1)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(1,2)	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
(1,3)	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
(2,1)	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	1	0	0	0
(2,2)	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
(3,1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,2)	1	1	0	1	1	0	0	1	1	1	0	0	0	0	0	1	0	1	0	1	1	1	0	1
(3,3)	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

Figure 4.6: Usage matrix layout

The schedule for a device with id d from the apartment a is a boolean vector called S_{a_d} of 24 elements (Equation 4.27), similar to the current consumption vector. The value 0 in $Dev_{a_d}[h]$ means the device is turned off at hour h, conversely, the value 1 means it is turned on (Equation 4.26).

$$S_{d_a} = \{isUsed_h \mid h \in \{0,\ 1,\ ...,\ 23\}\} \tag{4.27}$$

In Figure 4.6, is shown how the schedule vectors S of devices from three apartments would look like stacked on each other.

Each type of device brings with it a constraint for the scheduler. Non-deferrable devices must maintain their initial consumption pattern throughout the scheduling procedure (Equation 4.28). The number of hours interruptable devices are turned on over a day must be greater than or equal to the minimum usage hours specified for the device (Equation 4.29).

$$S_{d_a}[h] = CPV_{d_a}[h], \forall h \in \{0,\ 1,\ ...,\ 23\} \tag{4.28}$$

$$\sum_{h=0}^{23} S_{d_a}[h] \ \geq \ minUsageHours_{d_a} \tag{4.29}$$

The target consumption curve TC (Equation 4.30) that will be approximated is a vector of 24 elements and has as values the overall consumption of the building expressed in kWh (Equation 4.31) divided for each hour of the day.

$$TC = \{consumption_h \mid h \in \{0,\ 1,\ ...,\ 23\}\} \tag{4.30}$$

$$consumption_h \in \mathbb{R}_{\geq 0} \tag{4.31}$$

The overall consumption of the entire building for the day (Equation 4.32) is represented by the vector SC that has 24 values, a consumption value for each hour (Equation 4.31). It is obtained by multiplying the rated power consumption of each device with its schedule values and summing these amounts up for each hour (Equation 4.33),

for a visualization see Figure 4.7.

$$SC = \{scheduled\_consumption_h \mid h \in \{0,\ 1,\ ...,\ 23\}\} \tag{4.32}$$

$$scheduled\_consumption_h = \sum_{a=0}^{AN} \sum_{d=0}^{DN_a} consumption_{d_a} * S_{d_a}[h] \tag{4.33}$$

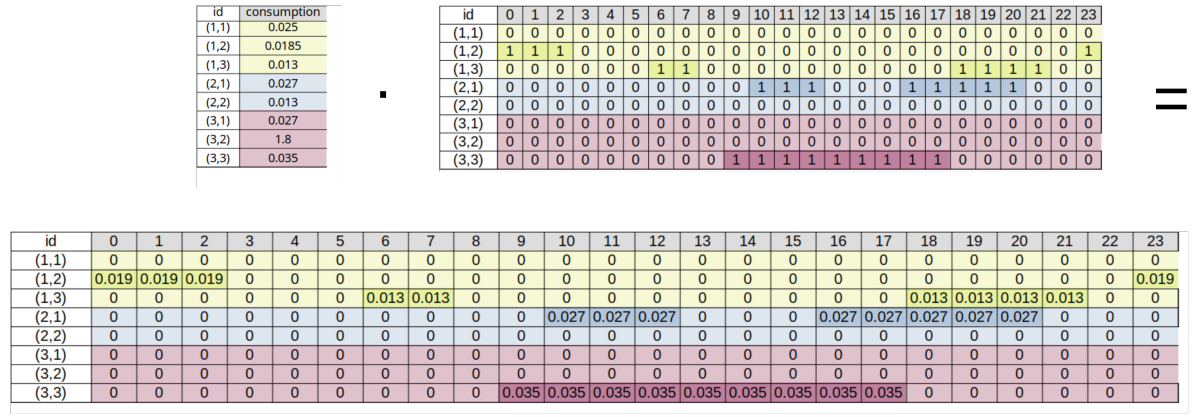


Figure 4.7: Computation of the scheduled consumption

The overall comfort penalization (OCP) for each device for the day will be given by the element-wise multiplication of the values from its schedule definition and comfort definition. For the operation see Equation 4.34, and for a visual representation see Figure 4.8

$$OCP_{d_a} = \sum_{h=0}^{23} S_{d_a}[h] * CV_{d_a}[h] \tag{4.34}$$

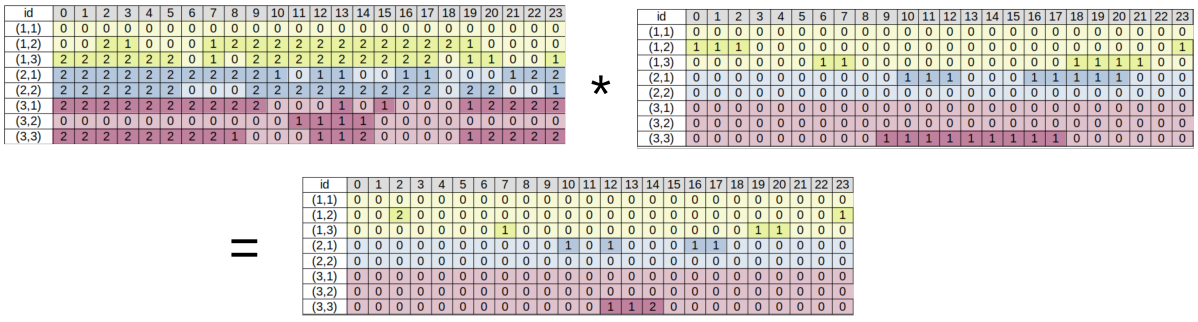


Figure 4.8: Applying the comfort penalty to the schedule

The objective function that needs to be minimized in the lower-level optimization is f_{obj_lower} and is defined in Equation 4.36. w represents a trade-off factor and measures the extent to which the user is willing to preserve its comfort to the detriment of the approximation objective ($w \in [0, 1]$). NF_{Approx} and $NF_{Comfort}$ are the normalized values of the approximation (Equation 4.37) and comfort objectives (Equation 4.42) respectively. *baseline_approximation* and *baseline_consumption* are reference values for the two objectives and represent the maximum fitness each can take. Normalization brings

the value from the range $[0, baseline\_upper]$ to $[0, 100]$. Equation 4.35 shows the formula for mapping an interval $[baseline\_lower, baseline\_upper]$ to $[scaled\_lower, scaled\_upper]$. Since the scaled lower bound is 0 in our case, the formula is simplified.

$$scaled = scaled\_lower + \frac{value - baseline\_lower}{baseline\_upper - baseline\_lower} * (scaled\_upper - scaled\_lower) \tag{4.35}$$

The normalized approximation function is a linear combination between the normalized Euclidean distance between the target and solution consumption curves, and the normalized Pearson correlation coefficient of the two (see Equation 4.38 and Equation 4.40).

The definition of the Euclidean distance is provided by Equation 4.39

The definition of the Pearson correlation coefficient is given in Equation 4.41, where $\overline{SC}, \overline{TC}$ are the mean values of the scheduled and target consumptions respectively. The Pearson correlation coefficient is used to measure the relationship between two data points. It results in a value from the interval $[-1, 1]$ in which a value close to -1 indicates an inverse linear relationship between the points, a value close to 1 shows a linear relationship between them, and a value close to 0 means no relationship between the sets.

$$f_{obj\_lower} = (1 - w) * NF_{Approx} + w * NF_{Comfort} \tag{4.36}$$

$$NF_{Approx} = w_{euclidean} * NF_{Euclidean} + (1 - w_{euclidean}) * NF_{Pearson} \tag{4.37}$$

$$NF_{Euclidean} = \frac{F_{Euclidean}}{baseline\_consumption} * 100 \tag{4.38}$$

$$F_{Euclidean} = \sum_{h=0}^{23} |SC[h] - TC[h]| \tag{4.39}$$

$$NF_{Pearson} = \frac{F_{Pearson} + 1}{2} * 100 \tag{4.40}$$

$$F_{Pearson} = \frac{\sum_{h=0}^{23}(SC[h] - \overline{SC})(TC[h] - \overline{TC})}{\sqrt{\sum_{h=0}^{23}(SC[h] - \overline{SC})^2}\sqrt{\sum_{h=0}^{23}(TC[h] - \overline{TC})^2}} \tag{4.41}$$

$$NF_{Comfort} = \frac{F_{Comfort}}{baseline\_comfort} * 100 \tag{4.42}$$

$$F_{Comfort} = \sum_{h=0}^{23} \sum_{a=0}^{AN} \sum_{d=0}^{DN_a} P^{OCP_{d_a}[h]} - 1 \tag{4.43}$$

The objective function that measures the discomfort generated by the schedule is defined in $F_{Comfort}$. It is the sum of the values given by an exponential function for each time devices are scheduled outside the comfort intervals as shown in Equation 4.43. P is a configuration parameter representing the base of the exponential function. The greater this value is the more penalty will be given for even a slight violation of the comfort interval.

At the upper level, the fitness function is described by f_{obj_up} as it can be seen in Equation 4.44. It is the weighted average of the fitness given by the minimum subset objective, and the over-sampling fitness, which are closely related, and the fitness values from the lower level. The weights of the objectives have to sum up to 1, as shown in Equation 4.45.

$$f_{obj\_up} = w_{subs} * F_{subs} + w_{over\_sampling} * F_{over\_sampling} + w_{approx_{up}} * NF_{Approx} + w_{comfort_{up}} * NF_{Comfort} \tag{4.44}$$

$$w_{subs} + w_{approx_{up}} + w_{comfort_{up}} + w_{over\_sampling} = 1 \tag{4.45}$$

The subset fitness is the percentage of apartments included in the optimization and can be expressed as shown in Equation 4.46, where AIN is the number of apartments included (Equation 4.47) in the subset defined by Equation 4.48. $isIncluded_a$ tells whether apartment a is in the subset or not.

$$F_{subs} = \frac{AIN}{AN} * 100 \tag{4.46}$$

$$AIN = \sum_{a=0}^{AN} Subs[a] \tag{4.47}$$

$$Subs = \{isIncluded_a | a \in \{0,\ 1,\ ...,\ AN\}\} \tag{4.48}$$

$$isIncluded_a = \begin{cases} 0, & apartment\ a\ not\ included \\ 1, & apartment\ a\ included \end{cases} \tag{4.49}$$

The over-sampling fitness gives the percentage of hours in a day when the target consumption is lower than the consumption of devices not selected to be part of the demand response program. It is defined in Equation 4.50. Where HOC is defined in Equation 4.51, using $isOverConsumption$ defined in Equation 4.52. CNI is the consumption of apartments not included and can be computed by applying Equation 4.32 on the set of apartments not included.

$$F_{over\_sampling} = \frac{HOC}{24} * 100 \tag{4.50}$$

$$HOC = \sum_{h=0}^{23} isOverConsumption[h] \tag{4.51}$$

$$isOverConsumption_h = \begin{cases} 0, & TC[h] \geq CNI[h] \\ 1, & TC[h] < CNI[h] \end{cases} \tag{4.52}$$

4.4. Optimization process

The algorithm performs the task divided into multiple stages, as shown in Figure 4.9. The input data that is passed must be parsed and then transformed into the representation expected by the algorithm. The problem is divided into two levels: an upper-level optimization problem that will search for the minimum subset of apartments to be included in the demand response program and a lower-level optimization problem to find a schedule that best approximates the target curve. At each fitness evaluation step from the upper level, a new lower-level optimization process is initiated with the subset of apartments selected. After the HHO algorithm makes a schedule, it and its fitness is passed to the upper level, where the genetic algorithm performs its specific steps based on the received fitness. After which the solution is exported into a specific format.

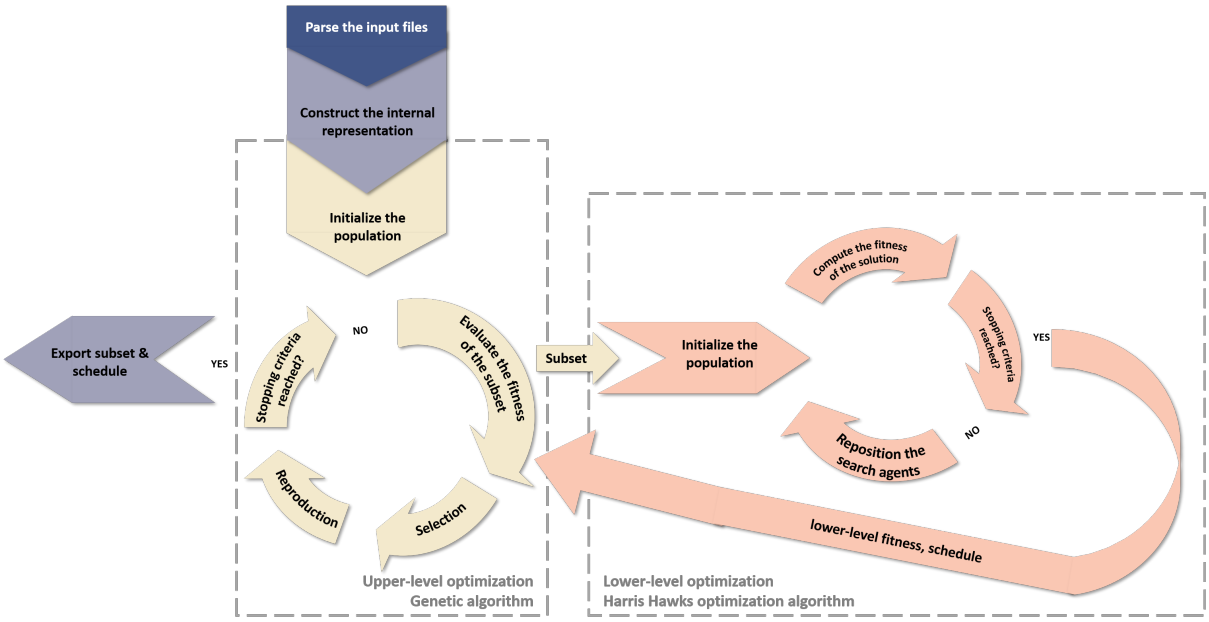


Figure 4.9: Stages of the scheduling algorithm

Chapter 5. Detailed Design and Implementation

This chapter will provide insight into the architecture and the implementation details of the application that will be used by the operator from the electricity distributor side to create a schedule of electricity-consuming devices from a residential building.

5.1. Detailed System Design

This section presents the architecture of the application together with the technologies that were used. It separates the design of the front-end, back-end applications, and the database server.

5.1.1. System Architecture

The system is divided into three distinct components: the front-end application, the back-end application, and the database server. They are contained within a Docker container. Figure 5.1 presents the visualization of this decomposition. The client-server architectural pattern was used to lay the ground for the whole application. It suits our needs since it offers high scalability, and low coupling between components all with small management overhead.

The operator has direct access to the front-end server. This is running on an NGINX web server, which offers high-performance, and out-of-the-box security features. It acts as a reverse proxy by accepting requests from the user that are forwarded to the back-end server, while the result of the request is displayed on the client side. The front end offers functionalities such as uploading a new dataset, performing operations on a dataset such as scheduling, or baseline establishment, and also provides functionalities to graphically display evaluation metrics of the resulting schedule.

The front-end server communicates with the back-end server by utilizing REST API endpoints. REST API defines a set of communication rules between the client and the server for transmitting data. It operates over the HTTP protocol, and in our implementation it uses JSON format to encapsulate data.

The back-end server runs on a lightweight Django application server and receives requests from the front-end. Django is a Python-based web development framework that offers request handling and routing features, protection from various attacks, and a comprehensive toolkit, that boosts development speed. The back-end server communicates directly with the front-end and is also responsible for persisting data into the database. The application uses the scheduling module as a library, only accessing its exposed functionalities, thus achieving low coupling between them and high cohesion.

To persist the dataset Postgres relational database system was used. The community behind it ensures the competitiveness of the database system, by making it ACID-compliant, offering data integrity and well-desired performance.

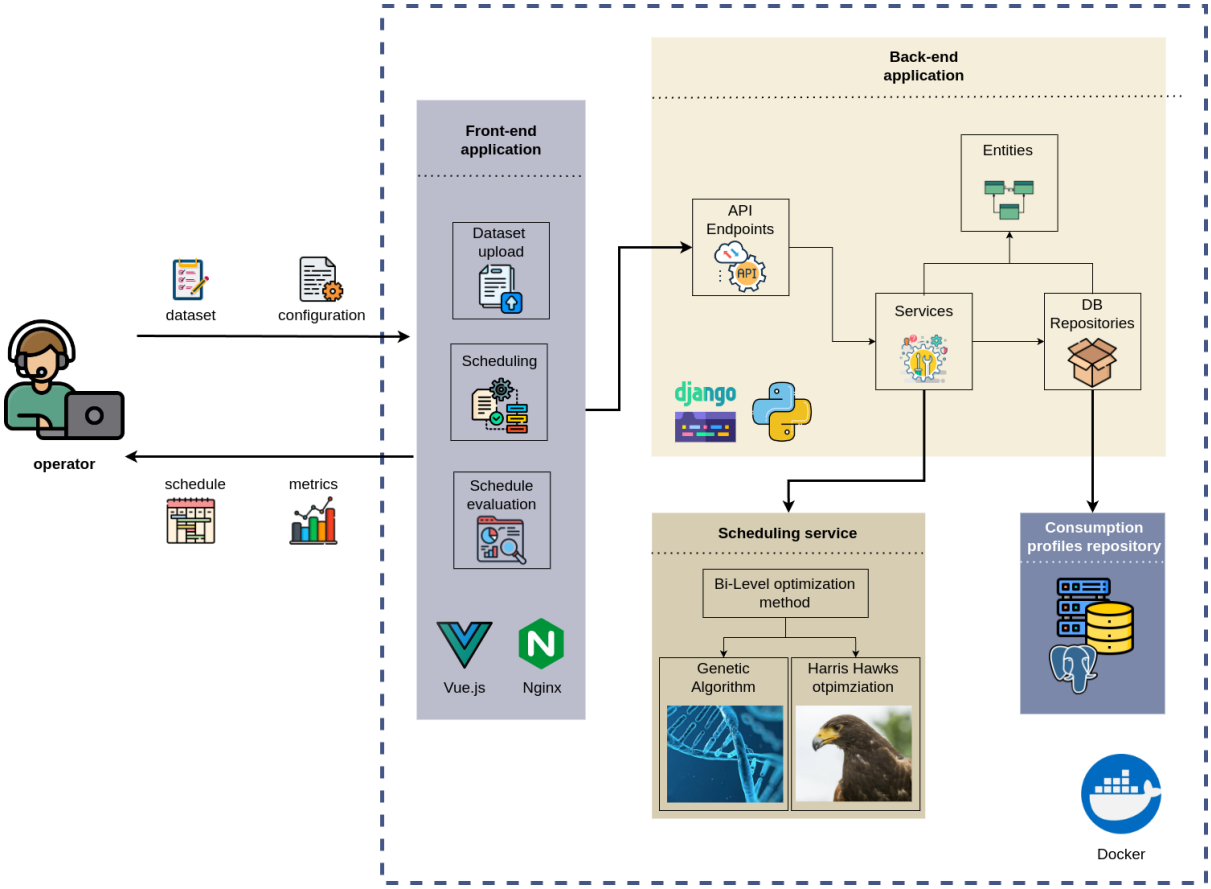


Figure 5.1: System architecture diagram

The mentioned components are enclosed in separate Docker containers which are connected together by a network. The deployment of all three containers is orchestrated using Docker Compose. The details of this multi-container setup are presented in Section 5.4. It was chosen for virtualization because portability is promoted which inherently ensures cross-platform consistency, it is also worth mentioning the ease of deployment in the cloud of this configuration.

Having this robust architecture the solution is future-proof. By being modular and decoupled it supports change of various components. It can also be scaled depending on the load under which it will be put. The API endpoints were developed having in mind the potential need of electricity distributors to integrate the application into their own systems.

5.1.2. Front-end Application Design

As it was mentioned, the front-end application runs on NGINX. It is implemented using Vue.js; a Javascript-based framework for developing user interfaces. It provides a toolkit for all levels of front-end developers, from beginners to novices, and helps them develop applications significantly faster by reusing code and lifting the burden of boiler-plate, and repetitive Javascript. As mentioned, it leverages the power of component-based development which ensures visual consistency over the website, besides making available already written code.

The user interface consists of a landing page and a dashboard from which the

operator can submit tasks. The application is installed on-premise and has a single type of user, who has been granted access through a local network.

The operations the dashboard has to support are described in this paragraph. The set of tasks the user can submit is divided into two: dataset upload and operations on a dataset. On one hand, when uploading a dataset, the UI should allow the user to give an identifier to it, and attach three text files. On the other hand, the user should be able to submit operations to the back-end server. These operations are dataset-specific, thus it has to select the dataset on which operations will be carried out, and the operation type. Since the operations are volatile, they will be queried from the server. The operations will be made based on two text files which will be uploaded. However, depending on the type of operation, they might not all be mandatory. The results of the submitted operations have to be displayed on the same page. It is also required to follow the status of a task by displaying notification messages.

From these requirements, we identify several attributes the design of the application has to ensure. It must be adaptive, as operations can change, at the same time it should be able to display a wide range of metrics, in various forms, such as charts, and tables. It also has to be divided into components for reusability, since operations such as file upload are repeated. The interface should also be responsive, intuitive to use and should handle alternate cases well.

5.1.3. Back-end Application Design

The back-end application receives requests through an API, performs computations, eventually communicates with the database, and returns a response. From this high-level description, three main layers can be identified: the presentation layer, the business layer, and the persistence layer. Thus a layered architecture was used to build the details of the server, as can be seen in Figure 5.2.

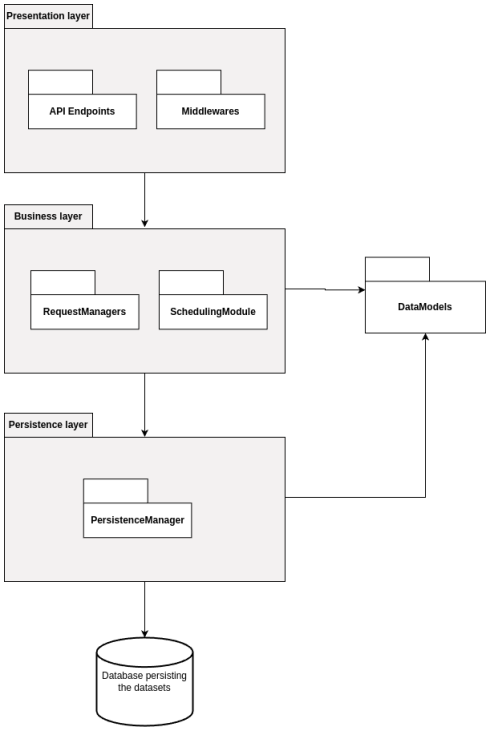


Figure 5.2: Back-end package diagram

The presentation layer is the outermost layer of the application, and it has an interface exposed to external systems, such as the front-end server to communicate with. The Controllers package contains classes that expose API endpoints of the server. All requests flow through this package. Besides it, there is a package called Middlewares which contains scripts that are run after receiving each request, but before it arrives at the controller. These scripts have a significant role in validating requests and ensuring all data and files are complying with the expected format.

The Business layer handles requests according to the business logic each operation has. Managers ensure that all the needed data is received for the selected operation, and that the state of the models will be kept in a consistent state by using transactions. The classes from this package communicate with the exposed functionalities of the scheduling module. Managers are also responsible for creating a new dataset and retreiving it on demand.

The Persistence layer makes the mapping between models and database tables. It is responsible for translating a python class into a table from a relational database, and to maintain the relationship between the entities within the table-space and vice versa. This complex management logic is taken over by the Django library.

5.1.4. Scheduling module design

The scheduling module is responsible for finding a solution to the multi-objective optimization problem. Namely to find the best usage schedule of appliances from a residential building such that a target curve is well approximated, and the impact on the comfort of tenants is minimized.

As shown in Figure 5.3, the solution receives as input the description of the devices from the apartment, the time intervals they are comfortable using a certain energy consumer, and the energy consumption expressed in kWh of the appliances provided by each apartment owner through the electricity distribution company. As output, each tenant will receive a schedule for their devices which is distributed by the company operating the application. The target consumption of the entire building will be provided by the system operator, namely the energy distribution company, which will receive, after the optimization process, an evaluation of the schedule.

