



university of  
 groningen

faculty of mathematics  
and natural sciences

# A software system to manage the domestic energy demand at the neighbourhood level

Master's thesis

August 2015

Student: Jan-Paul Eikelenboom

Primary supervisor: Prof.dr.ir Marco Aiello

Secondary supervisor: Prof.dr.ir Paris Avgeriou



## ABSTRACT

---

Energy plays an important role in the contemporary households, because without energy most appliances within the household will not work. The production of energy mostly comes from finite fossil fuels which contributes to global warming. In order to keep a functioning society, well managed energy consumption is important.

In contrary to other works as they focus more on the individual households, this work attempts to provide a system that minimizes the peaks in energy consumption by analysing the energy consumption of multiple households in a neighbourhood. An *energy consumption vector* defines the energy consumption of the appliances in the neighbourhood. The set of tasks for the appliances is created by the *energy consumption vector* in combination with the Max Steady-State technique. After task creation, the set with policies for each appliance is made. Based on the tasks and policies sets, the optimal schedule is created which minimizes the Peak to Average Ratio. The scheduling algorithm uses the Breadth-First Search algorithm in combination with a Priority Queue.

The results of the scheduling algorithm on the three data sets shows an average improvement of 64,9% in costs over the original input. In one case, the improvement over a data set was 3,3%. Upon inspection of this data set, we concluded that one appliance caused the peak. In the other data sets, where a higher percentage of change was found, multiple appliances caused the peak. We conclude that the system is useful for minimizing peaks within households in a neighbourhood, as long as the peak is introduced by multiple appliances. While at the same time contributing to the slow down of global warming.



## ACKNOWLEDGMENTS

---

I would like to express my gratitude to my supervisor Prof.dr.ir. Marco Aiello for the guidance, remarks, and ideas during the writing of my thesis. Furthermore, I would like to thank MSc. Marko Milovanovic for giving me the opportunity to work on this subject and for providing ideas and comments throughout the process. I also would like to thank Elmer Jansema and Marleen Eikelenboom for sharing their precious time to proofread my thesis and for giving comments. Finally, I would like to thank my family and friends who have supported me throughout my study and during the process of writing my thesis.



## CONTENTS

---

i	A SOFTWARE SYSTEM TO MANAGE THE DOMESTIC ENERGY DEMAND AT THE NEIGHBOURHOOD LEVEL	1
1	INTRODUCTION	3
1.1	Research problem	4
1.2	Thesis contribution	5
1.3	Thesis outline	5
2	SMART ENVIRONMENTS AND LOAD SHAVING	7
2.1	Smart Environments in households	7
2.1.1	GreenerBuildings	7
2.1.2	enerGQ I-CARE	8
2.1.3	Net2Grid Smartbridge	8
2.2	Load management and load shifting	9
ii	CONCEPT AND IMPLEMENTATION	11
3	CONCEPT	13
3.1	Gathering historical measurements	13
3.2	Tasks and policies	14
3.2.1	Tasks	14
3.2.2	Policies	15
3.3	Creating a schedule	17
3.3.1	Terminates	18
3.3.2	Completeness	18
3.3.3	Complexity	19
4	IMPLEMENTATION	23
4.1	System Overview	23
4.2	Development tools	24
4.3	Scraper	24
4.3.1	The spider	25
4.3.2	The pipeline	25
4.4	Web application	27
4.4.1	Peakfinder application	27
4.4.2	Scheduler application	28
4.4.3	REST application programming interface (API)	30
4.5	Database	31
iii	SIMULATION AND VALIDATION	33
5	RESULTS	35
5.1	Setup	35
5.1.1	The data	35
5.1.2	The metrics	35
5.2	Running	36
5.2.1	Generating tasks and policies	36

5.2.2	Generating schedules . . . . .	40
5.3	Analysis . . . . .	46
5.3.1	Task and Policies . . . . .	46
5.3.2	Before and after scheduling . . . . .	46
5.4	Discussion . . . . .	47
6	CONCLUSION . . . . .	51
6.1	Future work . . . . .	52
iv	APPENDIX . . . . .	53
A	GRAPHICAL USER INTERFACE OF THE SYSTEM . . . . .	55
A.1	Peakfinder application . . . . .	55
A.2	Scheduler application . . . . .	56
	BIBLIOGRAPHY . . . . .	59