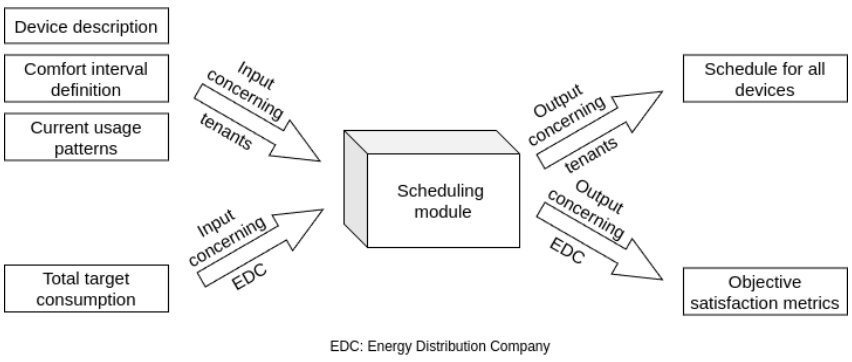


Figure 5.3: Black box diagram of the required solution

Package diagram

The package diagram in Figure 5.4 provides an overview of how classes are arranged within the module. The package *module_operations* contains classes that expose functionalities such as scheduling, consumption curve comparison, baseline value computation, etc. The *data_model* package encapsulates packages that contain the model classes, DTOs, and mappers from dictionaries to entities. The *business_logic* package contains the logic necessary to run the optimization algorithm. The sub-package *schedule_providers* contains classes that represent entry points to the scheduling algorithm. The *optimization_problem* package contains the customizations of the meta-heuristic algorithms provided by the MealPy library. The package *operations* is a utility package inside *business_logic* that contains common operations on the data that are from the domain field. The utility package *utils* encapsulates sub-packages that contain supporting classes for different operations. The *convertors* package has as its most significant class the one which converts csv files into model objects. The *generators* package contains classes such as the one used to generate test data. The *file_operations* bundle holds classes that read the input files. As its name suggests, the *configuration* package is responsible for holding runtime configuration data, and the *chart_engine* package is used to generate charts.

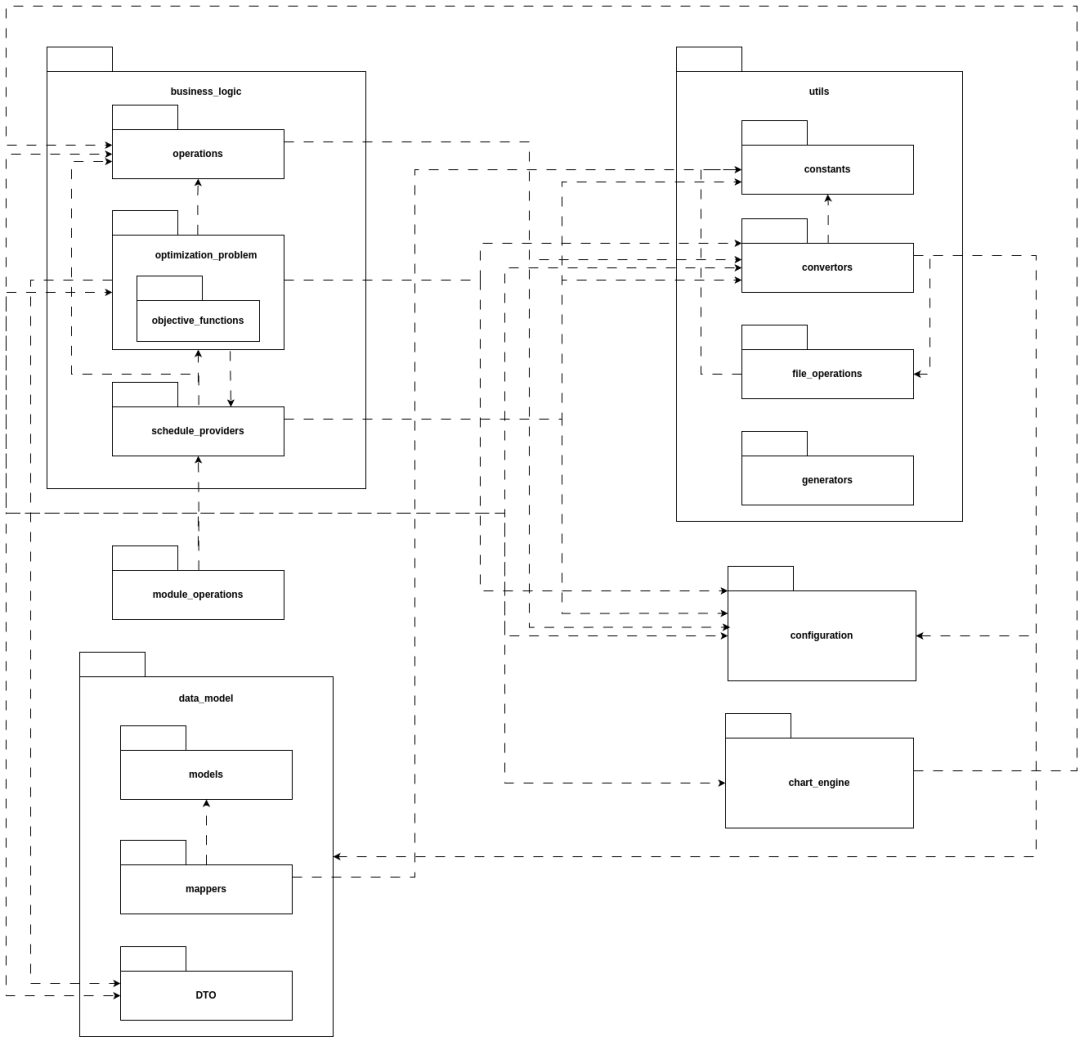


Figure 5.4: Package diagram of the scheduling module

5.1.5. Database Design

The system persists the data uploaded by the operator in a Postgres relational database. The schema tables represent a one-to-one mapping to the entities from the back-end server. In Figure 5.5 it is shown the entity relationship diagram. The table that contains the identifier of a building is called *scheduler_Dataset*, it holds as primary key its code. It is connected by one-to-many relationships to the table *scheduler_DeviceDetail*. The *scheduler_DeviceDetail* table holds information about each device; the apartment it belongs to, its name, consumption rating, type (which is 1 by default, meaning it is deferrable), and minimum usage hours. All columns are mandatory. Each device has exactly one entry in the *scheduler_ComfortDefinition*, and *scheduler_UsageDefinition* tables, modeled by a one-to-one relationship towards the *scheduler_DeviceDetail* table. The *scheduler_ComfortDefinition* and *scheduler_UsageDefinition* tables are almost identical by structure. The columns referring to hours in the *scheduler_UsageDefiniton* table are boolean values, while in the *scheduler_ComfortDefinition* table they are small integers.

The database is in Fifth Normal form (5NF), the highest level. It satisfies all requirements of preceding normal forms, such as each cell contains a single value, each non-key attribute depends on the primary key, all non-key attributes are independent of each other, each non-key attribute is dependent only on the candidate key, tables don't contain any multi-valued dependencies, and tables are decomposed in such a way to remove data redundancy and ensure integrity.

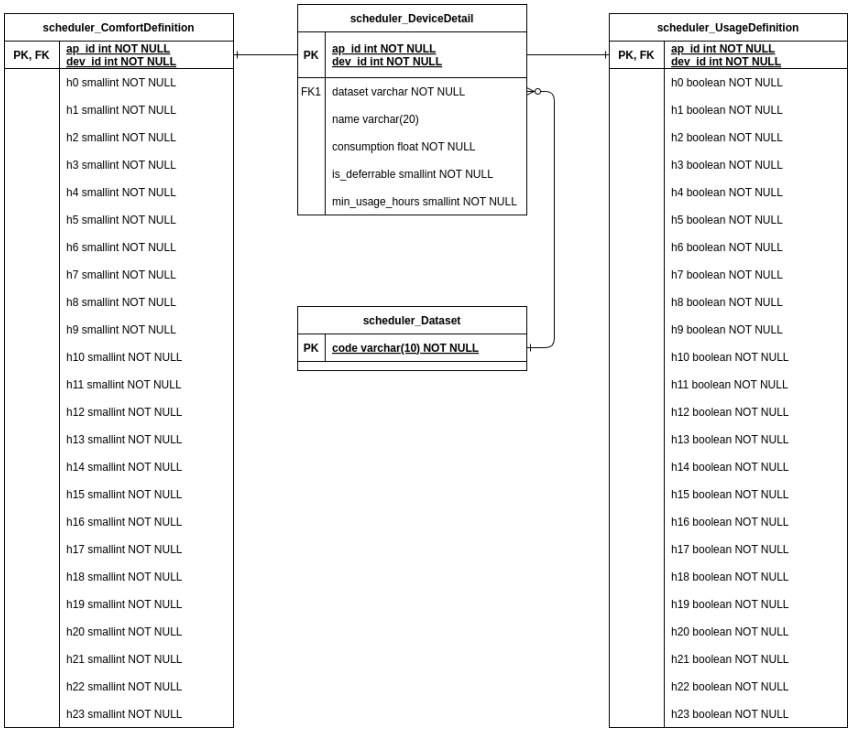


Figure 5.5: Entity relationship diagram

5.2. Detailed System Implementation

This section will describe the implementation details behind each component of the application. The mission-critical functionalities will be emphasized over the description.

5.2.1. Front-end Implementation

The front-end application leverages the power of components provided by Vue.js. It has defined two main pages, that are populated with reusable components. The `LandingPage` has a minimalist design, containing just a button that redirects the user to the main dashboard called `UploadPage`. The dashboard has three main functionalities: save a dataset, start an operation, and display an operation's results.

Both the file upload and operation submit functionalities contain fields for uploading files. These are created using a custom component called *EditRow* which has as input-type *UPLOAD*. When a file is attached, an event is generated using a call-back function, and the file is saved into a variable. In the case of uploading a file, a form is created to which the values of the fields are attached: dataset code, device detail file, comfort definition file, and device usage file. The form is submitted through a POST request to the server which responds with a status. In a similar way, in the case of the operation submit tab, the uploaded files are attached to a form, together with the selected operation code, and the selected dataset. The server provides the values of the two drop-downs by making an initial request when loading the page. The two requests are made in the *beforeCreate()* lifecycle hook, which is called before all. In contrast with the other tab, in this case, not all fields are required for certain operations, therefore the server will validate the existence of required data and will return an error message if not all the fields are complete. In the case of the file upload tab, this requirement is validated by the front-end as well. The result of an operation is provided in a special message by the server. This is required since different operations return different data, that has to be displayed differently. In this regard the following message structure is adopted, using the power of key-value pairs provided by JSON. A sample message can be seen in Listing A.1. When such a message is received, the application iterates through the result array, and depending on the type of each result: table, chart, or info pushes a new object with the respective data to the corresponding list, as seen in Listing 5.1. After that, using a reactive event, the application constructs a component from each object and displays it in the results section. The flow for downloading the schedule received as INFO type is slightly different, as a second request is made to the server to retrieve the bytes of the file, which is then attached to the DOM as a link. If the link is actioned, the attached file is downloaded. This second request can be seen in Listing 5.2.

```
for (var result of results) {
    var data = result.data
    if (result.type === "TABLE") {
        this.showTable = true
        this.tables.push({
            data: data.rows,
            header: data.header,
            title: data.title
        })
        continue
    }

```

```
    if (result.type === "CHART") {
        this.showChart = true

        this.charts.push({
            title: data.title,
            labels: data.label,
            type: data.type,
            datasets: data.valuesArray,
            yAxisTitle: data.yAxisTitle,
            xAxisTitle: data.xAxisTitle,
        })
        continue
    }

    if (result.type === "INFO") {
        if (result.data === "FILE") {
            this.makeRequestForFile()
        }
        continue
    }
}
```

Listing 5.1: Operation result processing

```
makeRequestForFile() {
    const getFileURL = "http://" + process.env.VUE_APP_SERVER_IP + ":"
   + process.env.VUE_APP_SERVER_PORT + "/scheduler/solution_file"
    axios.get(getFileURL)
        .then(result => {
            const url = window.URL.createObjectURL(new Blob([result.
   data]))
            const linkElement = document.createElement("a")
            linkElement.href = url
            linkElement.id = "solution_link_element"
            linkElement.setAttribute("download", "Schedule.csv")
            document.body.appendChild(linkElement)
            this.showDownload = true
            this.showScheduleTable = true
        })
        .catch(error => {
            this.showEmptyBackground = true
            this.$store.dispatch("handleInternalError", error)
        })
},
```

Listing 5.2: File download request

The system provides notifications for the user to inform him about the status of his submitted actions, or eventual malfunctions of the application. This feature enhances the user experience. To implement this a notification component was created which has three sub-components for success, warning, and error messages. The component is displayed over the contents of the page. It has set a timeout of 6 seconds. A list of notification components is maintained globally in a *vuex store*. This list enables the management of notifications from all over the user interface, thus by pressing the close button of the notification it will be removed from the list. Also, multiple notifications can be displayed in a stacked fashion. The text displayed in a notification is taken from a map of messages

based on the status key that was received from the server. Listing 5.3 shows how a notification can be created from every component of the application.

```
this.$store.dispatch("handleStatus", response.data.status)
```

Listing 5.3: Create notification

As could be seen above, requests are made to the server by using the Axios library [27]. Axios is a JavaScript library for making HTTP calls. It is widely used for making asynchronous requests due to its ease of use. As a response, it returns a JSON message, that can be easily processed in JavaScript.

To provide a uniform design across all pages Tailwind [28] was used. It is a CSS framework that provides already-made components, and styling definitions that can be used easily by only mentioning in the class attribute the styling that should be used.

For creating the charts the Chart.js [29] library was used. It provides a toolkit to create charts easily. It was used to implement the line charts, bar charts, and pie charts as well.

5.2.2. Back-end Implementation

Requests from the front-end application are received by the back-end application. Request handling and the API endpoints are provided by the Django framework for Python. To be able to identify the requestor, and prevent CORS attacks the server accepts requests from applications running on the mentioned IP addresses in Listing 5.4.

```
CORS_ALLOWED_ORIGINS = [
    "http://localhost:8080",
    "http://localhost:8081",
    "http://localhost",
    "http://172.19.0.30",
]
```

Listing 5.4: Allowed origins

All requests are intercepted by a middleware which checks the validity of the files that can be sent with the requests. The middleware is called `FileSecurityChecker`. The dataset code is validated using a regex to contain only letters and numbers of a maximum length of 10 characters. The files uploaded are validated using a library called *csvvalidator*. This library checks that the headers of the columns are the ones expected, and the values in each row are of a given type, or even have exact values from an array. The code for validating each file can be seen in Listing 5.5. The values of the column headers and the cell value definitions are provided in the arguments parameter which is an object containing a list of the header values, and a function that is used to check the validity of each cell's value. By these security measures, it is ensured that attacks such as SQL injection, command injections, file extension attacks, and so on are prevented.

```
def validate(file_csv, validation_builder, arguments):
    if file_csv is None:
        return True
    try:
        file_decoded = file_csv.read().decode('utf-8').splitlines()
        isValid = is_valid_csv_file(validation_builder, file_decoded,
   arguments)
        file_csv.seek(0)
        return isValid
```

```
    except Exception as e:
        return False


def is_valid_csv_file(validator_function, file, arguments=None):
    validator = validator_function(arguments)
    fileReader = csv.reader(file)
    problems = validator.validate(fileReader)
    if len(problems) > 0:
        write_problems(problems, sys.stdout)
        return False
    return True
```

Listing 5.5: File validation

After the requests pass the security check they are handled by a class called `RequestManager`. Here, each URL is mapped to a function that retrieves the parameters from the request, verifies the existence of mandatory fields, and forwards the request to the appropriate function in the `RequestManager`.

The request regarding saving a new dataset is transactional, ensuring that in case of a failure it will be rolled back without any side-effects. This transactional behavior and adherence to the ACID principles is ensured by an annotation exemplified in Listing 5.6. In case of an error with the database, or if the identifier is taken no data will be saved.

```
@transaction.atomic()
def saveDataset(code, device_details_csv, comfort_definitions_csv,
    usage_definitions_csv):
```

Listing 5.6: Persisting a new dataset is transactional

The same class is responsible for handling operation requests. Based on the operation code it selects a handler together with the mandatory files the request should contain. The way it selects a handler is written in Listing 5.7. The handlers are exposed by the *Schedulling module* described in Section 5.3. The available handlers are the following: *BaselineValues, InitialConsumption, CompareInitialAndTarget, FirstLevelOptimization, BiLevelOptimization, GreedyOptimization.*

```
def handleRequest(operation_code, dataset_code, target_curve,
    configuration):
    dataset = Dataset.objects.filter(code=dataset_code).first()
    if dataset is None:
        return {'status': 'UNKNOWN_DATASET'}

    appliances = SystemInitilizer.initialize_objects(list(dataset.
    devicedetail_set.all()), list(dataset.comfortdefinition_set.all()),
                                                        list(dataset.
    usagedefinition_set.all()))
    building_configuration = BuildingConfiguration.
    BuildingConfiguration(appliances)

    if operation_code not in request_handler.keys():
        return {'status': 'UNKNOWN_OPERATION'}

    if request_handler[operation_code]['isTargetCurveRequired'] and
    target_curve is None:
        return {'status': "INCOMPLETE"}

```

```
    if request_handler[operation_code]['isConfigurationRequired'] and
   configuration is None:
        return {'status': "INCOMPLETE"}
    else:
        if request_handler[operation_code]['isConfigurationRequired']:
            OptimizationConfiguration.configureFromFile(configuration)

    handler = request_handler[operation_code]['clazz']
    result = handler.do(building_configuration, target_curve)

    return result
```

Listing 5.7: Dispatching a request to a handler

The connection to the database is managed by the Django framework. In the `settings.py` file the connection details were defined as shown in Listing 5.8. The details are retrieved from environment variables for better portability and deployment.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': os.environ["licenta_db_name"],
        'USER': os.environ["licenta_db_user"],
        'PASSWORD': os.environ["licenta_db_pass"],
        'HOST': os.environ["licenta_db_ip"],
        'PORT': os.environ["licenta_db_port"],
    }
}
```

Listing 5.8: Connection details to the databse

5.3. Scheduling module implementation

This section will detail the implementation of the scheduling algorithm module of the application.

5.3.1. Used libraries

For the development of this module, the following libraries were used:

1. **MealPy** [30]
   The solution was heavily based on this library. It offers the implementation of cutting-edge meta-heuristic optimization algorithms. The library encapsulates an implementation that can be used out of the box, by just mentioning the number and types of parameters that should be optimized and an objective function. It also offers the possibility to customize every detail of the process for developers with complex tasks. It also offers an out-of-box solution for evaluating the optimization process by generating diagrams that show the evolution of fitness, diversity, and balance between exploration and exploitation.

2. **Numpy**
   Numpy is a well-known library offering highly efficient implementation of vector operations. It is widely used in the community of Artificial Intelligence developers because of its performance. This solution uses Numpy to make operations on matrices that contain information about devices within a residential building.

3. **Matplotlib**
   The Matplotlib library was created to release the burden of creating low-level code to generate diagrams in Python. It offers a high-level toolkit for generating a wide range of diagrams with great flexibility. It was used in this implementation in line with its purpose: to create charts.

5.3.2. Input data parsing

Format of the input files

The data fed into the scheduling algorithm contains the consumption patterns of a residential building. From these consumption patterns, we can deduce information such as the number of electricity consumers from the building, the hourly consumption (kWh) of each of them, and the comfort level of the tenants using a device at a given moment.

The following files have to be provided in csv format, with the comma symbol being used for separation:

- *Basic device information:* this file provides basic information about each device from the residential building. It must have the following columns:
  - ap_id: it is the number of the apartment the devices belong to
  - dev_id: it is the number of the devices from the apartment
  - name: it is the name or the category of the device
  - consumption: it is the consumption level of the device expressed in kWh
  - is_deferrable: a boolean field (containing 0 or 1) that marks whether the consumption of a device can be modified, or it should be left as it is in the current device usage schedule
  - min_usage_hours: is the minimum number of hours an appliance should be turned on over a day

  It has the role of describing each device willing to participate in the demand response program. The information contained in this file is used to compute the consumption of each appliance during the day.

- *Current device usage:* refers to the current usage patterns of each device. For each device, it will contain at which hour it is used. Each cell will contain the value 0 if the device on that row is not used at the hour identified by the column, and 1 if it is being used. Specifically, the file must have the following columns:
  - ap_id: identifier of the apartment
  - dev_id: identifier of the device from the apartment
  - 0: represents the 0^{th} hour of the day, rows contain the value 1 if the corresponding device is the state on, or 0 if it is switched off at this hour

  - 1: represents the 1^{st} hour of the day, rows contain the value 1 if the corresponding device is the state on, or 0 if it is switched off at this hour
    ⋮
  - 23: represents the 23^{rd} hour of the day, rows contain the value 1 if the corresponding device is the state on, or 0 if it is switched off at this hour

  The information shown in this file will contribute to extracting information about the current consumption patterns. It will be used to compare the target curve and the solution curve to the consumption patterns of residents without the demand response strategy. It is also used to impose non-deferrable constraints on devices.

- *Usage comfort definition:* describes how comfortable the user is to turn on the device at a specific hour. The columns of the file are exactly the same as in the previous file: ap_id, dev_id, 0, 1, ..., 23. In each cell, it is specified the discomfort level produced by turning on that device at a specific hour (a device is a row, an hour is a column). For instance, if the user from apartment 3 is very comfortable using the device with ID 2 at hour 15 it will introduce a value close to 0 (or 0) in the row identified by (3,2) in column 15. Conversely, if it doesn't want to use that device at that hour, he will introduce a greater number, for example, 2 marking that the discomfort produced by using the device at that specific hour is significant. This way the discomfort level for each device, at every hour, can be manipulated individually, depending on the user. This file will be used to compute the comfort penalization, in case the algorithm schedules appliances when the user doesn't want to use them.

- *Target consumption levels:* is a specification of the total consumption of the building for each hour. The values are expressed in kWh and represent the total consumption of the devices involved in the demand response program for the hour on the column. The columns are 0, 1, 2, ..., 23. It will contain only one row.

Internal representation

Internally the mentioned three files are encapsulated into a single class named `BuildingConfiguration`. As fields, it has a dictionary of appliances, a usage matrix, a comfort matrix, a consumption vector, and a constraint matrix.

The dictionary contains all the appliances (of type Appliance). This data is stored in case there is needed another representation of the input, it structures the data from the input, without being used in the operations of the scheduling algorithm. The key is the id of the device. The id is in the form of a pair `(apartment number, device id)`.

The value is of type `Appliance` which carries information specific to a device, such as its `id, name, type, consumption (in kWh), is_deferrable, min_usage_hours`, `comfort_vector`, and a `schedule_vector`. This class was introduced to decouple the format of the input files and the internal representation, thus containing eventual volatility in the format of the provided input files.

The initial schedule and comfort for the whole building are stored in two different tables. The matrix `usage_matrix` will contain the initial schedule of all the devices, and the matrix `comfort_matrix` will contain the comfort information. These tables are obtained by appending the vectors of each appliance. Storing the comfort information in a table of the same size as the schedule matrix brings the benefit of being able to identify

devices scheduled outside comfort hours after just an element-wise multiplication of the two matrices.

In the implementation, the rows of the usage matrix are identified by their index, as their order won't be modified during the process. Furthermore, when constructing the matrices a sorted data set (by id) is used, so it is ensured that a device with a certain index in the usage matrix will have the same index in all the other data structures. The columns from 0 to 23 represent the hours of the day.

This matrix is built from the input data by appending the usage vector of each device.

The comfort matrix, on the other hand, will be used directly in the scheduling process for computing the comfort penalty. It follows the same layout as the usage matrix, and it is built the same way. The higher the value in a cell, the greater the penalty that will be applied in case the usage is scheduled at that hour of the day.

The consumption vector will contain the consumption rating of each device expressed in kWh in a column vector. This representation is used to reduce the computational costs when establishing the consumption of all the devices given a schedule. More on this will be discussed in the section describing the computation of the fitness value.

The constraint matrix holds information about the constraints that apply to a certain device. It contains a row for each device, ordered by their IDs. The values of its two columns are taken from the field `is_deferrable` and `min_usage_hours` of an appliance.

Below one can see how these tables are constructed from the Appliance structure in Listing 5.9.

```
def buildInternalRepresentation(self, appliances):

    for index, appliance in enumerate(appliances):
        self.appliances[appliance.id] = appliance

        appliance_usage = appliance.schedule_vector
        self.usage_matrix.append(appliance_usage)

        self.consumption_vector.append(appliance.consumption)

        appliance_comfort = appliance.comfort_vector
        self.comfort_matrix.append(appliance_comfort)

        is_deferrable = appliance.is_deferrable
        min_usage_hours = appliance.min_usage_hours
        self.constraint_matrix[index][0] = is_deferrable
        self.constraint_matrix[index][1] = min_usage_hours
```

Listing 5.9: Constructing the BuildingConfiguration DTO

The target curve is stored as an array of 24 elements. Each value is expressed in kWh, and represents the total consumption of the devices involved in the demand response program at a given hour.

5.3.3. Scheduling process

This section will describe all the steps taken to create a schedule.

Upper-level optimization

At this level, a genetic algorithm will select the minimum number of apartments from the residential building which has to modify their consumption patterns to achieve the objectives described.

1. **Initializing the population**

   Apartments are represented as a binary array from 0 to the number of apartments in the building. The value 0 in the array means the apartment is not selected to be part of the subset, 1 has the opposite meaning.

   Each search agent has such an array, that is initialized randomly. The number of search agents is a configuration parameter *subset_population_size*, as well as the number of epochs *subset_epoch_number*, and the specific operation probabilities: *crossover_probability*, *mutation_probability*. The code for initializing the model by calling the MealPy library is shown in Listing 5.10.

```
optimization_model = MyEliteMultiGA.MyEliteMultiGA(epoch_number=
    epoch_number, population_size=population_size,
    crossover_probability=crossover_probability,
    mutation_probability=mutation_probability)
```

Listing 5.10: Initializing the subset optimization problem

   Besides an initialized model, a custom problem definition has to be created too. The class `SubsetOpProblemDefinition` implements the fitness function and has to be initialized before starting the optimization, as seen in Listing 5.11. The custom class implements the `Problem` class from the library.

```
return SubsetOpProblemDefinition.SubsetOpProblemDefinition(
    building_configuration=building_configuration,
     target_curve=target_curve,
     bounds=mp.BoolVar(no_apartments),
     log_to=None,
     obj_weights=[subset_weight, approximation_weight,comfort_weight
    , over_sampling_weight])
```

Listing 5.11: Defining the subset optimization problem

   The optimization process is started by calling the *solve* function of the model.

2. **Genetic operations**

   The steps of the genetic algorithm were kept the same as the ones which are described in Section 4.1.5. For each randomly initialized search agent, its fitness is computed.

   In the selection step, based on the computed fitness, two agents are selected using the tournament selection strategy. Tournament selection works as follows in the MealPy library: from the list of candidate agents, a subset of them will be selected randomly in the first tournament, and after that, the two agents with the lowest fitness are returned.

The selected agents move to the crossover step. In this phase, the bit-strings of the two are taken and interchanged based on a probability given by the configuration parameter *subset_crossover_probability*. A uniform crossover strategy is applied, which means that each pair of bits can be interchanged with the same probability. The next generation of agents will have the solution of one of the two crossed-over agents.

In the next step, a mutation operator is applied to the resulting individual. By this, each bit of an agent will be interchanged with a random bitstring's elements with a probability given by the configuration parameter *subset_mutation_probability*.

These operations are executed only on 90% of the least-performing agents. The rest are considered good enough and do not take part in this process, only in the selection operation.

After each epoch, a list of agents is maintained in an order given by their fitness values obtained by evaluating the fitness function described below.

3. **Fitness evaluation**

   The fitness function gives a measure of how good a solution is. It is composed of four elements: subset fitness, approximation fitness, comfort fitness, and over-selection fitness.

   The subset fitness and the over-selection fitness are both evaluating the subset chosen by the algorithm. The subset fitness is the percentage of devices that were selected from the total set. The over-selection fitness is the percentage of hours of a day, in which the target consumption curve is under the consumption curve given by the devices not selected. Since devices from apartments that were not selected still contribute to the total consumption of the building according to their initial usage data, this consumption has to be subtracted from the target curve. This means that the lower-level optimization algorithm has to obtain a solution that fills the gap between the consumption given by devices that will not be scheduled, and the target curve by using the devices from the selected apartments. In case the gap is negative (i.e. with the current subset there is no way of achieving the target consumption curve) the subset algorithm has to be penalized. This penalization is quantified by the over-selection fitness.

   The approximation fitness and comfort fitness values result from the scheduling made by the lower-level algorithm using the apartments selected into the demand response program and the modified target curve.

   Each component of the fitness function has an associated weight. The total fitness is computed by applying a weighted average. The weights of the fitness values are given by the configuration parameters: *subset_weight, approximation_weight, comfort_weight, over_sampling_weight*.

   The objective function is defined in Listing 5.12.

```
1
2 def obj_func(self, solution):
3     appliances_included, non_selected_consumption =
    SubsetOperations.get_appliances_according_to_subset(solution,
    self.building_configuration)
4
```

```
5       new_target_curve = self.target_curve - np.array(
        non_selected_consumption)
6       new_target_curve = np.where(new_target_curve < 0, 0,
        new_target_curve)
7
8       no_hours_scheduled_too_much = 0
9       for hour in range(0, 24):
10          if self.target_curve[hour] < non_selected_consumption[hour
        ]:
11              no_hours_scheduled_too_much += 1
12
13      over_selected_fitness = (no_hours_scheduled_too_much * 100.0) /
        24
14
15      new_building_configuration = BuildingConfiguration(
        appliances_included)
16      schedule_solution, metrics = ScheduleMaker.do_scheduling_hourly
        (new_building_configuration, new_target_curve)
17
18      no_apartments_included = sum(solution)
19      subset_fitness = no_apartments_included *
        OptimizationConfiguration.fitness_reference_value / self.
        no_total_apartments
20
21      return [subset_fitness, metrics["normalized_distance"], metrics
        ["normalized_comfort"], over_selected_fitness]
```

Listing 5.12: Fitness function of upper-level optimization

4. **Stopping condition**

   The optimization will stop after a certain number of epochs is reached. This number is given by the *subset_epoch_number* configuration parameter.

Lower-level optimization

At this level, a schedule will be elaborated by the Harris Hawks optimization algorithm for the subset of apartments received from the level above. This schedule will try to approximate a given target curve while minimizing the impact on comfort. The exact steps are described below.

1. **Initializing the population**

   For implementing the main steps of the algorithm, the MealPy library [30] is used. The search space is populated with agents, each representing a configuration of all the optimization parameters. An optimization parameter is a cell from the schedule lying on a row identifying a certain device and on a column representing the hour. Thus an agent contains an optimization parameter for the usage (turned on or off) of each device in the building at each hour. The number of optimization parameters each agent has to modify, that is the dimension of the search space, is computed based on the shape of the `usage_matrix` from the `BuildingConfiguration`, and it is described by Equation 5.1.

$$parameter\_number = total\_device\_number * hour\_number \tag{5.1}$$

The number of search agents that will be instantiated is given by the configuration parameter `population_size`. This initialization step can be seen in the code snippet from Listing 5.13. The class `MyHHO` is a customization of the `OriginalHHO` provided by the MealPy [30] library.

```
optimization_model = MyHHO.MyHHO(problem.building_configuration,
    epoch=epoch_no, pop_size=population_size)
```

Listing 5.13: Initializing the scheduling optimization problem

As shown in Listing 5.14, for defining the optimization problem a separate class was created named `HHO` which extends the `Problem` class from the MealPy library. This class contains the definition of the objective functions and some key configuration parameters such as the lower and upper bounds of the optimization parameters contained by an agent.

```
return ScheduleOpPrDef.HHO(building_configuration=
    building_configuration,
        target_curve=[target_curve],
        should_normalize=adjust_fitness,
        bounds=FloatVar(lb=[0] * param_no, ub=[1] * param_no),
        log_to=None,
        obj_weights=[weight_consumption_obj, weight_comfort_obj])
```

Listing 5.14: Defining the schedule optimization problem

The optimization process is started by running the model on the problem definition discussed above. This step is shown in Listing 5.15.

```
solution = model.solve(problem, mode="single", n_workers=1, seed=
    random_seed)
```

Listing 5.15: Running the model on the optimization problem

At the start of the algorithm, the population of search agents is randomly spawned onto the search space. That means, that the values of the optimization parameters are random values with the seed given when creating the model. For each agent, an initial fitness is computed, and the best and worst agents are stored as part of the initialization. These steps are ensured by the library. Each agent aims to minimize the fitness of its solution during the multi-objective optimization.

2. **Repositioning the search agents**

   For the number of epochs established each search agent is moved on the space, and the best and worst performing agents are stored continuously. They will be returned after finishing all the epochs.

   The agents move in the search space depending on the phase the algorithm is in which is described in Section 4.1.6. In the exploration phase, they are initialized randomly (the parameters have random values) onto the search space, and try to find areas of the search space that improve the fitness. After computing the fitness function of each member, the best is chosen, and the rest will be assembled around it. In the exploitation phase, the search agents will start moving towards the best solution found yet, also incorporating randomness in their trajectory. The fitness is continuously evaluated during this process to be aware of a new possible best

solution. Search agents are moved by modifying the values of their optimization parameters.

As defined in Section 2, E0 is the initial energy, computed based on a random variable having values between $[0, 1]$. Given the complexity of the problem, and to force a more comprehensive exploration of the search space it was decided that this variable should take values between $[-2, 2]$. The impact of this adjustment will be analyzed in section 6.4.2. In Listing 5.16 is shown how the initial energy is computed with the extended interval.

```
E0 = 4 * random_var - 2
```

Listing 5.16: Computing the initial energy

3. **Constraint handling**

Each solution has to be amended to obey the constraints imposed by the user of the appliance. To ensure that these rules are followed the solution of each agent is modified after each iteration based on the constraints. The class `MyHHO` contains this functionality by overriding the function `amend_solution` from the `OriginalHHO` class provided by the MealPy library. This function is called every time a new solution is generated during the optimization process. This function applies the two constraints.

The constraint regarding the non-deferrable type of devices is applied by iterating through the rows of the constraint matrix, and if the value of the column associated with this constraint is 0 for a certain device, its schedule from the initial usage matrix will be copied to the solution schedule.

The minimum usage hours constraint is applied by iterating through the schedule row-by-row and if the number of hours the device was scheduled is less than the mentioned number of hours in the constraint matrix, then it will be randomly scheduled within its comfort interval. The process of applying this constraint is the following. First, the list of hours from the comfort interval that were not used in the schedule is obtained by a bit-wise AND operation between the normalized comfort interval (will contain 0 for all hours producing discomfort, and 1 for comfort hours) of the device, and the negated rounded usage row. From this list will be selected randomly so many numbers to fill the gap. The code for this operation is written in Listing 5.17.

```
def apply_min_usage_hour_constraint(solution, constraint_matrix, comfort_matrix):
    if OptimizationConfiguration.apply_min_usage_hour_constraint == 0:
        return solution

    for index, usage_row in enumerate(solution):
        usage = np.sum(usage_row)
        min_usage_hours = constraint_matrix[index][1]
        if usage >= min_usage_hours:
            continue

        rounded_usage_row = np.where(usage_row >= OptimizationConfiguration.discrete_transform_threshold, 1, 0)

```

```
13          comfort_row = comfort_matrix[index, :]
14          comfort_row_normalized = np.where(comfort_row > 0, 0, 1)
15
16          usage_row_neg = 1 - rounded_usage_row
17          available_hours = np.bitwise_and(comfort_row_normalized,
    usage_row_neg.astype(int))
18          hours_available_index = np.where(available_hours == 1)[0]
19
20          hours_to_choose = min_usage_hours - usage
21          chosen_hours = np.random.choice(hours_available_index,
    replace=False, size=int(hours_to_choose))
22          for hour in chosen_hours:
23              solution[index][hour] = 1
24      return solution
```

Listing 5.17: Minimum usage hours constraint

4. **Fitness evaluation**

   As stated by the problem, the optimization algorithm has to find the best schedule of devices participating in the demand response program, such that the total consumption of these devices approximates a target curve, while minimizing the impact on user comfort.

   Thus, the objective function contains two components: the deviation from the target curve, and the violation of the comfort requirements. The fitness of a solution will result from the linear combination of these two metrics, see Equation 4.36. The first objective is the approximation of the target curve. The second objective is maintaining the comfort level.

   The weights of the objectives are given by a configuration parameter called *comfort_tradeoff* (see Section 6.3). It is called comfort trade-off, because it quantifies the willingness of the residents to give up on their comfort, in favor of other benefits.

   The objective function receives a schedule as input. This schedule is provided by each search agent of the HHO algorithm. By default, search agents contain the parameters as a 1D vector. Therefore, the array has to be transformed into a matrix. Also, the optimization parameters from the schedule received are float real numbers between $[0, 1]$, thus they have to be rounded with the breaking point being set by a configuration parameter called *discrete_transform_threshold*, described in Section 6.3.

   To compute the first objective, the total hourly consumption of the building will be calculated based on the schedule.

   To have the hourly consumption of the solution, as a first step, the usage matrix has to be transformed into a consumption matrix. This is done by multiplying the consumption vector with the usage matrix. After this multiplication, each cell will contain the consumption of the devices turned on. See the operation visualized in Figure 4.7. There is only one step left to obtain the scheduled hourly consumption. The values from the consumption matrix have to be added column-wise.

   After these steps, the current solution and the target solution are in the same format: an array of 24 values containing the total consumption for each hour of the day. Thus, we can compare them.

Two functions were considered for evaluating the fitness with regard to the approximation objective: the Euclidean distance and the Pearson correlation coefficient (see Equation 4.37). The overall approximation fitness is the linear combination between the two. A thorough analysis of the usage of the two fitness functions will be made in Section 6.4.4.

In the following paragraphs, the computation of the fitness for the second objective is discussed. For achieving the comfort objective (i.e. scheduling appliances in a time interval when it is comfortable to use them) we are applying a penalty function on each appliance that is scheduled out of its comfort intervals. This objective is computed by element-wise multiplication of the schedule/usage matrix with the comfort matrix defined by the input data. The result of this matrix multiplication will contain the comfort levels associated with appliances scheduled outside the desired time intervals. This is illustrated by figure 4.8. The comfort penalty is computed according to Equation 4.43. The base of the exponential function is given by the configuration parameter *comfort_penalty_base* detailed in Section 6.3.

It is important to note, that the fitness given by the two objectives are on different scales. The Euclidean function from the approximation objectives yields values expressed in kWh on a scale whose upper limit can be on the magnitude of thousands. The Pearson correlation coefficient can take values only between $[-1, 1]$, and the function corresponding to the comfort objective has yet another range. Therefore, to be able to combine these values into a single fitness, we need to normalize them (bring them to the same magnitude). The process of normalizing these values is described in the following paragraphs.

Before starting the optimization algorithm two baseline values are established: a baseline value for the approximation objective evaluated with the Euclidean function, and one for the comfort objective. These baseline values establish a fixed value, close to the upper bound of possible values for each objective. They provide terms of comparison. Once we have the upper bounds we can define the range of fitness values as $[0, baseline\_value]$. By knowing this, we can scale the range to any other range. The configuration parameter *fitness_reference_value* gives the upper bound of the scaled range; i.e. $[0, fitness\_reference\_value]$.

The implemented objective function is shown in Listing 5.18.

```
1
2  def obj_func(self, solution):
3      reshaped_solution = MatrixOperations.reshape_flat_matrix(
       solution, initial_solution_shape)
4
5      rounded_solution = np.where(reshaped_solution >=
       OptimizationConfiguration.discrete_transform_threshold, 1, 0)
6
7      # compute fitness related to consumption
8      raw_consumption_fitness_pearson,
       raw_consumption_fitness_euclidean = ConsumptionObjective.
       compute_objective(
9          self.building_configuration.consumption_vector,
10         rounded_solution,
11         self.target_curve)
12
13     # compute fitness related to comfort
```

```
14      raw_comfort_fitness = ComfortObjective.compute_objective(self.
    building_configuration.comfort_matrix, rounded_solution)
15
16      aligned_consumption_fitness_pearson = 1 -
    raw_consumption_fitness_pearson
17
18      if self.should_normalize:
19          # normalize fitness values
20          consumption_fitness_euclidean =
    normalize_euclidean_consumption_fitness(
    raw_consumption_fitness_euclidean)
21          consumption_fitness_pearson =
    normalize_pearson_consumption_fitness(
    aligned_consumption_fitness_pearson)
22          comfort_fitness = normalize_comfort_fitness(
    raw_comfort_fitness)
23      else:
24          consumption_fitness_euclidean =
    raw_consumption_fitness_euclidean
25          consumption_fitness_pearson =
    aligned_consumption_fitness_pearson
26          comfort_fitness = raw_comfort_fitness
27
28      # compute consumption fitness as a linear combination of the
    result of the two evaluation functions
29      consumption_fitness = compute_consumption_fitness(
    consumption_fitness_euclidean, consumption_fitness_pearson)
30
31      return [consumption_fitness, comfort_fitness]
```

Listing 5.18: Minimum usage hours constraint

The baseline values are specific to each dataset. The baseline consumption value using the Euclidean distance as the similarity metric is obtained by running the scheduling algorithm with the *comfort_tradeoff* configuration parameter set to 1. Conversely, the baseline comfort value is obtained by setting the *comfort_tradeoff* configuration parameter to 0. As it was mentioned, these values are close to the upper bound of the interval fitness can take values from. Thus by focusing on only one objective, the close to maximum fitness is obtained for the other objective.

5. **Stopping condition**

   The stopping condition of the optimization is given by the total number of epochs the algorithm is let to run. This can be modified using the parameter *epoch_no*. An epoch is an iteration of the Harris Hawks algorithm in which it computes the fitness of all the agents from the population, establishes the agent with the best fitness, and performs specific operations to generate a new solution based on the current one.

6. **Improved scheduling algorithm**

   In the previous section, a base version of the algorithm was described in detail. In that version, the search agents receive the whole scheduling matrix, containing all the devices and all the hours. As it was shown in Equation 5.1 the total number of parameters is the result of multiplying the number of columns and rows of the schedule matrix. This results an extremely large search space the optimization algorithm had to explore, as shown in Listing 5.13 and Listing 5.15.

   In the improved version of the algorithm, the dimension of the space is reduced. The reduction is achieved by starting a separate optimization process for each hour of the day. In this way, the number of parameters is reduced by a factor of 24, thus the number of parameters for an hour i is $parameter\_number_i = total\_device\_number$. To be able to do this, without modifying the base algorithm the usage and comfort matrices are built by extracting the column corresponding to each hour one-by-one, and running the algorithm on it. After all hours are scheduled, the whole schedule for the day is reconstructed. The minimum usage hour constraint evaluation is moved after this step, as by scheduling one hour at a time, the information regarding the whole day cannot be obtained after each iteration. Listing 5.19 shows how the improved version is constructed. By creating separate processes for each hourly optimization we can leverage the fact that consumptions between separate hours are independent of each other, achieving a significant speed-up.

```
1  def do_scheduling_hourly(building_configuration, target_curve):
2
3      # number of processes
4      pool_size = OptimizationConfiguration.first_level_no_processes
5
6      # create a task for each hour of the day
7      for hour in range(0, 24):
8          seed = random.randrange(sys.maxsize)
9          payload = {
10             "building_configuration": copy.deepcopy(
   building_configuration),
11             "target_curve": target_curve,
12             "hour": hour,
13             "seed": seed,
14             "adjust_fitness": adjust_fitness
15         }
16         queue.put(payload)
17
18     # start execution on pool_size number of processes
19     with concurrent.futures.ProcessPoolExecutor(max_workers=
   pool_size) as executor:
20         futures = [executor.submit(solve_multiprocessing, queue)
   for _ in range(0, 24)]
21         for i, future in enumerate(concurrent.futures.as_completed(
   futures)):
22             # put together the results
23             subSolution, hour = future.result()
24             np.copyto(solution[:, hour], subSolution.squeeze())
25
26     solution = ConstraintOperations.apply_min_usage_hour_constraint
   (solution, building_configuration.constraint_matrix,
   building_configuration.comfort_matrix)
27
```

```
28      return solution
29
30
31  def solve_hourly(building_configuration_hourly, target_curve, hour,
        seed, adjust_fitness):
32
33      # extract only the columns for the current hours
34      building_configuration_hourly.usage_matrix = np.array(
35          [building_configuration_hourly.usage_matrix[:, hour]]).
        reshape(-1, 1)
36      building_configuration_hourly.comfort_matrix = np.array(
37          [building_configuration_hourly.comfort_matrix[:, hour]]).
        reshape(-1, 1)
38
39      # build and solve the reduced optimization problem
40      optimization_problem = build_problem(
        building_configuration_hourly, target_curve[hour],
        adjust_fitness)
41      best_solution, model = solve_problem(optimization_problem, seed
        )
42
43      reshaped_matrix = MatrixOperations.reshape_flat_matrix(
        best_solution.solution,
44                                                              (
        building_configuration_hourly.usage_matrix.shape[0], 1))
45      solution = np.where(reshaped_matrix >=
        OptimizationConfiguration.discrete_transform_threshold, 1, 0)
46
47      return solution
```

Listing 5.19: Minimum usage hours constraint

In section 6.4.5 a detailed comparison will be conducted between the two versions.

5.3.4. Output representation

The best solution found by the algorithm is exported in the form of a CSV file (comma-separated) with 24 columns. The structure of the output is the same as the usage matrix shown in Figure 4.6.

5.4. Deployment

All the components, namely the front-end application, the back-end server, and the database are deployed using docker containers. They are connected through a defined network with a subnet at *172.19.0.0/24*. The deployment diagram can be seen in Figure 5.6.

The front-end application is running on an NGINX server that is accessible on IP 172.19.0.30, port 80. A volume is created in the folder containing the source code, this ensures hot-reload in case any change occurs. The back-end application is running on a Python server, utilizing the Django framework. It accepts requests on IP 172.19.0.20, and port 8000. Similarly to the front-end server, hot-reload is ensured by a volume created in the folder having the source code. The database server is a Postgres server connected on IP 172.19.0.20 to the network on an exposed port 5433.

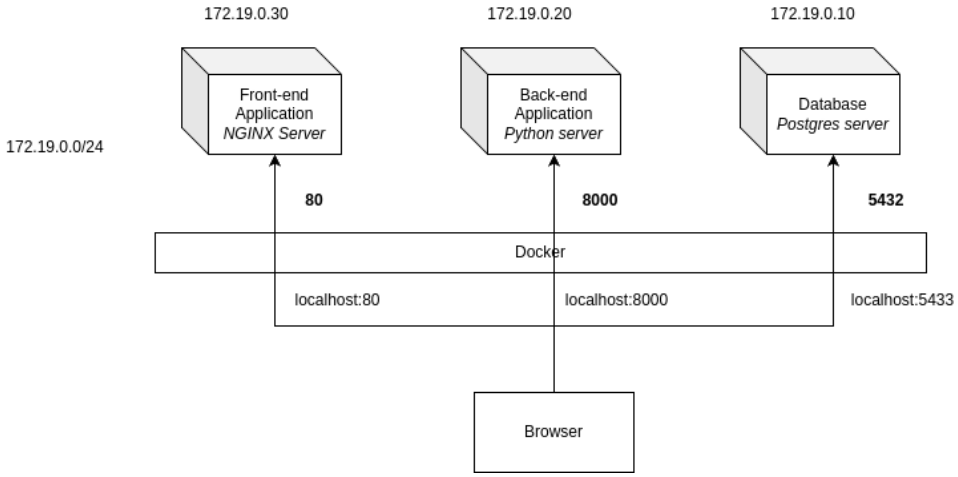


Figure 5.6: Deployment diagram

Chapter 6. Testing and Validation

In this chapter, a detailed analysis will be presented regarding the performance of the algorithm. Also, it will describe how certain design decisions impact the ability of the algorithm to converge towards a good solution.

6.1. Dataset

The algorithm was tested on two types of input data. Real-world data was collected using a survey involving 39 respondents, and a dataset generator was also made to increase the number of apartments while also mimicking multiple usage patterns.

The survey was filled out by students. It contains 39 apartments, each having defined 8 categories of devices: heating, cooling, laundry, waterheating, cooking, lighting, fridge, and microwave. This will result in a total of 312 devices to be scheduled, yielding 7800 parameters to be configured by the optimization algorithm. Respondents also marked for each device the hours when they were comfortable using them.

The consumption of these devices is computed based on the statistical data provided on the website of Silicon Valley Power, blog post named "US appliance energy usage chart".

The target consumption is established based on the current usage pattern of the devices. The current consumption curve is reduced by 30% and shifted with 2 hours to the right. A comparison between the initial and target consumption curves can be seen in Figure 6.1.

The script used for generating datasets, other than the one that resulted from the survey, is specified in Appendix A.2. In this script, two usage patterns were considered, a usage pattern resulting from individuals who are at home in the morning and get back from work at night. Thus having consumption between 8 am and 12 pm and between 9 pm and 12 am. The other category of residents consumes electricity a bit in the morning between 7 am and 8 am, and mostly in the evening between 6 pm and 11 pm.

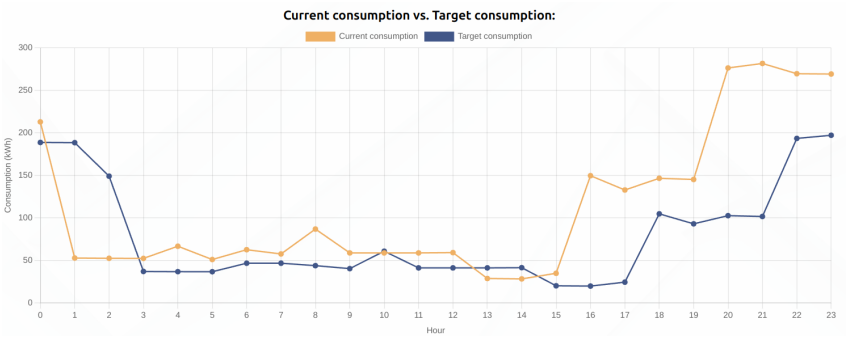


Figure 6.1: Current consumption and target consumption curve comparison

This script constructed the dataset using 4 categories of devices: lighting, fridge, laptop, and microwave. The fridge is set to be non-deferrable, and the comfort intervals of the microwave are set only for a few hours specific to dining, also the comfort interval of the lighting is set to follow the sunset and sunrise hours specific to Eastern Europe on April 19th. 8 such datasets are generated, each having apartments from 50 to 400 with an increment of 50 apartments.

6.2. Metrics for the algorithm evaluation

The quality of a meta-heuristic algorithm is measured by its ability to fulfill the objective function as closely as possible, in a reasonable amount of time. Besides analyzing the fitness function, some additional metrics can be employed to elaborate a qualitative analysis of the algorithm. These are particular to meta-heuristic algorithms, not just the Harris Hawk optimization algorithm. As the literature says the key to the performance of such an algorithm relies on its ability to explore as much of the solution space as possible, but still focusing on the areas that yielded the best results. The metrics that measure this ability are the diversity of the search agents, the ratio between exploration and exploitation, and whether the fitness converges towards an extremity.

6.2.1. Diversity

The diversity of the population measures how different solutions are from each other over the optimization process. This metric shows how much the algorithm covers the search space: the greater the diversity, the more diverse areas are explored. This is computed as the mean deviation of the solutions from the median of the same population:

```
diversity = np.mean(np.abs(np.median(pos_matrix, axis=0) - pos_matrix),
axis=0)
```

6.2.2. Exploration and exploitation

As mentioned before, it is crucial for the meta-heuristic algorithm to both explore new areas of the search space, as well as to focus on existing good solutions. This necessity is ensured by balancing the exploration phase when agents are generated randomly onto the search space, and the exploitation phase, when the solutions of agents are moved closer to the solutions of the best-performing agents.

The Mealpy library offers tools to evaluate the balance between exploration and exploitation. However, it is worth noting that the library computes these percentages based only on the diversity obtained over time. Therefore, the exploration will be expressed as the percentage of the diversity at an iteration from the maximum diversity obtained. The exploitation will be $100 - exploration\%$.

Despite this, keep in mind that the energy variable makes us certain about the balance of exploration and exploitation.

6.2.3. Fitness

The overall fitness value is the result of the objective function, and it measures how well the objectives are satisfied by a solution. It is the most important metric of the

optimization algorithm, as it directly relates to how well it can find a solution close to the ideal.

The optimization aims to minimize fitness. The fitness value will decrease or remain constant no matter the hyper-parameters. This behavior is ensured by the nature of the optimization algorithm. However, the rate of decrease is bound to the hyper-parameter configuration.

6.2.4. Distance from the target curve

To evaluate the performance of the algorithm concerning the approximation objective the distance between the consumption resulting from the schedule over a day and the target consumption curve has to be analyzed.

6.2.5. Number of devices scheduled outside comfort hours

In contrast with the approximation fitness, the comfort fitness value doesn't give a detailed view of how well the intervals are obeyed, it just gives a sense of the magnitude.

Counting how many devices are scheduled outside comfort is a suitable metric to give a feel on how well the comfort objective is achieved.

6.2.6. Comfort evaluation charts

Another set of charts that evaluate the comfort objective is comprised of two pie charts that must be interpreted together, they can be seen in Figure 6.2.

The first chart shows the proportion of devices having their schedule in the comfort interval divided into categories. The first category consists of devices having 75%-100% of their schedule in their comfort interval (in the figure given, 42.85% of devices do), the second category comprises devices having 75%-50% of their schedule in their comfort interval, the third category consists of devices having only 50%-25% of their schedule in the comfort interval, and the last category comprises devices having just 25%-0% of their schedule in their comfort interval. The more of the schedule is in the comfort interval the better the objective is satisfied. This is color-coded: categories with a warm color are desired categories, while categories represented with a blueish nuance are undesirable categories.

The second chart has a slightly different meaning. It shows the percentage of devices that have their comfort interval utilized by the schedule in a certain category. A category shows how much of the comfort interval of an appliance is used in a schedule. Like in the previous chart, the categories are divided into four. For instance, the category 100%-75% means that the comfort interval is almost fully used, meaning that the device is scheduled during almost all the hours mentioned in the comfort interval.

The difference between the two charts is subtle. The first one measures how much of the schedule is in the comfort interval, and the second measures how much of the comfort interval is in the schedule. At first glance, they may seem that they show the same aspect, but at a closer look it can be observed, that when a device is not scheduled at all, or it is scheduled only a few hours, the first chart will show a very good performance, as all the hours are in the comfort interval, however, the second chart will indicate that something is wrong, as the comfort interval is barely utilized. For this reason, the two charts have to be evaluated together.

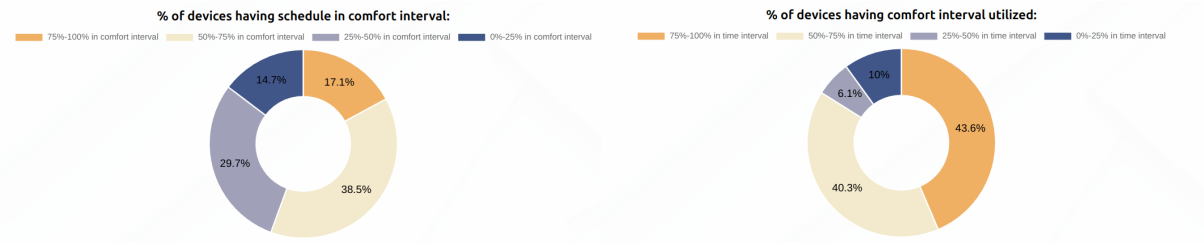


Figure 6.2: Pie charts evaluating the comfort objective

6.3. Configuration parameters

This section will describe the details of configuration parameters and will analyze the impact of each on the algorithm's performance. These parameters have to be uploaded before starting a scheduling process since they directly influence it. A file format was chosen rather than hard-coded values, or values persisted in the database since they are specific to each dataset, and during tuning it is insightful to observe how the performance is affected by it. Configuration parameters marked with * are not dataset-specific.

- **comfort_penalty_base***

  It is the base of the exponential function described in Equation 4.43. The penalty will increase exponentially the more uncomfortable the user is turning on a device at a certain hour.

- **schedule_epoch_no**

  It defines the number of epochs the Harris Hawks optimization algorithm runs for. The larger the value is the more time it runs for, but it won't necessarily mean better results. As the search agents converge toward a solution, exploration is not made any longer, thus no better solution will be discovered.

- **schedule_population_size**

  This configuration parameter specifies how many search agents will be spawned onto the search space of the Harris Hawks algorithm. At each iteration, this many agents will have their optimization parameters recomputed.

- **comfort_tradeoff**

  This configuration parameter measures the willingness of residents to exchange their comfort for a better approximation of the curve. It is applied to the whole dataset. This parameter was used in Equation 4.36.

  The range this parameter can take values from is $[0, 1]$, where 0 means that tenants are willing to sacrifice their comfort to achieve the approximation objective, and 1 means that only the comfort objective should be followed during the optimization.

- **comfort_weight**

  It can take values within the interval $[0, 1]$. It represents the weight of the comfort objective from the upper-level total fitness.

- **approximation_weight**

  It can take values within the interval $[0, 1]$. It represents the weight of the approximation objective from the upper-level total fitness.

- **over_sampling_weight**

  It can take values within the interval $[0, 1]$. It represents the weight of the fitness given by the percent of hours in which the target curve is under the consumption curve that results from devices not included in the chosen subset.

- **subset_weight**

  It can take values within the interval $[0, 1]$. It represents the weight of the fitness given by the percent of apartments included in the subset from the total number of apartments.

- **subset_epoch_number**

  It specifies the number of epochs the Genetic Algorithm from the upper-level optimization runs for.

- **subset_population_size**

  Specifies the number of agents that should be created during the optimization process of the Genetic Algorithm. The more agents are created, the more solution variants are represented at once.

- **subset_mutation_probability**

  It is a number from the interval $[0, 1]$ that is specific to the mutation operation of the Genetic Algorithm described in Section 4.1.5. It specifies the probability with which bits of an agent's solution might be interchanged with a random bit. The higher the probability the more diverse the solutions are, however, convergence is made more difficult.

- **subset_crossover_probability**

  It is a number from the interval $[0, 1]$ that is used in the crossover operation from the upper-level optimization. Bits of the two selected agents will be swapped by a probability given by this parameter. A greater probability will sustain the diversity of the population, however, will make convergence harder.

- **discrete_transform_threshold***

  It contains the value from the interval $[0, 1]$ under which the continuous values of the optimization parameters will be rounded to 0, and above which they will be rounded up.

- **weight_euclidean_obj_metric***

  This configuration parameter measures how much should the fitness of the normalized Euclidean distance function count in the overall approximation fitness (to which the fitness resulting from the Pearson correlation coefficient is added too).

  It is used to test the importance and relevance of the two fitness evaluation methods.

- **weight_pearson_obj_metric***

  In a similar fashion to the previous parameter, this parameter measures how much the normalized fitness resulting from the Pearson correlation coefficient function counts in the overall approximation fitness.

  It is used to test the importance and relevance of the two fitness evaluation methods.

- **baseline_consumption_pearson***

  This configuration parameter measures the upper bound of the interval that the Pearson correlation coefficient can take values from. This is a fixed value that is resulting from Equation 4.41.

- **baseline_consumption_euclidean**

  This configuration parameter sets a value close to the upper bound of the interval from which the fitness value of the Euclidean distance function can take values. This interval ($[0, baseline\_consumption\_euclidean]$) will be as reference for normalization which is defined in Equation 4.37. The value of this configuration parameter is obtained by running the scheduling algorithm to maximize the other objectives, thus a maximum penalization for this objective is obtained.

- **baseline_comfort**

  Similarly to the previous parameter, this configuration parameter sets a value close to the upper bound of the interval the function evaluating the comfort objectives can take values from. This interval ($[0, baseline\_comfort]$) will be used for normalization which is defined in Equation 4.42. It is obtained by running the optimization algorithm to maximize the other objectives.

- **should_adjust_fitness***

  This parameter is boolean, marking whether the values of the fitness functions should be normalized according to Equation 4.35 or not.

- **fitness_reference_value***

  This parameter sets the upper bound of the normalized interval. Its usage is defined also in Equation 4.35.

- **first_level_no_processes***

  This parameter mentions how many processes should be created for the improved version of the scheduling algorithm that is described in Section 6.

- **apply_min_usage_hour_constraint***

  This boolean parameter marks whether the constraint regarding the minimum usage hours of a device should be applied or not.

6.4. Scheduling based on Harris Hawks optimization

In this section, a detailed analysis will be made to evaluate the impact of various design decisions on the performance of the scheduling algorithm.

6.4.1. Hyper-parameter tuning

Hyper-parameter tuning was made using the dataset resulting from the survey.

1. **The epoch number and population size**

   A larger number of epochs won't necessarily lead to better results, as from a point onwards agents will homogenize. This fact can be deduced by analyzing the evolution of the population's diversity (see Figure 6.3) and fitness (see Figure 6.4) over time. After several iterations, the diversity drops drastically, and the search agents will all be holding the same schedule. After that, nearly no adjustment is made.

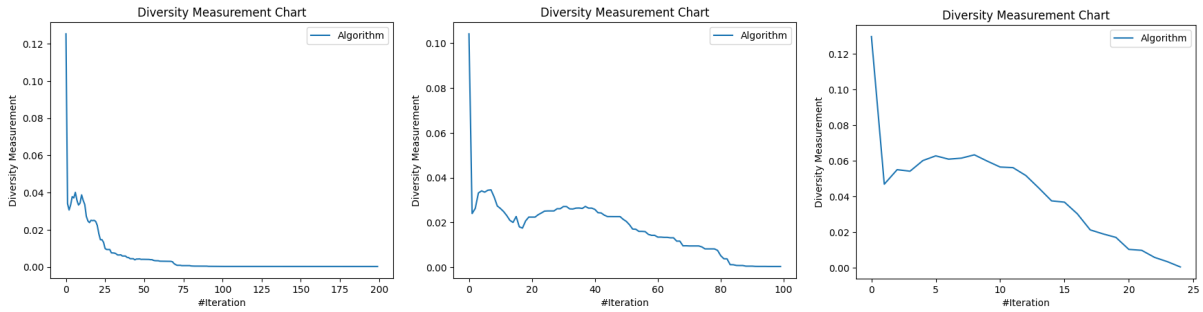


Figure 6.3: Diversity evolution for different number of epochs

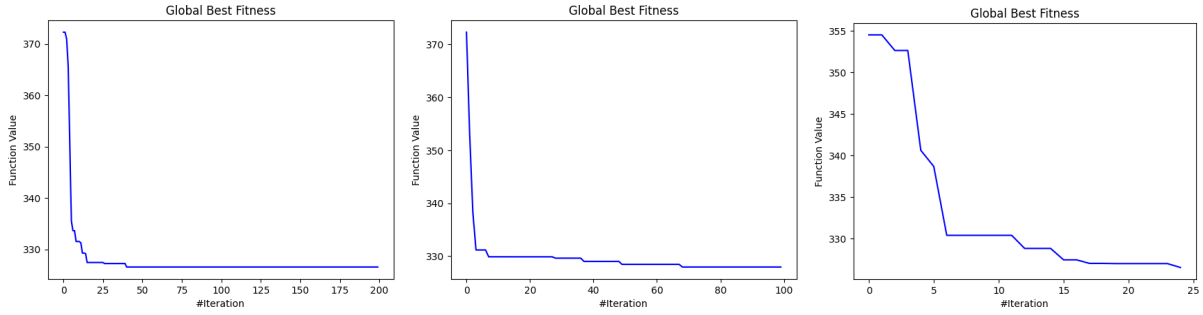


Figure 6.4: Fitness evolution for different number of epochs

As a consequence, the size of the population should be set to a minimum value around which the diversity and fitness approaches and stay close to a global minimum. For this dataset, 25 epochs would be sufficient to find the best solution, more epochs wouldn't yield better results, as the search agents will have the same solution for the rest of the epochs.

The same is true for the *population_size* parameter. This parameter directly impacts diversity (see Figure 6.5) and consequently the area of the search space covered by agents. Despite the obvious benefits it brings for the exploration phase, by having many agents to handle, the execution time increases with an exponential tendency (see Figure 6.7).

From the figures shown, we can observe that by increasing ten times the size of the population, fitness is reduced by only a value of 10 (representing 28.5% of the total fitness reduction), and the running time is increased by a factor of nearly 18. When it comes to diversity, it is increased only slightly by the great population number, as a result making it unfeasible to trade off so much of the running time to have just a few tenths increase in diversity.

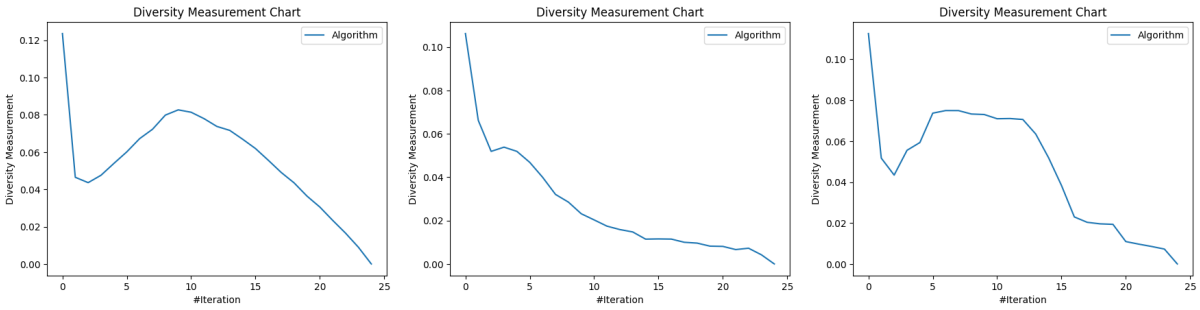


Figure 6.5: Diversity evolution for different population size

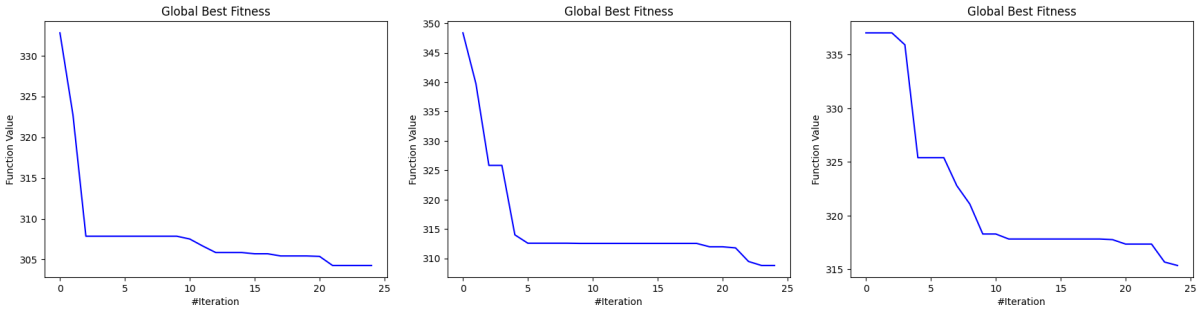


Figure 6.6: Fitness evolution for different population size

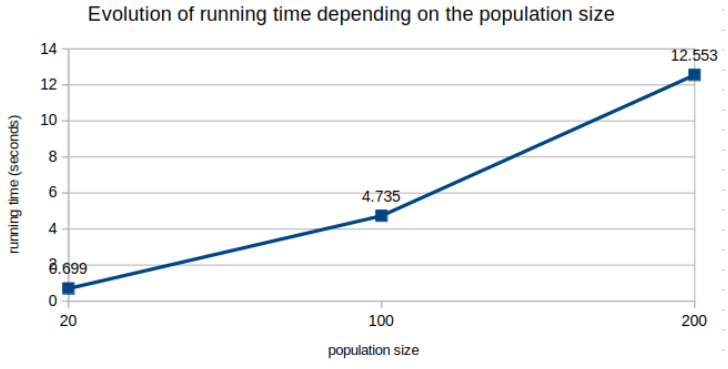


Figure 6.7: Evolution of running time, based on the population size

2. **Comfort tradeoff**

The fitness evolution with different values for the *comfort_tradeoff* parameter is shown in Figure 6.8. It can be seen that the penalty associated with the comfort objective decreases as the user is more restrictive with the discomfort allowed, and at the same time, the distance of the consumption from the target consumption increases since that objective is neglected in the favour of the other.

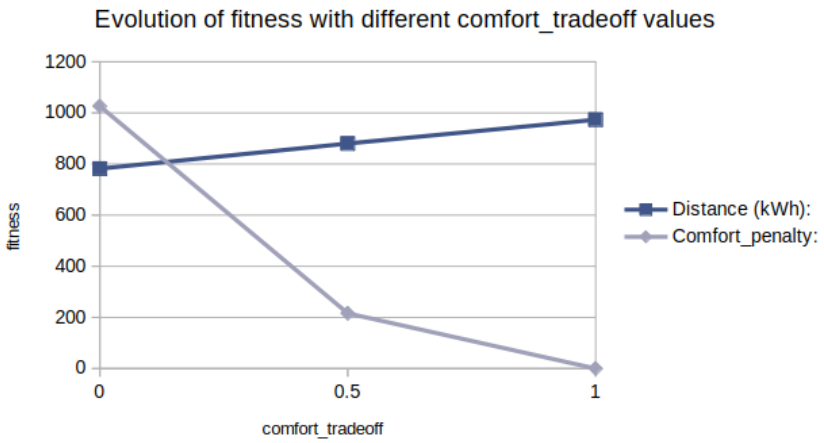
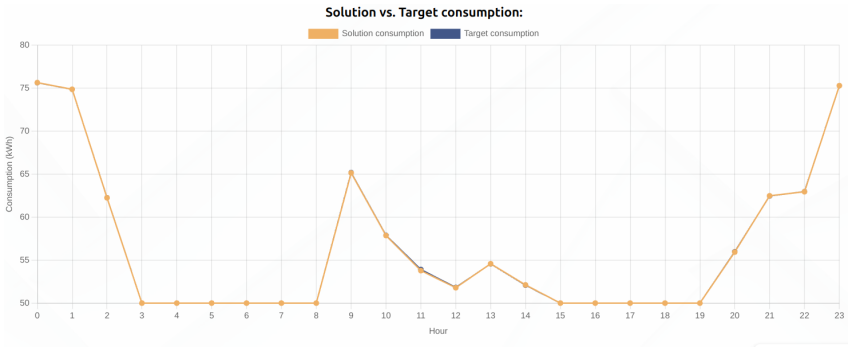


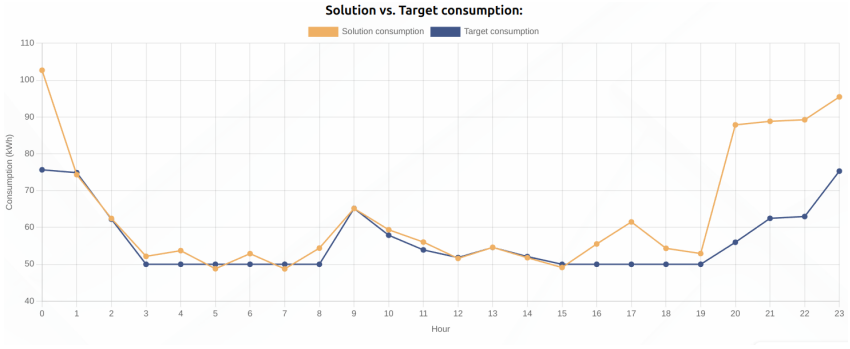
Figure 6.8: Fitness evolution depending on comfort_tradeoff

3. **Minimum usage hour constraint**

In Figure 6.9 it can be seen that if this constraint is applied the distance from that target curve can be significantly larger compared to a schedule in which it is not applied (with all the other parameters left the same). Since all devices must function a minimum number of hours the algorithm will schedule them within their comfort interval, thus disregarding the approximation objective.



(a) Minimum hour constraint not included



(b) Minimum hour constraint included

Figure 6.9: Impact of the minimum usage hour constraint

6.4.2. Evaluation of the impact of larger interval for the initial energy of search agents

In this section, it will be evaluated how the initial energy E0 affects the diversity of the population.

As described in Section 2, the balance between exploration and exploitation is ensured by the energy variable, which has a direct impact on diversity.

The larger the interval E0 can take values from, the larger the energy will be in the initial phases. The decrease of energy over iterations is illustrated in Figure 6.10 for E0 having values from the interval $[-1, 1]$, and $[-2, 2]$.

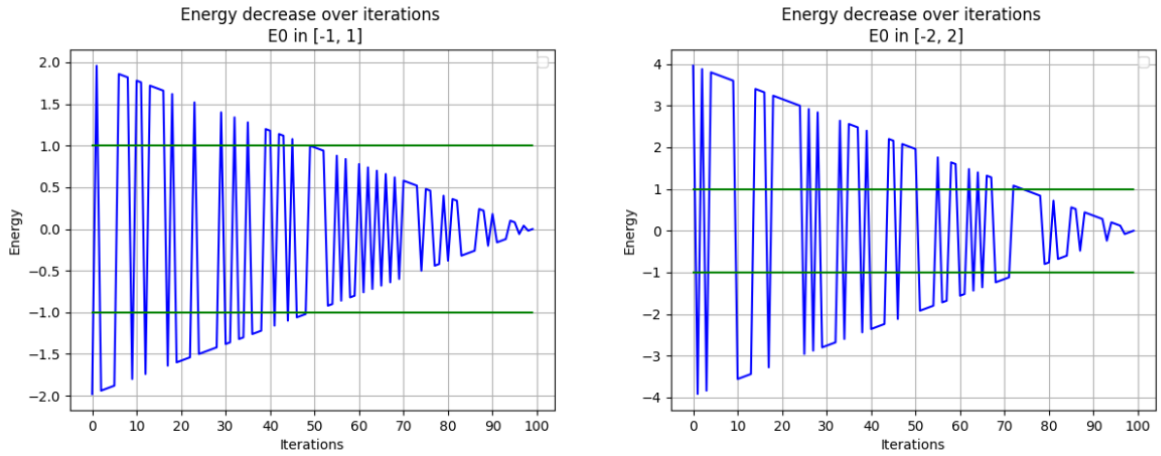
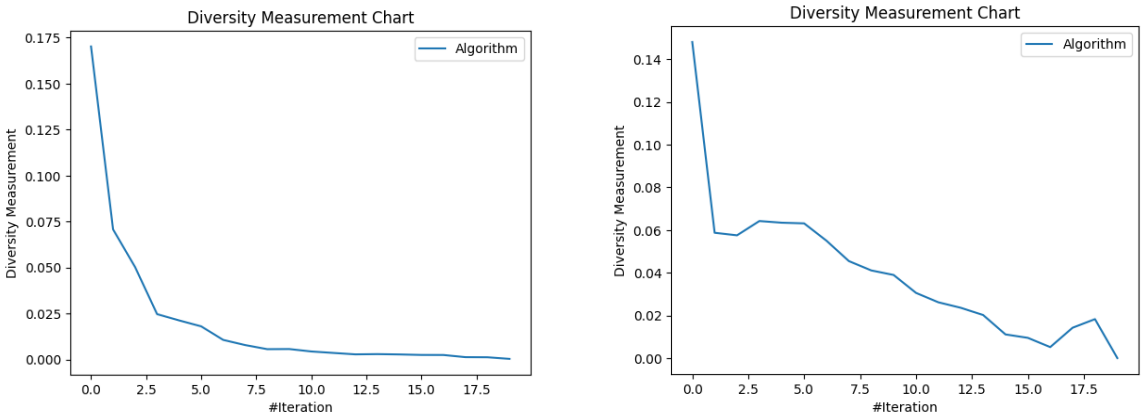


Figure 6.10: Evolution of the energy variable over iterations.

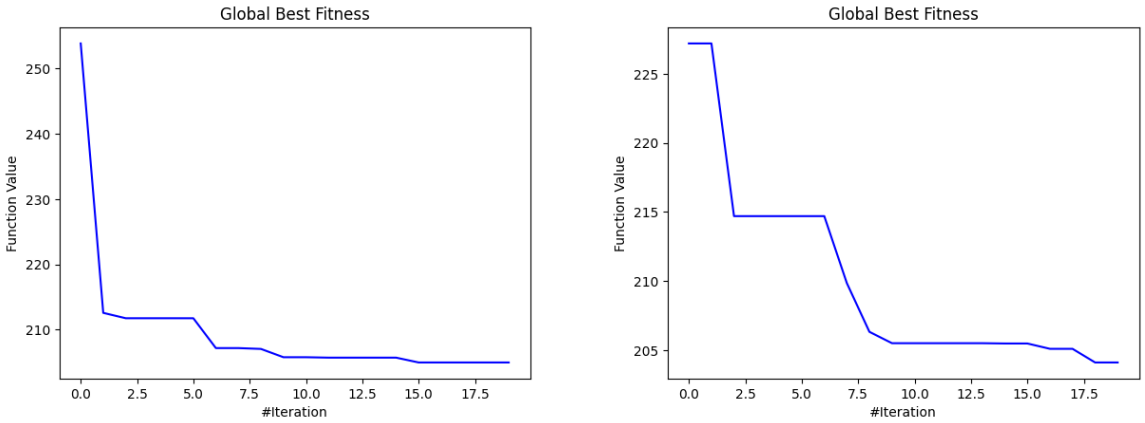
For iterations in which the energy is outside the $[-1, 1]$ interval exploration is made (outside the horizontal green lines), and for energy values within the interval exploitation is performed (inside the horizontal green lines). So, by extending the interval of E0 the algorithm is forced to do exploration for a larger number of iterations. A fact that impacts diversity, and the fitness evolution. In Figure 6.11 it can be observed, that diversity has a slower decrease in case the interval of the initial energy E0 was doubled, compared to the case when a smaller interval is used. In this way, search agents were able to discover a larger solution space which resulted in smaller fitness, as seen in Figure 6.12.



(a) Diversity when E0 in [-1, 1]

(b) Diversity when E0 in [-2, 2]

Figure 6.11: Impact of E0's range on diversity



(a) Fitness when E0 in [-1, 1]

(b) Fitness when E0 in [-2, 2]

Figure 6.12: Impact of E0's range on fitness

6.4.3. Impact of rounding the optimization parameters

There are two variants when it comes to handling the optimization parameters (the parameters each search agent works with): work with binary values throughout the process, or let agents do operations on real values, and round the optimization parameters only when evaluating the fitness. In the following paragraph, the impact of the two candidate implementation decisions will be evaluated.

As a first thought, it would be expected that this modification to impact the fitness of the solution, since having no intermediary position values between 0 and 1, it would take larger random values to move beyond the rounding threshold, thus making it harder to change the value of a binary optimization parameter. The experiments confirm this behavior, as seen in Figure 6.13. The global fitness of the population which was rounded at every iteration decreases slower than the fitness of the solution which was rounded only when computing the objective functions. Not only was there a more significant decrease in global fitness over time in the second case, but the diversity of the population improved as well when the algorithm worked on real values.

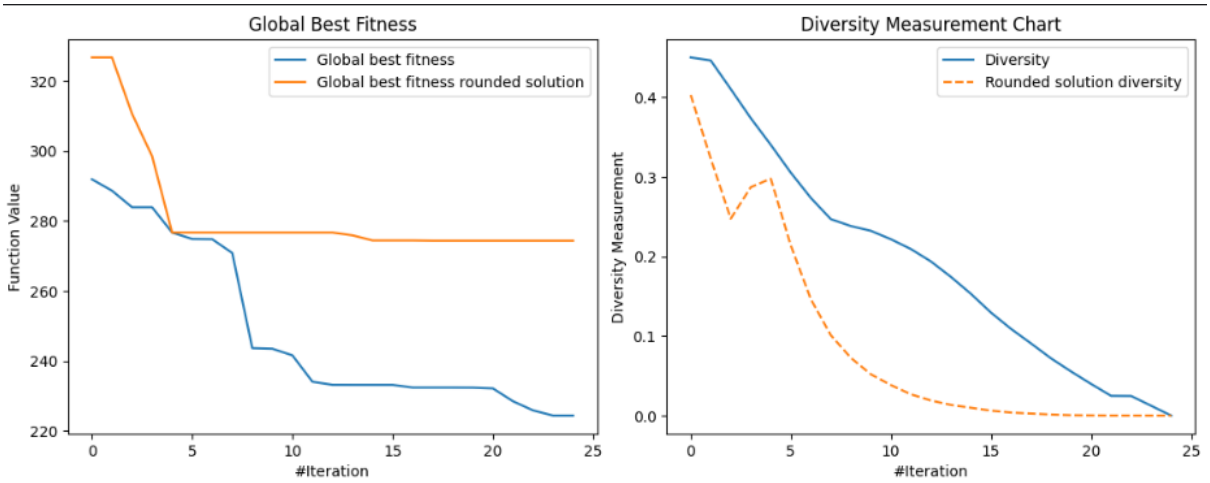


Figure 6.13: Comparing fitness and diversity evolution when rounding at every step, or just at fitness evaluation

It was suspected that as a result of the subtraction and using real values from the interval $[0, 1]$ the values of the optimization parameters of each agent to tend towards 0. This hypothesis was refuted by measuring the average between the optimization parameters of each agent in the exploration phase. It was shown, that the average value of the optimization parameters of an agent is maintained around 0.3, shown in Figure 6.14. Thus the optimization parameters won't tend to 0 even though subtraction is done at each step.

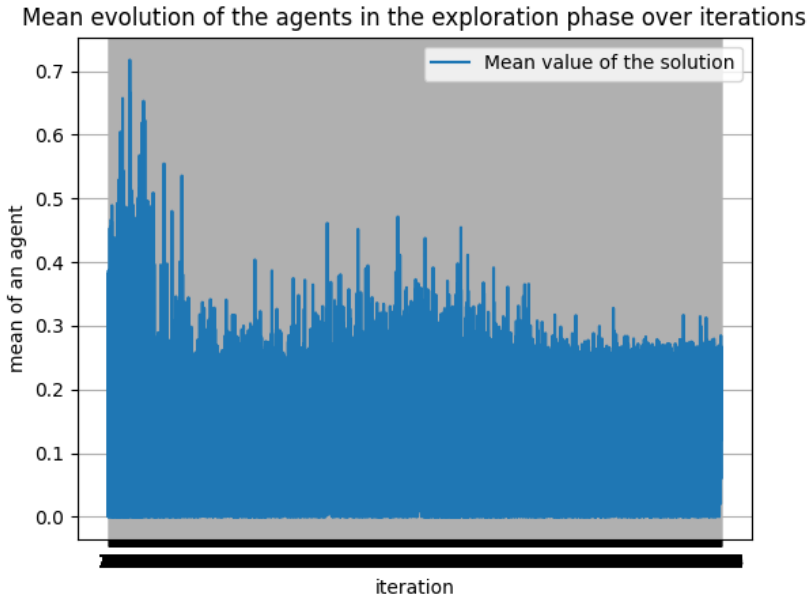


Figure 6.14: Mean values of the agents generated in the exploration phase

6.4.4. Analysis of the Euclidean distance compared with Pearson correlation coefficient as fitness evaluation functions

This section compares the performance of the Euclidean distance and Pearson correlation coefficient metrics in the approximation part of the objective function of the HHO algorithm.

The target curve used in these tests was obtained as follows: the HHO algorithm was used to approximate the curve obtained by reducing the current total consumption over a day by 30%. The result of the optimization was used as the target curve. In Figure 6.15 it can be seen how these curves compare to each other.

In this way it is ensured that the target curve could be achieved with the input data given. So to rule out the situation in which the algorithm could not produce a better approximation because there were not any more devices to work with, or that no combination of devices could get close to the curve. The goal is to obtain a solution curve exactly matching the target curve, knowing that this is possible since the target curve given can surely be approximated with the given data.

To exclude any kind of interference, constraints were disabled when doing the evaluation. Also, the configuration parameter *comfort_tradeoff* was set to 0, transforming it into a single objective problem.

The configuration parameters used for the meta-heuristic are the following:

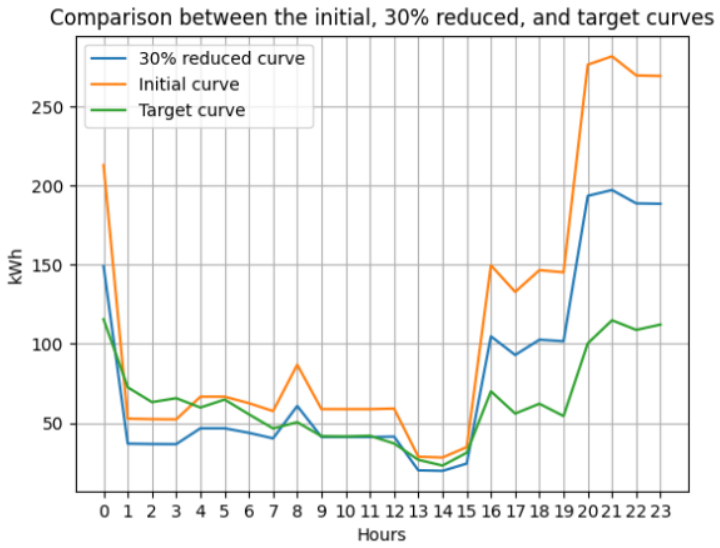


Figure 6.15: Comparing the initial curve, 30% reduced initial curve, and target curve

- the population consists of 300 individuals
- the algorithm runs for 25 epochs

1. **Euclidean distance as fitness function**

   The target curve was obtained by using also the Euclidean distance.

   The average offset over 10 iterations is 84.31 kWh. It can be noted that while the distance at many hours is small, there are a set of hours where the algorithm fails to aggregate electricity consumers in such a way as to get close to the desired consumption. Even more worrying is that from the 3rd to the 5th hour of the day the consumption exceeds the target curve. However, the degree of penalization is not distinct in the case the consumption is under or over the curve. This can be treated as a further improvement, but the objective of the current research is to approximate as closely as possible not focusing on the kind of deviation present. A solution resulting from an optimization using the Euclidean distance as evaluation function is shown in Figure 6.16.

2. **Pearson correlation coefficient as fitness function**

   Besides trying to minimize the distance between two curves, another approach to approximating a target curve is to match its shape. By using Pearson correlation coefficient as evaluation function, the HHO algorithm will try to maximize the coefficient so that the two curves are as correlated as possible.

   Note that the target curve was obtained by using the Pearson correlation coefficient as a fitness metric.

   It is insightful to observe that despite that the algorithm does well in matching the shape of the target curve, it is flawed when trying to get the two curves close together. The cause is that the fact that two curves have a very similar shape does not tell anything about the magnitude of the values that resulted in that shape. As illustrated in Figure 6.17 the second solution has a greater correlation coefficient,

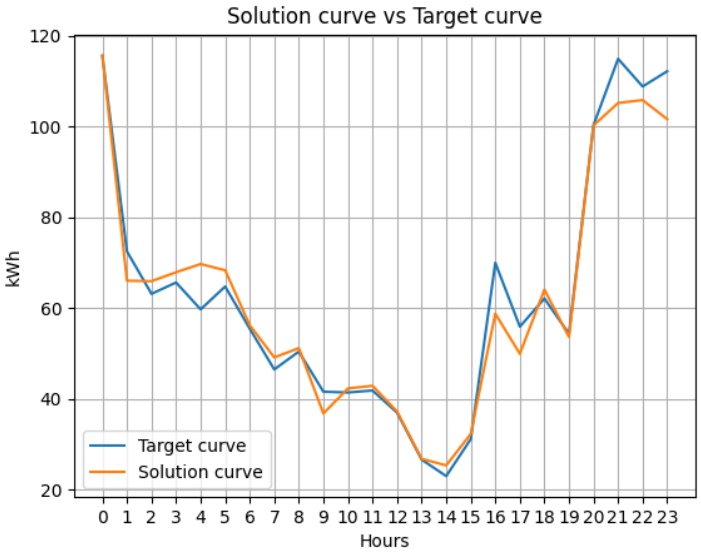
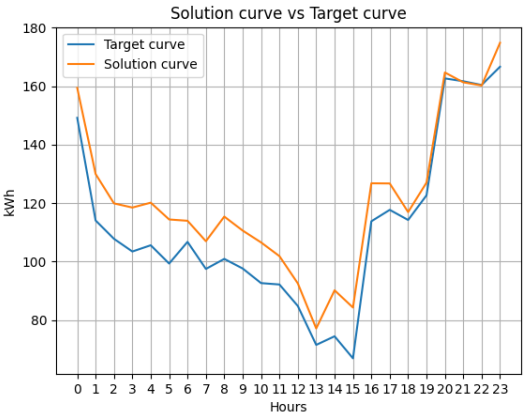
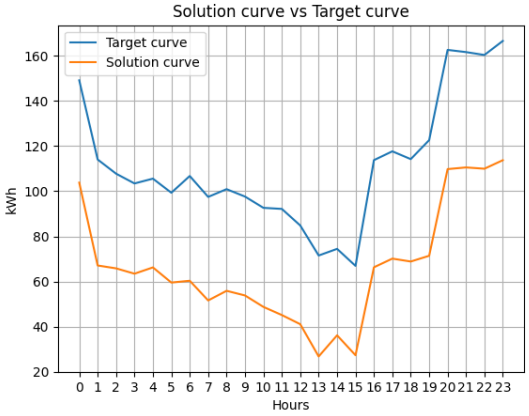


Figure 6.16: Comparison between the target and solution curve when Euclidean distance is used as the fitness evaluation function. Distance: 83.15 kWh

meaning that its shape very closely matches the shape of the target curve, but it is further away from the target curve than the solution which has a smaller correlation coefficient. The average offset over 10 iterations is 186.03 kWh, further emphasizing that the correlation does not ensure a small distance.



(a) Pearson correlation: 0.9869
Distance: 182 kWh



(b) Pearson correlation: 0.9942
Distance: 1009 kWh

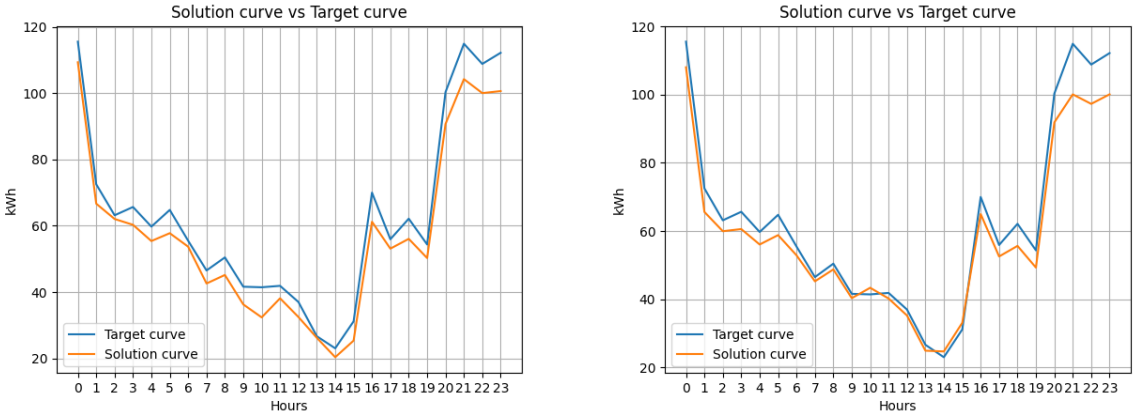
Figure 6.17: Comparison between the target and solution curve when Pearson correlation coefficient is used as evaluation metric

The strengths of this metric are the flaws of the Euclidean distance metric, and vice versa. Thus, a possible improvement would be to combine these two metrics to reduce the distance between the curves, and at the same time match the shape of the target curve, this way eliminating the hours when spikes are present.

3. **Comparison between Euclidean distance and Pearson correlation coefficient as evaluation metrics**

   Since the target curve was obtained in both cases by using the same objective function it might be relevant to test the behavior of the objective functions in case the target curve was obtained by running the first optimization using the other metric.

   Thus Figure 6.18 captures the result obtained using the Pearson correlation coefficient trying to approximate a curve resulting from an optimization that used the Euclidean distance as the evaluation metric.



(a) Pearson correlation: 0.9966
Distance: 72 kWh

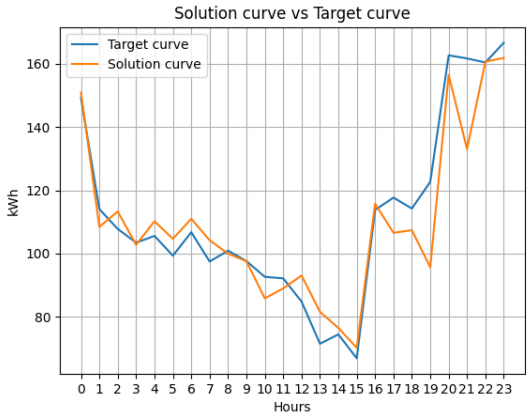
(b) Pearson correlation: 0.9974
Distance: 70 kWh

Figure 6.18: Pearson correlation coefficient as a metric in the case the target curve is obtained using Euclidean distance

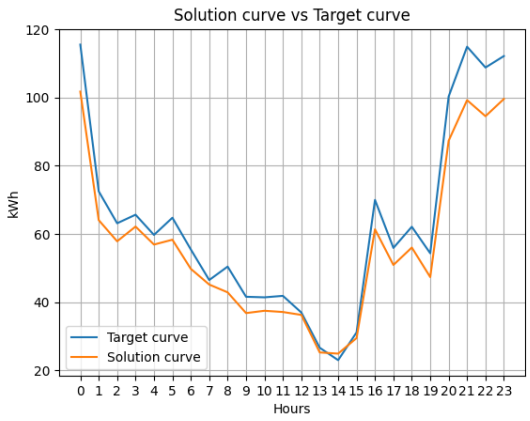
It can be noted that in this case, the algorithm yielded a better Pearson correlation coefficient, than in the case the target curve was obtained using the same metric. Even more surprising is that the average offset over 10 iterations is 76.68 kWh, which is the smallest offset found so far, despite that the correlation coefficient does not enforce proximity.

In Figure 6.19 it is shown how the Euclidean distance metric works on a target curve obtained using the Pearson correlation coefficient. It can be seen that it has a larger offset than in the case when it was used on a target curve computed utilizing the same function. The average offset over 10 iterations is 156 kWh. It is still better than the Pearson correlation coefficient let alone, but worse than any other result.

After the four tested scenarios, we can make several conclusions. The Euclidean distance ensures proximity to the target curve, however, the shape of the curve is not uniform and is prone to spikes in consumption. Because the Pearson correlation coefficient does not measure distance it is more likely to yield a result with a greater offset from the target curve, but matching very closely its shape, and having a smoother transition between hours, than the Euclidean distance metric would. It is also worth mentioning that it has a better performance in terms of offset in case the target curve was obtained using the Euclidean distance function, however this is attributed to a coincidence.



(a) Pearson correlation: 0.9592
Distance: 155 kWh



(b) Pearson correlation: 0.96625
Distance: 140 kWh

Figure 6.19: Euclidean distance as metric in the case the target curve is obtained using Pearson correlation coefficient

4. **Mixed fitness evaluation metrics**

   By taking the best from each scenario we could obtain the best results. Thus, in this part, we will evaluate the performance of the model when the target curve is obtained using Euclidean distance, and the optimization algorithm is evaluated by combining the results of the two metrics.

   For this evaluation Equation 4.37 is used, taking the linear combination of the normalized values of the two objective functions.

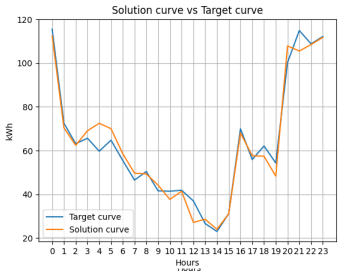
   The average offset in the configuration where the two metrics had equal weight over 10 iterations is 82.96 kWh, and the average correlation is 0.98.

   An improvement is visible compared to the cases when only one evaluation metric is employed. The solution curve is more flat, without spikes, matching very closely the shape of the target curve, due to the Pearson correlation coefficient component. Also, the distance from the target curve is minimal as a result of having a component that computes the Euclidean distance.

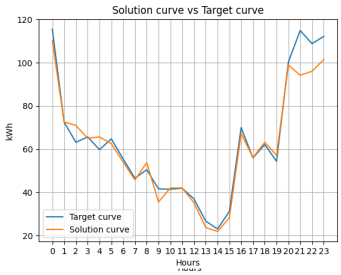
   It would be insightful to see how the performance changes when the weights of the two metrics are imbalanced. In Figure 6.20 one can see a comparison between the cases when the fitness resulting from the Euclidean distance metric and the Pearson correlation coefficient have different weights.

   From the comparison of the three cases, we can conclude that the ratio in which the two evaluation metrics are present has not a significant impact on the overall offset of the solution from the target curve. However, in the case in which the Pearson correlation coefficient has a greater weight, there is a slight improvement, compared to the others.

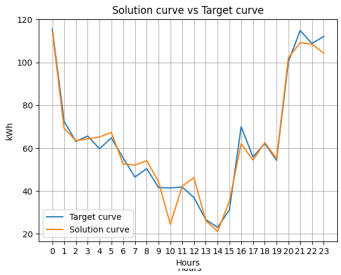
   But it is worth keeping in mind, that for an unknown reason, this evaluation metric kept a smaller offset from this objective curve, even though it does not have a distance factor in it. For this reason, if we try to run the algorithm on the target curve obtained using the Pearson correlation coefficient the importance of the weights might change. As a reminder, both metrics had a difficult time approximating this curve.



(a) Euc. dist. weight: 0.25
Pearson corr. weight: 0.75
Pearson correlation: 0.9859
Distance: 84 kWh



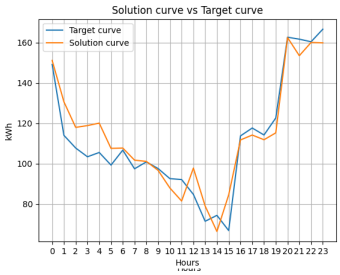
(b) Euc. dist. weight: 0.5
Pearson corr. weight: 0.5
Pearson correlation: 0.9838
Distance: 91 kWh



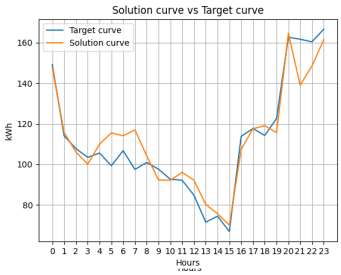
(c) Euc. dist. weight: 0.75
Pearson corr. weight: 0.25
Pearson correlation: 0.983
Distance: 86 kWh

Figure 6.20: Comparing the impact of weights for combined evaluation metric on the target curve obtained using Euclidean distance

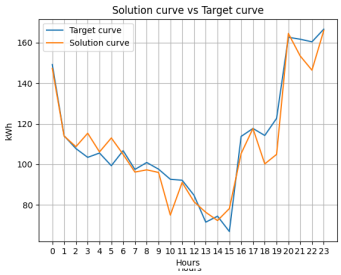
In Figure 6.21 are shown the results of this experiment. Opposed to the expectations, having a greater weight for the Pearson correlation coefficient still resulted in a slightly better result, than the other scenarios. But the difference between the results is even smaller, than in the case of the other objective function.



(a) Euc. dist. weight: 0.25
Pearson corr. weight: 0.75
Pearson correlation: 0.95
Distance: 160 kWh



(b) Euc. dist. weight: 0.5
Pearson corr. weight: 0.5
Pearson correlation: 0.949
Distance: 165 kWh



(c) Euc. dist. weight: 0.75
Pearson corr. weight: 0.25
Pearson correlation: 0.95
Distance: 163 kWh

Figure 6.21: Comparing the impact of weights for combined evaluation metric on the target curve obtained using Pearson correlation coefficient

5. **Concluding the comparison of fitness functions for the approximation objective**

   It was shown that while the Euclidean distance metric results in a solution that minimizes the offset from the target curve, the Pearson correlation coefficient excels in building a solution that closely matches the shape of the given curve, but fails at minimizing the offset. As a result, a combined evaluation metric was proposed with different weights for the fitness given by the metrics. In this method, even though the two metrics complemented each other's weaknesses, the results were poorer, than in the case when just the Euclidean distance was taken into account. It was also discovered, that the Pearson correlation coefficient metric reduced the offset remarkably from the target curve obtained using the Euclidean distance for unknown reasons, but performed the poorest on the curve obtained by the same evaluation metric.

Evaluation metric	The offset given from the target curve obtained using the evaluation metric		**Variation ratio**
	Euclidean distance	**Pearson corr. coeff**	
Euclidean distance	84.31 kWh	156 kWh	**0.134**
Pearson corr. coeff.	76.68 kWh	186.06 kWh	**0.335**
Mixed evaluation metric	87 kWh	162.66 kWh	**0.779**
Average	*82.66 kWh*	*168.23 kWh*	

Table 6.1: Comparing the distance resulting from each scenario and the variability ratio

Since each configuration had a better performance on different cases, when it comes to choosing the best among them, the one has to be chosen which can give the best solution given a general case. In Table 6.1, the scenarios tested are compared. The variation ratio is computed as the ratio of deviation from the mean of all three evaluation metrics in case the target curve was obtained using the Euclidean distance, and the case when it was used using the Pearson correlation coefficient. A value close to 1 indicates that the variation is low, thus showing a better generalization capability of the metric.

In this regard, the mixed evaluation metric is best at giving an acceptable solution for a greater range of target curves. If we consider the weight rates for each metric, it was shown that a higher rate for the Pearson correlation coefficient yields slightly better results.

In conclusion, the best-performing fitness evaluation metric found was a mix between the Euclidean distance metric and the Pearson correlation coefficient metric with a weight of 0.4 and 0.6 respectively.

6.4.5. Comparison between the base version, the improved version, and the greedy optimization

To be able to evaluate the performance of the HHO algorithm in depth there is a need for a second algorithm as a benchmark, which uses a different approach in solving the same problem. The easiest algorithm to build is a greedy algorithm that uses the knowledge from the problem's domain to make a targeted search in the space. All the features have to be implemented in the algorithm that is present in the HHO to have a fair comparison.

The definition of the greedy optimization method proposed to solve the target curve approximation objective while minimizing the impact on the comfort of the building tenants can be seen in Appendix A.3. It uses a knapsack approach. The algorithm starts with an initial solution (which can be a matrix containing all zeros, or an attempt of a previous scheduling algorithm), and iteratively builds the schedule by adding and removing devices. As input, it also receives the constraint matrix, comfort matrix, and consumption vector of the residential building.

1. **Results of the greedy algorithm**

   The algorithm was run on the dataset used to evaluate the HHO algorithm as well. The results are shown in Figure 6.22, and Figure 6.23. It was able to approximate the solution curve within 0.034 seconds with an offset from the target curve of 48.72 kWh, and a comfort penalty of 4194. Because of the implementation which takes into account the comfort objective as well, the output solution managed to schedule 64.74% of the devices fully within their comfort interval, and 21.79% of the devices had their comfort interval fully utilized. The resulting deviation is due to the minimum usage, and non-deferrable constraints.

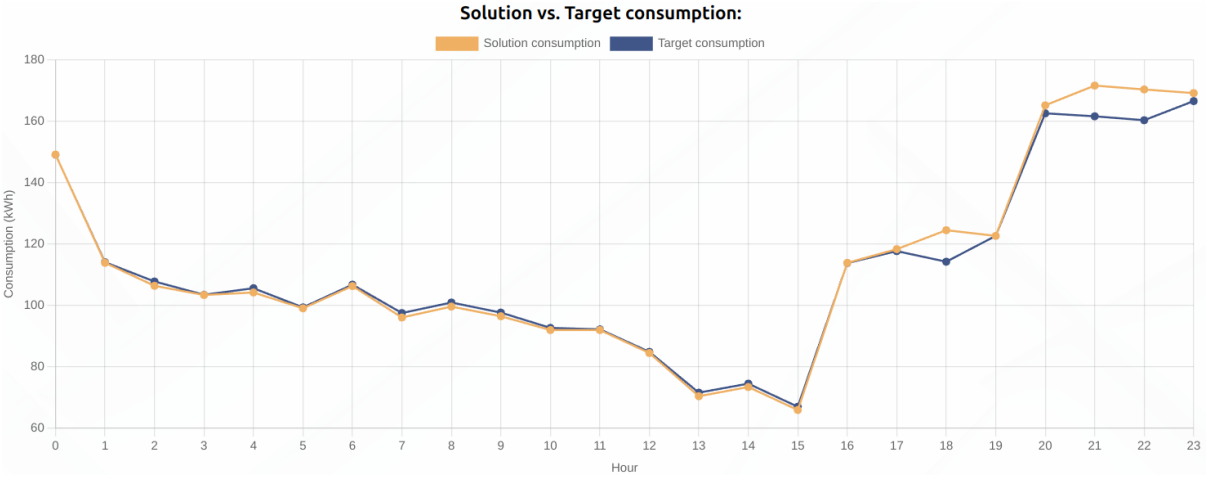


Figure 6.22: Approximation of the target curve of the greedy algorithm

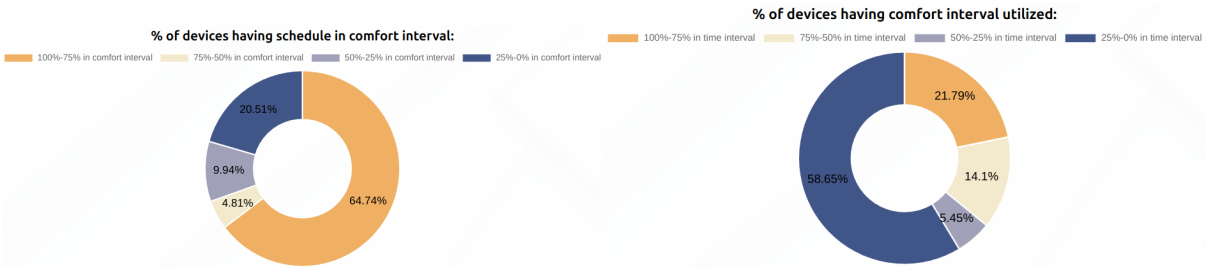


Figure 6.23: Comfort objective evaluation of the solution given by the greedy algorithm

The results are outstanding for a greedy algorithm having to optimize the consumption of 312 devices at one hour under two constraints. Even more impressive is that it does this while keeping a low discomfort. It would be insightful to see the performance of the two algorithms side-by-side and to evaluate it with a considerably higher number of apartments involved.

2. **Comparison between HHO and the Greedy optimization**

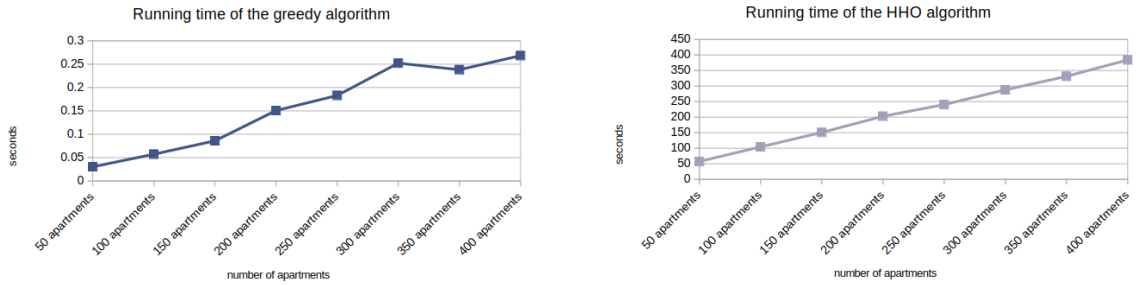
   The comparison presented below is made for a dataset generated with 50 to 400 apartments, each 50% of the apartments having 5 devices, and 50% having 7 devices. There are 2 types of tenants, those who have 5 devices are using them in the mornings, and late evenings, while those with 7 devices use them a few hours in the morning, and a larger interval in the evening. The initial configuration is made to respect the comfort interval, minimum usage constraint, and the non-deferrable

property of some device categories. Two target curves were used; one obtained with a 30% reduction of the consumption that is above the minimum consumption of the building (the minimum consumption is given by the non-deferrable devices, and it cannot be further reduced), and one with the reduction mentioned above, and a shift of consumption with 2 hours towards the evening.

(a) **Running time comparison**

The running time of each algorithm was measured for each configuration of the apartments, but only using the target curve obtained by reduction of consumption, since this characteristic is not affected by the shape of the curve. The hardware and the configuration used were the same for each test.

As shown in Figure 6.24, it was observed, that in contradiction to our intuition, the greedy algorithm had a net superior running time, of just fractions of a second, while the HHO algorithm had a second running time for each apartment added. The running time of both devices increases linearly together with the number of apartments.



(a) Running time evolution of the greedy algorithm with increasing number of apartments

(b) Running time evolution of the HHO algorithm with increasing number of apartments

Figure 6.24: Running time comparison between the greedy algorithm and the HHO algorithm

(b) **Fitness comparison**

For comparing the fitness of the approximation and comfort objective for the two algorithms, the same dataset was used. Additionally, in this case, besides the target curve obtained by reduction, the algorithms were evaluated also on the one obtained by applying the shift operation. It was done so, to evaluate the algorithms in a more general case.

i. **Approximation objective fitness**

As seen shown in Figure 6.25, the HHO algorithm has a much larger fitness and, thus a poorer performance compared to the greedy algorithm on the approximation objective.

It can be also observed that the fitness resulting from the HHO algorithm doesn't change depending on whether the target curve was shifted or not, meaning it is able to handle a much wider range of scenarios. On the other hand, the greedy algorithm is more sensitive to the shape of the target curve, which is a pitfall. It shows that it cannot adapt.

If we look into the details it can also be seen that the fitness from the HHO algorithm has a steeper increase, than the fitness from the greedy

algorithm. The curve showing the HHO algorithm increases linearly with the number of apartments, but the curves showing the greedy algorithm have an attenuated linear relationship.

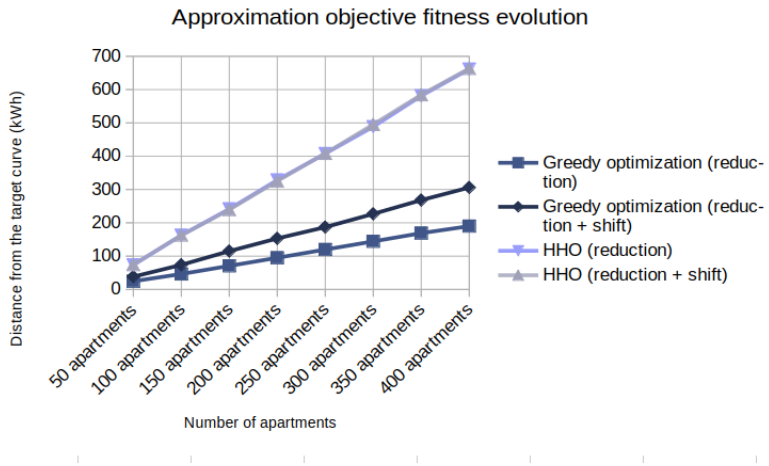


Figure 6.25: Comparison between the fitness of the approximation objective given by the algorithm variants

ii. **Comfort objective fitness**
Similarly to the previous objective, the HHO algorithm performs poorly no matter the type of the target curve, as shown in Figure 6.26. However, it is consistent, not like the greedy algorithm that has an excellent performance in case the target curve is not shifted, but a significantly bad performance if the target curve is shifted. The outstanding performance is probably resulting from the fact that it maintains a list of devices ordered by comfort, and because of this the greedy algorithm can find only devices within the comfort interval, since those intervals map exactly onto the target curve if it was just reduced. This performance gap exposes a major pitfall of this algorithm; it is unable to maintain its performance in a general scenario.

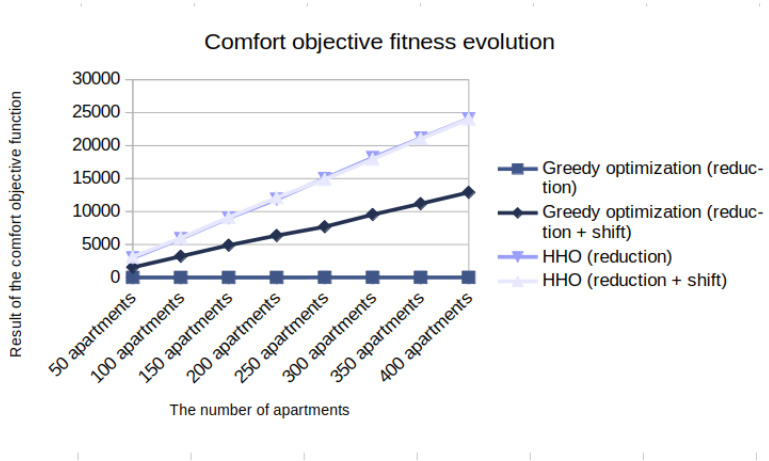


Figure 6.26: Comparison between the fitness of the comfort objective given by the algorithm variants

3. **Results of the improved version of the HHO algorithm**

   It was suspected that the HHO algorithm fails to approximate the target curve because of the large number of parameters it has to adjust. Therefore the optimization algorithm was run for each hour separately, thus reducing the number of optimization parameters by a factor of 24, as presented in Section 6. The dataset used to evaluate the performance of this version of the HHO algorithm is the same as the one used to evaluate the greedy algorithm, namely, the configuration with 39 apartments. The algorithm ran for 1.27 seconds to make the schedule. The distance from the target curve was 13.72 kWh (see visualization in Figure 6.27), and the comfort penalty obtained was 2487 (see evaluation in Figure 6.28).

   The results are superlative compared to the initial version of the scheduling algorithm in which all the hours were included at once in the optimization. To feel the magnitude of the parameter reduction, in the initial version, the HHO algorithm received 7488 optimization parameters to work with, while in this improved version it only receives 312 at once. A more manageable amount. Furthermore, since a schedule from one hour to the other is independent the benefits of parallelization can be exploited.

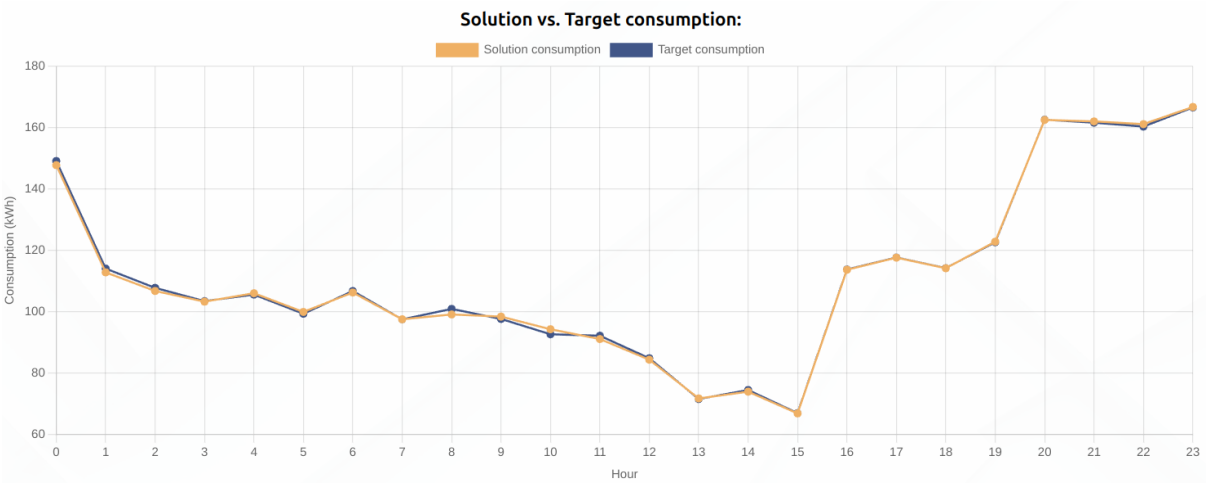


Figure 6.27: Approximation of the target curve of the improved HHO algorithm

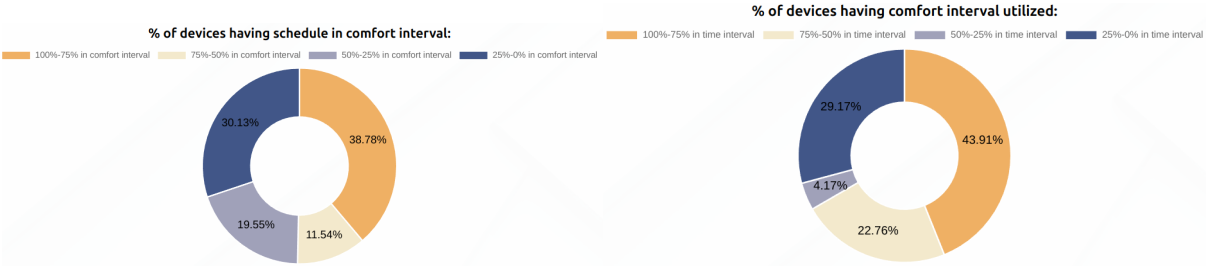


Figure 6.28: Comfort objective evaluation of the solution given by the improved HHO algorithm

(a) **Running time comparison**

In Figure 6.29 it is shown how the running time evolves with the number of apartments included in the demand response program. Compared to the initial version of the algorithm, the speedup is substantial. This version is 50 times faster than the base version of the HHO algorithm. By evaluating the performance without multiprocessing it was established, that having multiple processes run in parallel decreases the running time by a factor of 2. The rest of the improvement is thanks to the smaller number of parameters that have to be subtracted, multiplied, evaluated, and so on. When compared to the greedy algorithm, the running time is still greater than the improved version by a factor of 61. Meaning that the improved version was able to halve the gap between the greedy and the base version of the HHO algorithm.

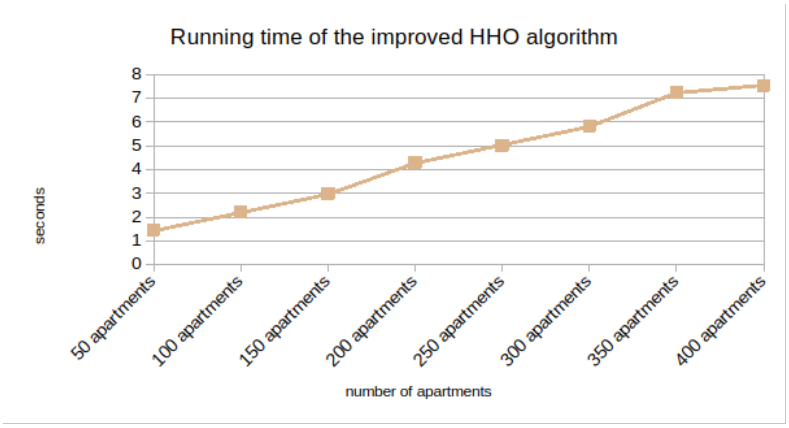


Figure 6.29: Running time of the improved HHO algorithm

(b) **Fitness comparison**

The fitness values from the improved version are compared with the values resulting from the other algorithms evaluated.

i. Approximation objective fitness
The best-performing algorithm for the approximation objective is the improved HHO algorithm for all scenarios tested (ranging from buildings with 50 apartments to buildings with 400 apartments). It not only outperforms all other implementations but it exhibits a valuable capability of generalizing, since the fitness is not dependent on the target curve used, in contrast with the greedy algorithm. Figure 6.30 displays how the algorithms compare to each other in different scenarios.

ii. Comfort objective fitness
When it comes to the comfort objective, the scoreboard changes. The improved version of the HHO algorithm scores below the greedy algorithm no matter the target curve used. However, it still performed better than the base version. In Figure 6.31 is visible this fact. Even more, in this case, there is a performance gap in the improved HHO algorithm depending on the target curve used.

As it was mentioned before, one of the most important characteristics of the metaheuristic algorithm is its capacity to make trade-offs between the objectives. In

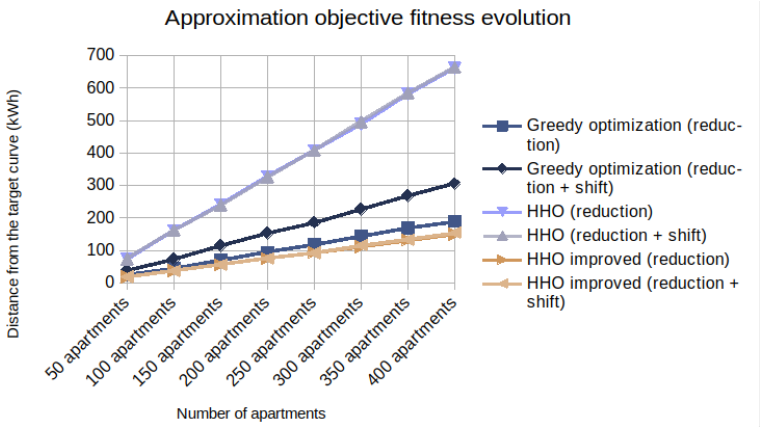


Figure 6.30: Comparison of the approximation objective fitness between the improved algorithm, and the other versions

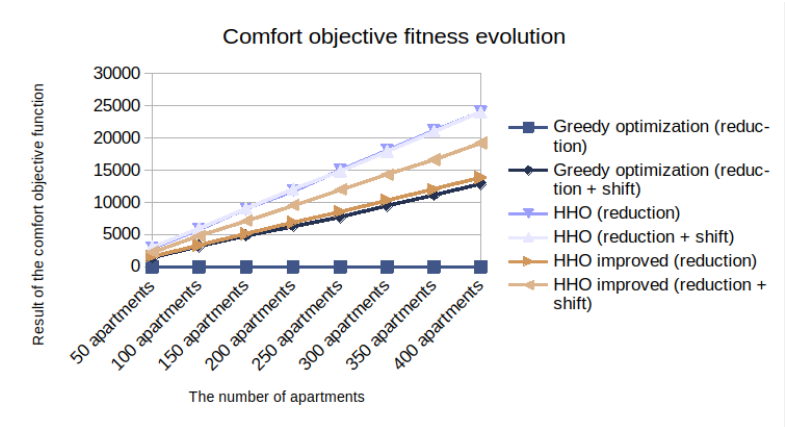


Figure 6.31: Comparison of the comfort objective fitness between the improved algorithm, and the other versions

Figure 6.32 it can be noticed, that the Harris Hawks Optimization algorithm can prioritize between the objectives by adjusting the by which each contributes to the overall fitness. The weight can be adjusted based on the preferences of tenants from a residential building.

4. **Concluding the comparison of implementation versions**

   It was shown that the base version of the HHO algorithm performs the poorest on all comparison terms. Therefore, it is not feasible to be used at all. However, the performance between the improved version and the greedy algorithm is tight. Therefore, it has to be outlined which scenarios are suitable to be used.

   The greedy algorithm, by its nature, cannot make a tradeoff between the objectives. It gets the best devices to fulfill one objective, and then in its remaining movement space, it tries to enhance the other objective. For this reason, it would be a net superior solution for a problem with only one objective. For the problem we are trying to solve, it is suitable only if the target curve maps directly onto the comfort intervals, meaning that there will always be a set of devices to schedule that would fulfill the approximation objective, and that are also in their comfort intervals.

   On the other hand, the strength of the improved version of the HHO algorithm is

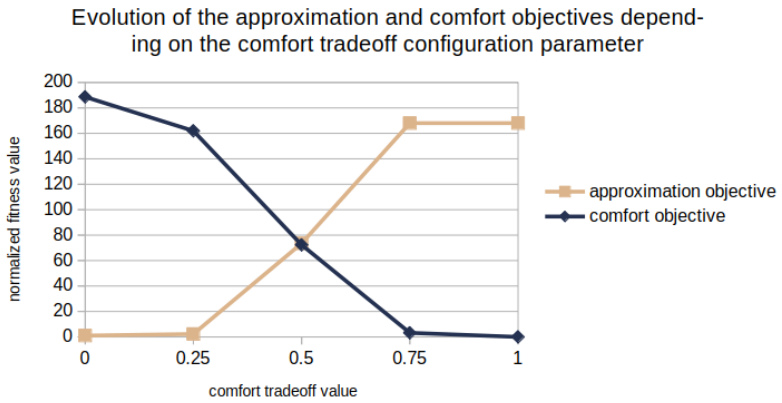


Figure 6.32: Impact of the comfort_tradeoff configuration parameter on the fitness values of the two objectives

that it is capable of adjusting the schedule based on the preferences of residents. This means, that if they are desiring more comfort over the approximation objective, the algorithm can prioritize that objective over the other.

In the case of this problem, the greater running time of the improved version of the HHO algorithm compared to the greedy algorithm is acceptable in the benefit of better objective balancing, and better fitness values.

6.5. Scheduling a minimum subset of apartments chosen by the Genetic Algorithm

In this section, an analysis will be presented regarding the impact of design decisions on the performance of the subset-making algorithm. Evaluating this level can only be done when the lower-level optimization yields good enough solutions.

6.5.1. Hyper-parameter tuning

Hyper-parameters that strictly concern this part of the optimization can be divided into two categories: those which are specific to the standard Genetic Algorithm, and those specific to our implementation, namely the weights of each objective. The dataset used for analysis is the one obtained from the survey.

1. Genetic algorithm-specific parameters

   The configuration parameters specific to the Genetic Algorithm are the *subset_epoch_number, subset_population_size, subset_mutation_probability*, and *subset_crossover_probability*. The impact on the running time of these parameters is significant, as for each added agent a new lower-level optimization will be triggered at each epoch, and for each added epoch the number of new low-level optimizations will be equal to the number of agents. Thus by careful analysis, the lowest amount of epoch number and population needs to be selected, such that the impact on the quality of the solution is minimal.

   In Figure 6.33 the evolution of diversity is shown depending on the number of epochs, similarly in Figure 6.34 the evolution of fitness is shown for the number of

epochs. It can be observed that even by tripling the number of epochs neither the diversity nor the fitness is significantly better. The smaller number of epochs means greater fitness just by 0.7 fitness values (from an interval from 0 to 100). However, when the number of epochs is greater, the diversity is maintained better ensuring an improved exploration of the search space. Since fitness is stagnating after 30 epochs, it is not worth going beyond it.

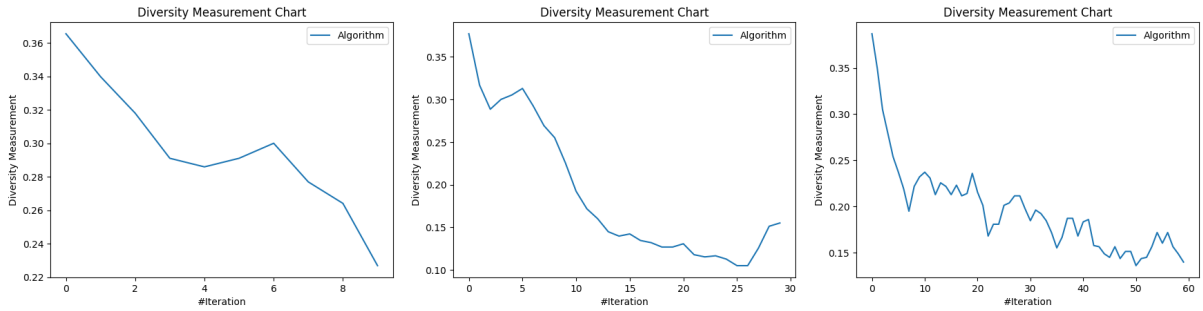


Figure 6.33: Diversity evolution for different number of epochs

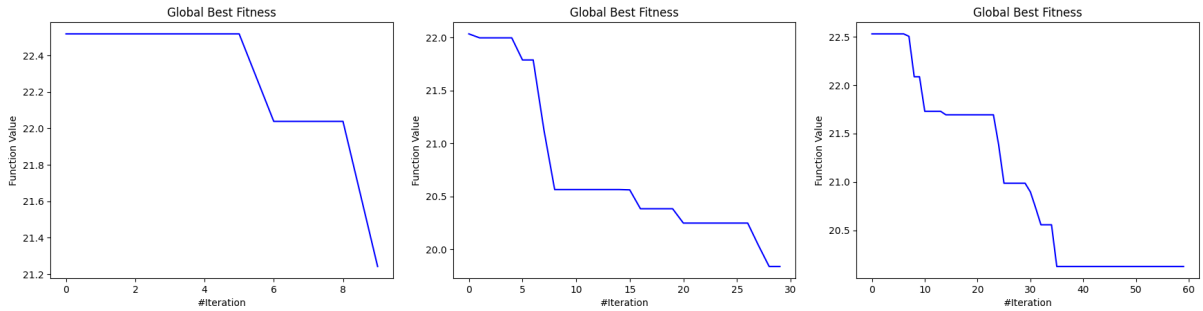
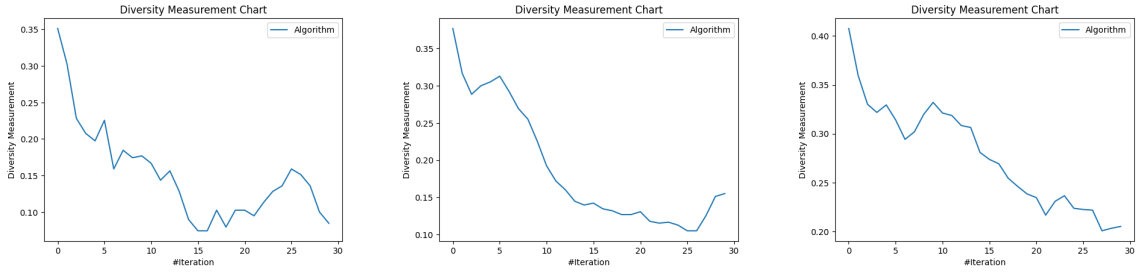


Figure 6.34: Fitness evolution for different number of epochs

When examining the impact of population size we can observe that there is no impact on the fitness of the solution but on the algorithm's ability to maintain an increased diversity over iterations. These facts can be seen in Figure 6.35 and Figure 6.36.

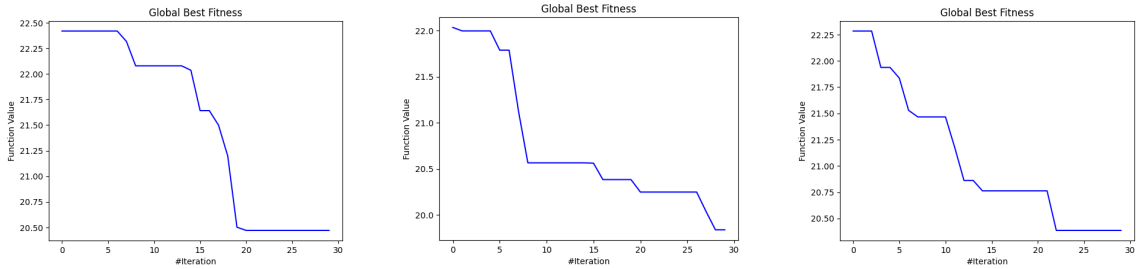
Since a greater value for the two parameters is not feasible from the running time perspective a trade-off should be made. It is suggested to have a middle-sized number of epochs and a smaller population. The purpose of this is to reduce running time. A smaller population will mean less diversity, but this metric can be enhanced by tuning the crossover and mutation probabilities that will be described in the following paragraph.

Another two parameters that are specific to Genetic Algorithms are the crossover and mutation probabilities. These directly contribute to the population's diversity as they influence how often will the bits be interchanged, or modified from a solution. In Figure 6.37 four configurations of these two parameters were analyzed. It was observed that diversity is best maintained with a higher value for the two parameters, therefore values around those are recommended, even more so, if a smaller population is generated.



(a) subset_population_size = 10 (b) subset_population_size = 30 (c) subset_population_size = 40

Figure 6.35: Diversity evolution with respect to the population size



(a) subs_pop_size = 10 (b) subs_pop_size = 30 (c) subs_pop_size = 40

Figure 6.36: Fitness evolution with respect to the population size

2. Weights of the fitness values

   Weights for each objective will directly determine how well the algorithm can fulfill each of them. The fitness values related to the *subset_weight* and the *over_sampling_weight* both penalize the solution if it includes too many apartments, however from different perspectives. Therefore, the values of their weights have to be balanced, such that if the algorithm chooses a subset of apartments that triggers both objectives, the overall weight of the upper level not to shadow the fitness values from the lower level. By adjusting the *comfort_weight*, and *approximation_weight* the importance of the fitness values from the lower-level optimization can be adjusted.

6.5.2. Performance evaluation

The performance of the Bi-Level architecture is evaluated in this section using the same datasets as in the evaluation of the lower-level optimization.

In the first instance, the algorithm was run on the survey dataset. Figure 6.38 shows how the target curve is approximated by the solution which modified the consumption pattern of only a subset of apartments, and in Figure 6.39, and Figure 6.40 the impact on comfort can be viewed. The solution included 48.7% of the residential building's apartments, the overall consumption for the building deviated from the target curve by 18.18 kWh, and the comfort penalty was 1365.

These results are significant compared to the solution given by only the first-level optimization. Despite the approximation objective being greater with 4.46 kWh, which is a small amount compared to 312 devices, the comfort objective is improved by 45.11%, nearly halving the penalty. All this with fewer devices included in the demand response program.

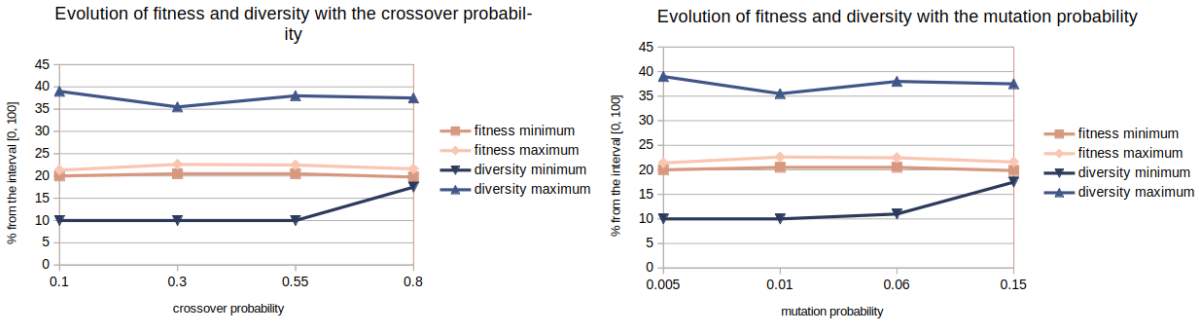


Figure 6.37: Fitness and diversity evolution for different values of crossover and mutation probabilities

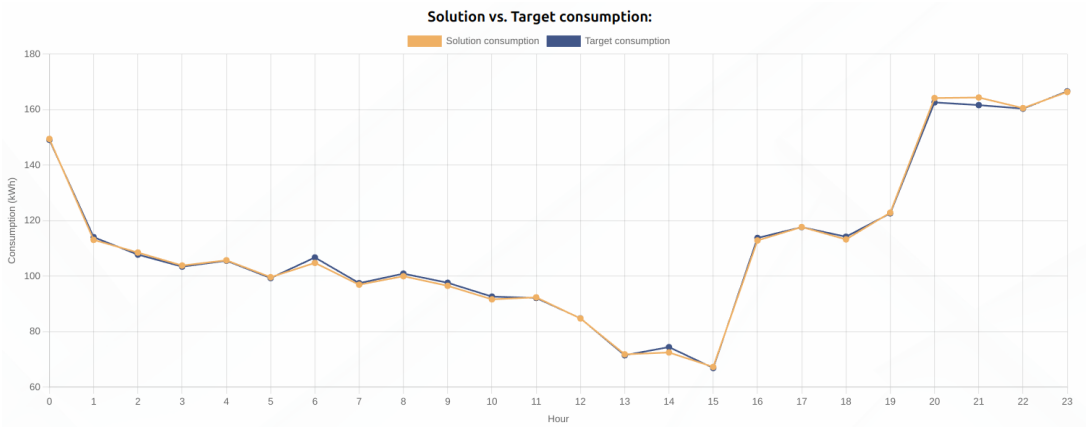


Figure 6.38: Approximation of the target curve of the solution given by the Bi-Level optimization

To have a fair comparison we need to look at the behavior of the algorithm when it is put under stress with an increased number of apartments. For this evaluation, the generated datasets will be used, together with the shifted target curve.

As expected, the running time of the Bi-Level optimization algorithm is substantially larger compared to the other algorithm, as shown in Figure 6.41. It is not of surprise as it runs the algorithm from the second level for each agent at each epoch.

When comparing the fitness of the approximation objective, the algorithm performs very closely to the improved HHO algorithm, better than the greedy implementation, see Figure 6.42. This fact doesn't come as a surprise either as in the lower level the improved HHO was utilized. The slight difference between the two fitness values might be attributed to the fact that the scheduling algorithm has a reduced set of devices to choose from, and might not find a suitable combination, without greatly impacting the comfort.

When looking at the comparison of the comfort fitness values of the analyzed algorithms, shown in Figure 6.43, it can be observed that the Bi-Level optimization design scores second here as well. It outperforms the improved HHO scheduling and gets close to the excellent results of the greedy method. Since devices belonging to apartments not included in the subset maintain their initial schedule, they operate within their comfort intervals. In this regard, the penalty associated with comfort is less, than in case all of them participate in the scheduling procedure.

To conclude this section, it can be said, that by its nature, the Bi-Level optimization design is expected to have a running time much higher than in the case of the

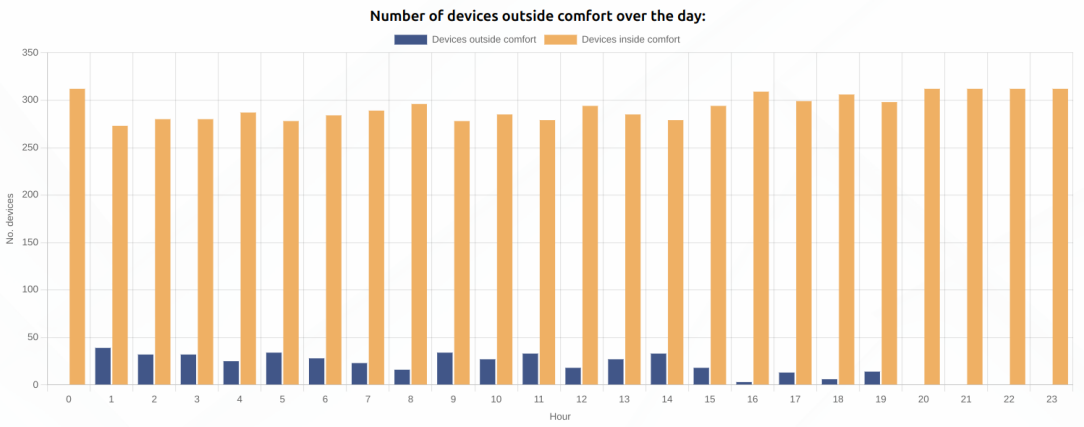


Figure 6.39: Number of devices outside and inside comfort at each hour

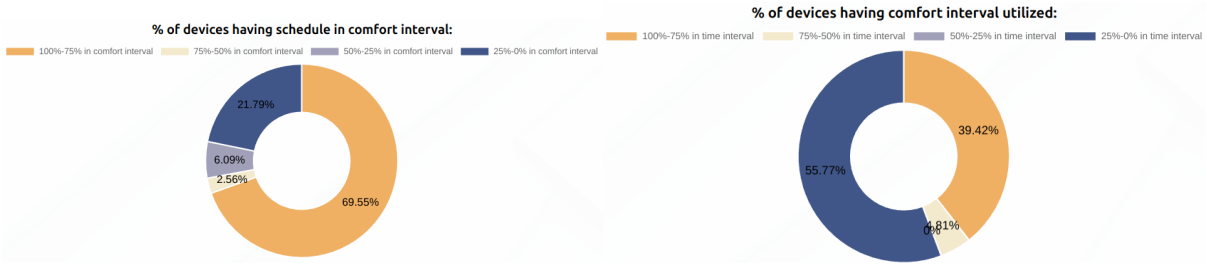


Figure 6.40: Evaluation of fulfilling the comfort objective by the solution given by the Bi-Level optimization

lower-level optimization. However, because the approximation objective together with the comfort objective can be maintained or even improved while including fewer apartments in the demand response program, the solution is feasible.

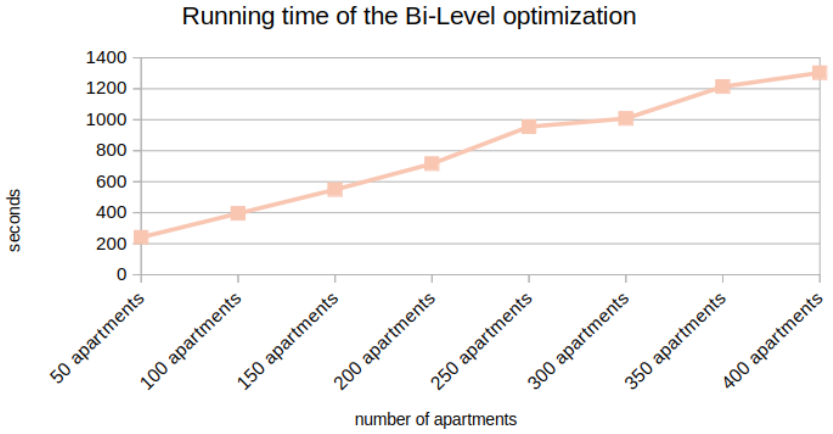


Figure 6.41: Running time comparisons of the Bi-Level optimization

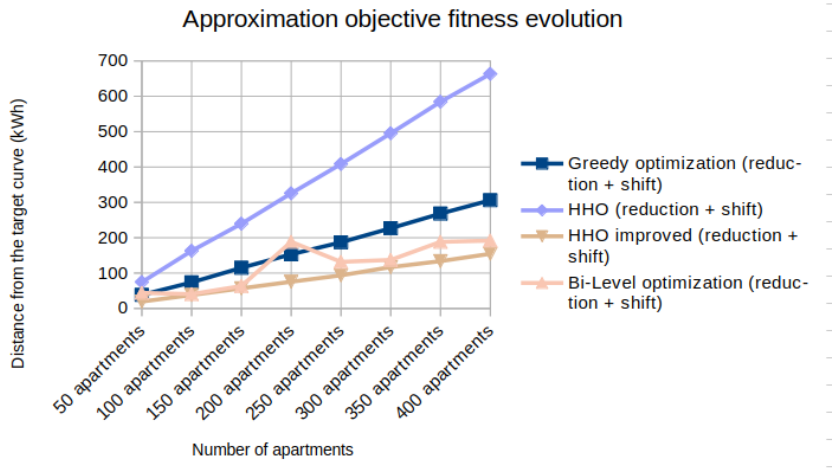


Figure 6.42: Approximation fitness comparisons of the Bi-Level optimization

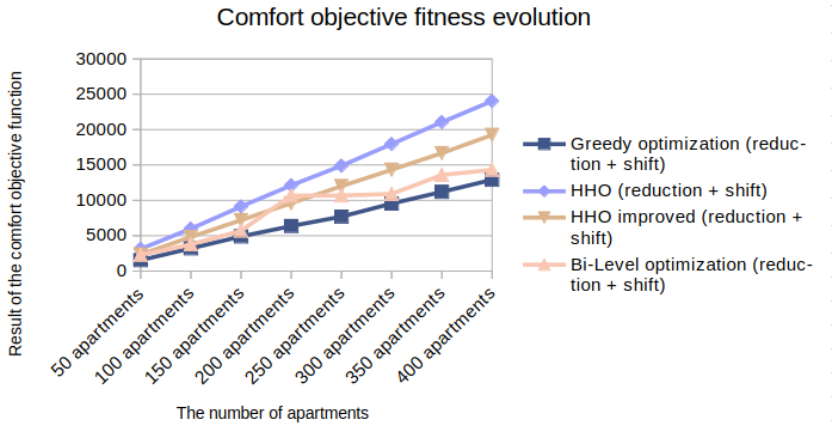


Figure 6.43: Comfort fitness comparisons of the Bi-Level optimization

Chapter 7. User's Manual

In this chapter, the installation guide of the application will be presented together with a guide on how to use the application.

7.1. Deployment

The minimum hardware requirements of the application are given by the libraries used, and the deployment server. In this regard, to deploy the application a minimum of 1 core processor is needed (but multiple cores are recommended to leverage the benefits of the parallel implementation of the algorithm), 1 GB of RAM is required, and at least 5 GB of storage is desirable. A network connection is mandatory.

The single software requirement is to have Docker installed, and the Docker daemon to be running. The application benefits of the virtualization provided by Docker. Thus, it downloads all the libraries it uses and runs on a lightweight Linux environment.

To deploy the application follow the steps above:

1. Create a network A docker network has to be created before running the container files. This network is used to communicate inside the docker environment between containers. From outside, this network is not visible, it is mapped onto localhost.

   To create the network run the following command with elevated privileges, or run the *create-private-network.bat* file.

   ```
   docker network create --gateway 172.19.0.1 --subnet 172.19.0.0/24 licenta-network
   ```

   This instruction creates a sub-network in docker with the starting address 172.19.0.0 and with the mask 255.255.255.0. The gateway of the network is running at address 172.19.0.1.

   Restart the host machine, or reload the network interfaces so the change is visible.

2. Copy the source files into a directory

   The source files will be cloned from the repository into a specific folder.

3. Run the docker composer

   In the folder where the source files are, run the following docker command:

   ```
   docker compose -up
   ```

This command will create the containers by following the configurations present in the *docker-compose.yml* file.

After running this command, all the containers should be deployed. The front-end server is available at the address *http://172.19.0.30/*, or *http://localhost/*.

7.2. Usage guide

In Figure 7.1 the landing page is shown. The operator assigned from the energy distribution company can enter the dashboard shown in Figure 7.2 by pressing the *Get started* button. On this page, there are three sections: the upload section, the operations section, and the results section.

When the operator uploads a new dataset it has to choose a unique identifier of length 10 for it and to upload the required files. If the dataset code is taken, or the uploaded files are incorrect an error message is displayed in the right-bottom part of the page, as shown in Figure 7.3. In case some mandatory fields are missing the missing input fields will be highlighted as shown in Figure 7.4, or a notification is received from the server.

To start an operation, the user will select first the available datasets, and the type of the operation and it will upload the necessary files, if any. For computing the initial consumption no additional files are needed, to compare the initial consumption and the target curve the target curve has to be uploaded. For the rest of the operations, all files are needed. The configuration parameters that are contained in the configuration file are described in detail in Section 6.3.

The result of the operations will be displayed in the Result section, in the form of a table, chart, or downloadable file, depending on the operations initiated.

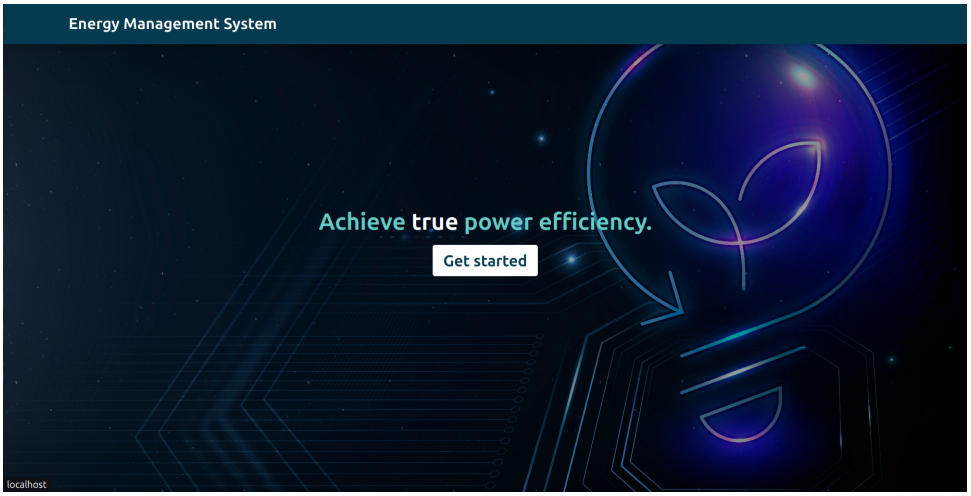


Figure 7.1: Landing page

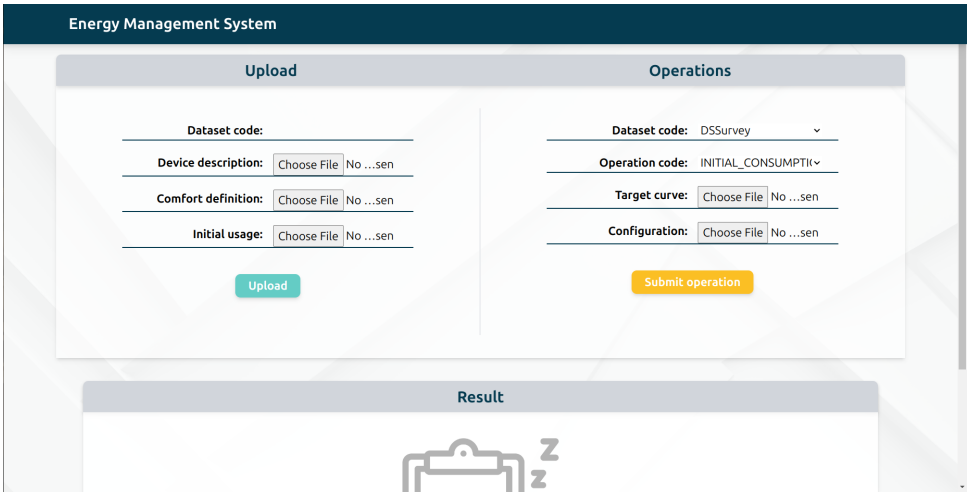


Figure 7.2: Dashboard page

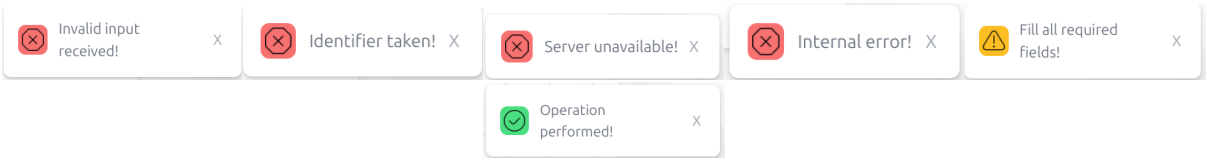


Figure 7.3: Notification messages

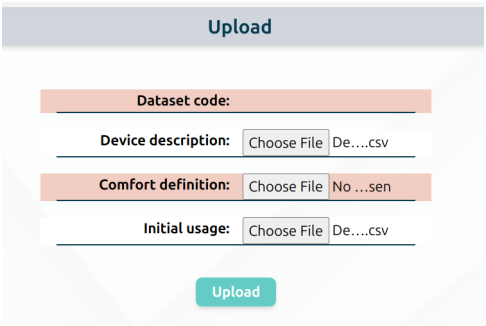


Figure 7.4: Missing fields

Chapter 8. Conclusions

The problem solved by this project is generated by peak hour electrical energy consumption that places intensive stress on the electrical grid. To tackle this problem, modern solutions include smart meters, which are unfortunately not an option in old residential buildings. This project aimed to develop a scheduling algorithm that makes a timetable for appliances from old residential buildings such that a target consumption curve provided by the electricity distribution company is approximated while reducing the impact on the comfort of tenants. The schedule had to be made for a day with a description of the devices from the building provided. An enhancement of the solution was the development of a Bi-Level architecture to identify the minimum subset of apartments that have to be involved in the program, besides achieving the approximation and comfort goals. As this domain is very well covered by the specialty literature, existing solutions were examined. As a result common design roadmaps were identified, as well as pitfalls of current solutions were pointed out. As part of the analysis and theoretical foundation, the most promising meta-heuristic algorithms were chosen to be used for each level, and the problem was mapped to suit them. Several design decisions were made all of which were evaluated and tested, and the best-performing decisions from each domain were selected as part of the final solution.

Moreover, an application was developed, to be used by the operator pointed by the electricity distribution company. This application contains four sub-modules; the front-end application, back-end application, scheduling module, and database server. All of these are deployed in Docker containers.

8.1. Contributions

The main contributions to the field are: making a representation of the problem to be solved by the meta-heuristic algorithm, developing a Bi-level optimization approach for the problem, and improving its performance. Below all contributions are detailed:

- **Review and analysis of state-of-the-art solutions for load scheduling**

  A comprehensive review was made of the available solutions in the domain of load scheduling. The strengths and pitfalls of each strategy were exposed, and analysed compared to the methods proposed by this thesis.

- **Defining the scheduling problem so that it can be solved by the Harris Hawks Optimization Algorithm**

  A well-thought representation of the problem was developed so that the optimization problem to be able to handle it. The most important details, that were not seen in the specialty study and included in this thesis, were the normalization of the two

objective values, the minimum usage hours constraint besides the non-deferrable constraint, and the comfort evaluation function employed.

- **Correcting the shortcomings of the HHO algorithm**

  It was revealed that the HHO algorithm cannot handle the total dimension of the problem, thus an improvement was proposed which reduces the number of parameters optimized at once. This improvement reduces the running time by a factor of 50 and considerably reduces the fitness of the two objectives as well.

- **Designing a Bi-Level optimization architecture to solve the multi-objective problem**

  The architecture proposed which incorporates the additional requirement to choose a minimum subset of apartments from the residential building that will be included in the demand response program is structured on two levels: on the first level a Genetic Algorithm selects the subset of apartments, that is passed to the improved Harris Hawks optimization algorithm which makes a schedule. The result from the lower level is then recombined with the non-included apartments to compute the fitness values.

- **Tuning the parameters to obtain the best performance**

  All the configuration parameters were analyzed and fine-tuned to achieve the best performances, and not to waste computational resources.

- **Developing a greedy algorithm to have terms of comparison**

  To have terms of comparison a greedy algorithm was developed to solve the same problem.

- **Developing a visual interface for accessing functionalities offered by the scheduling module**

  The functionalities of the application can be accessed via a user interface designed to be easy to use by an operator. The scheduling module was included in a back-end application to maintain connectivity via API calls with interfaced systems.

8.2. Result analysis

The scheduling algorithm was tested on two types of datasets: one gathered using a survey filled out by 39 students about their usage patterns, and comfort intervals, and a set of data generated using a proprietary script that takes into account multiple consumption profiles.

The results of the lower-level optimization were analyzed by comparing three implementation variants: the base version of the optimization algorithm, the improved version, and the greedy algorithm. While the base version yielded the worst results in all the evaluated fields: running time, approximation fitness, and comfort fitness, the other two competed very closely. The improved version of the Harris Hawks Optimization algorithm had a greater running time, compared to the greedy algorithm, but it managed to balance better the two objectives, thus having better performance for the approximation objective when the optimization placed a greater accent on it. The greedy algorithm

was unable to provide consistent fitness regardless of the target curve, which was another pitfall of it.

As a final note on the implementation versions, for the problem at hand, the improved version of the Harris Hawks optimization algorithm is the most suitable. It handles very well the multiple objectives, and it is consistent in terms of results, independently of the target curve given. It also converges to a solution in a reasonable time, despite being slower than the greedy method.

After examining the Bi-Level implementation, it can be stated that it is a feasible solution. While this design increases running time significantly, but still in an acceptable range, it yields the same results for the approximation objective, while significantly reducing the impact on comfort due to the smaller number of apartments included. This way not all tenants will have to change their consumption patterns, resulting in better user acceptance and satisfaction. For the examined dataset it was able to maintain the same distance from the target curve over a day, by including only around half of the apartments from the building, thus halving the comfort penalty.

8.3. Further Development

Despite the many achievements, there is still plenty of room for improvement. Each design decision can be extended or further refined to meet higher standards.

In the bibliographical research, it was discovered, that in the literature more types of devices are considered, thus as a future development, the solution can be extended with more constraints. To further enhance the representation, each apartment could have distinct comfort trade-off values to represent more accurately the wide range of preferences of tenants. To improve the performance of the optimization algorithm a more detailed analysis could be conducted on what caused exactly the poor performance of the base version, and how it could be solved. Another possibility would be to extend the objectives of the problem with a cost reduction target, which goes hand-in-hand with the target curve approximation, but provides a stronger incentive for tenants.

On the other side, the web application can be refined by adding more user types. For instance, tenants and building administrators could be given access to the application as well to upload the details of their devices, and to provide the platform with their comfort preferences. This way the operator would not have to gather their data outside the application.

Bibliography

[1] A. Shewale, A. Mokhade, N. Funde, and N. D. Bokde, "An overview of demand response in smart grid and optimization techniques for efficient residential appliance scheduling problem," *Energies*, 2020.

[2] C. Chen, "Demand response: An enabling technology to achieve energy efficiency in a smart grid," *Application of Smart Grid Technologies*, pp. 143–171, 2018.

[3] A. Power, "California leans hard into demand response programs to manage energy consumption," Jun. 2023. [Online]. Available: https://www.constructiondive.com/press-release/20230613-california-leans-hard-into-demand-response-programs-to-manage-energy-consum/

[4] D. H. Muhsen, H. T. Haider, Y. Al-Nidawi, and G. G. Shayea, "Operational scheduling of household appliances by using triple-objective optimization algorithm integrated with multi-criteria decision making," *Sustainability*, 2023.

[5] Z. Luo, J. Peng, and R. Yin, "Many-objective day-ahead optimal scheduling of residential flexible loads integrated with stochastic occupant behavior models," *Applied Energy*, 2023.

[6] A. Shewale, A. Mokhade, A. Lipare, and N. D. Bokde, "Efficient techniques for residential appliances scheduling in smart homes for energy management using multiple knapsack problem," *Arabian Journal for Science and Engineering (2024)*, 2023.

[7] Y. Wei, Q. Meng, F. Zhao, l. Yu, L. Zhang, and l. Jiang, "Direct load control-based optimal scheduling strategy for demand response of air-conditioning systems in rural building complex," *Building and Environment*, 2024.

[8] L. Guanghua, S. Ullah, G. Hafeez, and H. Alghamdi, "An application of heuristic optimization algorithm for demand response in smart grids with renewable energy," *AIMS Mathematics*, 2024.

[9] A. A. Dashtaki, M. Khaki, M. Zand, M. A. Nasab, P. Sanjeevikumar, T. Samavat, M. A. Nasab, and B. Khan, "A day ahead electrical appliance planning of residential units in a smart home network using its-bf algorithm," *International Transactions on Electrical Energy Systems*, 2022.

[10] Z. Chen, Y. Chen, R. He, J. Liu, M. Gao, and L. Zhang, "Multi-objective residential load scheduling approach for demand response in smart grid," *Sustainable Cities and Society*, 2021.

[11] Y. Liu, H. Li, J. Zhu, Y. Lin, and W. Lei, "Multi-objective optimal scheduling of household appliances for demand side management using a hybrid heuristic algorithm," *Energy*, 2022.

[12] M. Pothitou, R. F. Hanna, and K. Chalvatzis, "Ict entertainment appliances' impact on domestic electricity consumption," *Renewable and Sustainable Energy Reviews*, 2016.

[13] Y. Chen, P. Xu, J. Gu, F. Schmidt, and W. Li, "Measures to improve energy demand flexibility in buildings for demand response (dr): A review," *Energy & Buildings*, 2018.

[14] F. Pallonetto, M. D. Rosa, F. D'Ettorre, and D. P. Finn, "On the assessment and control optimisation of demand response programs in residential buildings," *Renewable and Sustainable Energy Reviews*, 2020.

[15] P. Cappers, C. Goldman, and D. Kathan, "Demand response in u.s. electricity markets: Empirical evidence," *Energy*, 2009.

[16] A. L. da Fonseca, K. M. Chvatal, and R. A. Fernandes, "Thermal comfort maintenance in demand response programs: A critical review," *Renewable and Sustainable Energy Reviews*, 2021.

[17] S. Bragagnolo, R. Schierloh, J. Vega, and J. Vaschetti, "Demand response strategy applied to planning the operation of an air conditioning system. application to a medical center," *Journal of Building Engineering*, 2022.

[18] S. Guenther, "What is pmv? what is ppd? the basics of thermal comfort," https://www.simscale.com/blog/what-is-pmv-ppd/, Dec. 2023.

[19] S. Leyffer, *Optimization: Applications, Algorithms, and Computation*, 2016.

[20] A. Sinha, P. Malo, and K. Deb, "A review on bilevel optimization: From classical to evolutionary approaches and applications," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 22, no. 2, pp. 276–295, Apr. 2018.

[21] J. F. Camacho-Vallejo, C. Corpus, and J. G. Villegas, "Metaheuristics for bilevel optimization: A comprehensive review," *Computers and Operations Research*, 2023.

[22] C. Pop, T. Cioara, and I. A. et al., "Review of bio-inspired optimization applications in renewable-powered smart grids: Emerging population-based metaheuristics," *Energy Reports*, 2022.

[23] X. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2010.

[24] F. Fausto, A. Reyna-Orta, E. Cuevas, A. G. Andrade, and M. Perez-Cisneros, "From ants to whales: metaheuristics for all tastes," *Artificial Intelligence Review (2020)*, 2019.

[25] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[26] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Generation Computer Systems*, 2019.

[27] "Axios library," https://axios-http.com/.

[28] "Tailwind library," https://tailwindcss.com/.

[29] "Chartjs library," https://www.chartjs.org/.

[30] V. Thieu, Nguyen, Mirjalili, and Seyedali, "Mealpy: An open-source library for latest meta-heuristic algorithms in python," *Journal of Systems Architecture*, 2023.

Appendix A. Relevant Code sections

```
{
    'status':"OPERATION_PERFORMED",
    'result':
    [
        {
            'type': "TABLE",
            'data':
            {
                'rows': [["Row1 Col1", "Row1 Col2"],
                         ["Row2 Col1", "Row2 Col2"]],
                'header': ["Header Col1", "Header Col2"],
                'title': "Title of the table"
            }
        },
        {
            'type': "CHART",
            'data':
                {
                    'label': [1,2,3,4,5],
                    'type': 'LINE',
                    'title': 'Chart Title',
                    'yAxisTitle': 'Title yAxis',
                    'xAxisTitle': 'Title xAxis',
                    'valuesArray':
                        [
                            {
                                'seriesName': 'Series 1',
                                'seriesValues': [8,6,5,3,4]
                            },
                            {
                                'seriesName': 'Series 2',
                                'seriesValues': [6,5,7,3,2]
                            }
                        ]
                }
        },
            {
                'type': "CHART",
                'data':
                    {
                        'label': [1,2,3,4,5],
                        'yAxisTitle': 'Title yAxis',
                        'xAxisTitle': 'Title xAxis',
                        'type': 'BAR',
                        'title': 'Chart title',
                        'valuesArray':
                            [
```

```
                                    {
                                    'seriesName': 'Series 1',
                                    'seriesValues': [8,6,5,3,4]
                                    },
                                    {
                                        'seriesName': 'Series 2',
                                        'seriesValues': [6,5,7,3,2]
                                    }
                                ]
                        }
                },
                {
                    'type': "CHART",
                    'data':
                        {
                            'label': ['1', '2', '3', '4'],
                            'type': 'PIE',
                            'title': 'Chart title',
                            'valuesArray': [
                                {
                                    "seriesName": "",
                                    "seriesValues": [20,30,10,40]
                                }
                            ]
                        }
                },

                {
                    'type': "INFO",
                    'data': "FILE"
                },
    ]
}
```

Listing A.1: Operation result message

```

import copy
import random

import numpy as np

# device name, consumption kWh, isDeferrable, minHours
device_info = np.array([
    ["light", 0.06, 1, 3],
    ["fridge", 0.5, 0, 24],
    ["laptop", 0.065, 1, 3],
    ["microwave", 0.083, 1, 2]
])

usage_patterns = np.array([
    #0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    [1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],  # at home in the morning
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]  # at home at evening
])
```

```
20
21 microwave_intervals = [
22     [[8, 10], [11, 12], [21, 23]],
23     [[7, 8], [18, 21]]
24 ]
25
26 lighting_need = np.array([1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1])
27
28 apartment_configuration = np.array([
29     [5, 1, 2, 1],  # apartment with 2 persons
30     [5, 1, 4, 1]  # apartment with 4 persons
31 ])
32
33 MAX_COMFORT_PENALTY = 2
34 comfort_penalty = [0, 1, 2]  # no penalty, small penalty, large penalty
35
36
37 def generate_dataset(no_apartments, type_ratio):
38     building_device_info = []
39     building_comfort = []
40     building_usage = []
41
42     ap_index = 0
43     for type_index, ratio in enumerate(type_ratio):
44         no_aps_in_type = int(no_apartments * ratio)
45
46         for ap in range(0, no_aps_in_type):
47             ap_info, ap_comfort, ap_usage = build_apartment(ap_index,
       type_index)
48             building_device_info.extend(ap_info)
49             building_comfort.extend(ap_comfort)
50             building_usage.extend(ap_usage)
51
52             ap_index += 1
53
54     return np.array(building_device_info), np.array(building_comfort),
       np.array(building_usage)
55
56 def build_apartment(apartment_id, type):
57     ap_device_info = []
58     ap_device_comfort = []
59     ap_device_usage = []
60
61     device_configuration = apartment_configuration[type]
62     device_index = 0
63     for device_type_index, device_count in enumerate(
       device_configuration):
64
65         for i in range(0, device_count):
66             id = [apartment_id, device_index]
67             info_row = copy.deepcopy(id)
68             info_row.extend(device_info[device_type_index])
69
70             ap_device_info.append(info_row)
71
72
73             comfort_row = copy.deepcopy(id)
```

```
            comfort_row_raw = build_comfort_row(device_type_index, type
    )
            comfort_row.extend(comfort_row_raw)
            ap_device_comfort.append(comfort_row)

            usage_row = copy.deepcopy(id)
            usage_row.extend(build_usage_row(comfort_row_raw,
    device_type_index))
            ap_device_usage.append(usage_row)

            device_index += 1

    return ap_device_info, ap_device_comfort, ap_device_usage


def build_usage_row(comfort_row, device_type):
    max_usage_row = np.where(np.array(comfort_row) == 0, 1, 0)

    hours = int(device_info[device_type][3]) + random.randrange(0, 4)
    return build_row(max_usage_row, hours, 1, 0)


def build_comfort_row(device_type, apartment_type):
    if device_type == 0:
        return build_comfort_row_lighting(apartment_type)

    if device_type == 1:
        return build_comfort_row_fridge(apartment_type)

    if device_type == 2:
        return build_comfort_row_laptop(apartment_type)

    if device_type == 3:
        return build_comfort_row_microwave(apartment_type)


def build_comfort_row_fridge(apartment_type):
    return [0 for _ in range(0, 24)]


def build_comfort_row_lighting(apartment_type):
    possible_hours = np.bitwise_and(lighting_need, usage_patterns[
    apartment_type])
    min_hours = int(device_info[0][3])

    hours = min_hours + random.randrange(1, 4)

    return build_row(possible_hours, hours, comfort_penalty[0],
    comfort_penalty[MAX_COMFORT_PENALTY])


def build_comfort_row_laptop(apartment_type):
    hours = int(device_info[2][3]) + random.randrange(1, 4)
    return build_row(usage_patterns[apartment_type], hours,
    comfort_penalty[0], comfort_penalty[MAX_COMFORT_PENALTY])


def build_comfort_row_microwave(apartment_type):
```

```
    available_hour_index = [random.randrange(start, end) for start, end in microwave_intervals[apartment_type]]
    available_hours = [comfort_penalty[0] if np.isin(index, available_hour_index) else comfort_penalty[MAX_COMFORT_PENALTY] for index in range(0, 24)]
    return available_hours

def build_row(hourly_availability, no_hours_to_choose, value_if_chosen, value_if_not_chosen):
    possible_hours_index = np.where(hourly_availability == 1)[0]

    if no_hours_to_choose > len(possible_hours_index):
        no_hours_to_choose = len(possible_hours_index)

    hours_active_index = np.random.choice(possible_hours_index, replace=False, size=no_hours_to_choose)
    return [value_if_chosen if np.isin(index, hours_active_index) else value_if_not_chosen for index in range(0, 24)]
```

Listing A.2: Test dataset generation script

```

solve(constraint_matrix, comfort_matrix, consumption_vector, initial_solution)
Begin
    solution = copy(initial_solution)

    translation_table = make_translation_table(consumption_vector)
    translation_table = remove_non_deferrable_devices(translation_table, constraint_matrix)

    foreach hour in hours_of_a_day do
        model_for_hour = get_model_for_hour(translation_table, comfort_matrix, consumption_vector, solution, hour)

        gap = compute_gap(solution, target_curve)

        if gap == 0 then
            continue
        end if

        model_for_hour = remove_greater_consumption(model_for_hour, abs(gap))

        if gap < 0 then
            model_for_hour = remove_not_scheduled(model_for_hour)
            model_for_hour = sort_consumption_large_first(model_for_hour)
            model_for_hour = sort_not_in_comfort_first(model_for_hour)
            remove_devices(translation_table, model_for_hour, gap, hour, solution)
        end if

        if gap > 0 then
            model_for_hour = remove_scheduled(model_for_hour)
            model_for_hour = sort_consumption_large_first(model_for_hour)
            model_for_hour = sort_in_comfort_first(model_for_hour)
```

```
31              add_devices(translation_table, model_for_hour, gap, hour,
    solution)
32          end if
33      end foreach
34      solution = apply_min_usage_constraint(constraint_matrix,
    comfort_matrix, solution)
35      return solution
36  End
37
38  remove_devices(translation_table, model gap, hour, solution)
39  Begin
40      device_index
41      while (gap < 0 and device_index < length(model)) do
42          consumption = get_consumption_for_device(model, device_index)
43          if consumption < abs(gap)then
44              unschedule(device_index, hour, solution)
45              gap += consumption
46          end if
47          device_index++
48      end while
49  End
50
51  add_devices(translation_table, model gap, hour, solution)
52  Begin
53      device_index
54      while (gap > 0 and device_index < length(model)) do
55          consumption = get_consumption_for_device(model, device_index)
56          if consumption < gap then
57              schedule(device_index, hour, solution)
58              gap -= consumption
59          end if
60          device_index++
61      end while
62  End
63
64  apply_min_usage_constraint(constraint_matrix, comfort_matrix, solution)
65  Begin
66      index = 0
67      foreach usage_row in solution do
68          no_used_hours = get_usage(usage_row)
69          min_usage_hours = get_min_usage_hours(constraint_matrix[index])
70          if no_used_hours >= min_usage_hours then
71              continue
72          end if
73
74          available_hours = get_not_used_hours_within_comfort(
    comfort_matrix[index], usage_row)
75          hours_needed = min_usage_hours - no_used_hours
76          chosen_hours = choose_numbers_from_array(available_hours,
    hours_needed)
77
78          solution = schedule_chose_hours(index, chosen_hours, solution)
79          index++
80      end foreach
81      return solution
82  End
```

Listing A.3: Pseudocode of the greedy scheduling algorithm