## LIST OF FIGURES

Figure 1	The web portal of enerGQ which displays the energy consumption. . . . .	8
Figure 2	The Net2Grid Smartbridge overview . . . . .	9
Figure 3	Components overview of the Cooperative Load Scheduling System (CLSS). . . . .	23
Figure 4	Components overview of Scrapy [23]. . . . .	24
Figure 5	Dataflow diagram of the scraper component. . . . .	26
Figure 6	Screenshot from the browsable API [4]. . . . .	31
Figure 7	Different data model types [14]. . . . .	32
Figure 8	Energy consumption of the appliances in data set A. . . . .	37
Figure 9	Energy consumption of the appliances in data set B. . . . .	38
Figure 10	Energy consumption of the appliances in data set C. . . . .	39
Figure 11	Energy consumption and cost charts for data set A. . . . .	40
Figure 12	Energy consumption and cost charts for data set B. . . . .	41
Figure 13	Energy consumption and cost charts for data set C. . . . .	42
Figure 14	Energy consumption for data set A with and without schedule. . . . .	43
Figure 15	Energy consumption for data set B with and without schedule. . . . .	44
Figure 16	Energy consumption for data set C with and without schedule. . . . .	45
Figure 17	Cumulative percentage of change according to Equation 9 after scheduling for data set A. . . . .	48
Figure 18	Cumulative percentage of change according to Equation 9 after scheduling for data set B. . . . .	48
Figure 19	Cumulative percentage of change according to Equation 9 after scheduling for data set C. . . . .	48
Figure 20	Screenshot showing a list of households. . . . .	55
Figure 21	Screenshot showing a list of households for a specific neighbourhood. . . . .	55
Figure 22	Screenshot showing an overview of the household. . . . .	56
Figure 23	Screenshot showing the index page of the Cooperative Load Scheduling System (CLSS). . . . .	56
Figure 24	Screenshot showing the login page. . . . .	56

Figure 25	Screenshot showing a list with neighbourhoods.	57
Figure 26	Screenshot showing the details of a neighbourhood. . . . .	57

## LIST OF TABLES

---

Table 1	Summary of the different policies their associated appliances. . . . .	16
Table 2	Appliances available in each of the three data sets. . . . .	36
Table 3	Policies assigned in each of the three data sets	46
Table 4	Energy used by appliances that can be scheduled active between timeslot 11 and 15, according to <a href="#">Figure 8</a> . . . . .	49
Table 5	Energy used by appliances that can be scheduled with active between timeslot 11 and 16, according to <a href="#">Figure 9</a> . . . . .	50
Table 6	Energy used by appliances that can be scheduled with active between timeslot 17 and 22, according to <a href="#">Figure 10</a> . . . . .	50



## ACRONYMS

---

API	application programming interface
BFS	Breadth-First Search
CLSS	Cooperative Load Scheduling System
CSV	Comma Separated Values
DE	Delay
FIFO	First In First Out
GUI	graphical user interface
HTML	Hypertext Markup Language
HTTPS	Hypertext Transfer Protocol Secure
IDE	integrated development environment
IDD	Instantaneous Demand Delivered
IDR	Instantaneous Demand Received
JSON	JavaScript Object Notation
LED	light emitting diode
MAC	media access control
RFC	Request for Comments
RP	Repeating
SG	Smart Grid
SI	Single
ST	Strict
SQL	Structured Query Language
PAR	Peak to Average Ratio
PR	Profile
PQ	Priority Queue
URL	Uniform Resource Locator



## Part I

# A SOFTWARE SYSTEM TO MANAGE THE DOMESTIC ENERGY DEMAND AT THE NEIGHBOURHOOD LEVEL





## INTRODUCTION

---

These days energy plays an important role in most households. Each household has a multitude of electric and electronic devices helping them in their daily tasks, such as refrigerators for storing food, computers and mobile telephones for communication and washing machines for cleaning the laundry. Although energy is a normal commodity in most households, it needs to be produced. Like most production processes, making energy comes with a cost.

The majority of energy is still generated by burning fossil fuels, that in return produces CO<sub>2</sub> as a by-product. CO<sub>2</sub> gasses are believed to be the main cause for the global warming [15]. In addition the reserves of fossil fuels are not infinite and eventually these will be depleted. Another but smaller part of the energy production comes from renewable sources, such as solar, hydro and wind energy. Even though these methods are environmental friendlier than fossil fuels, the investments required are high and the amount of energy produced is low in comparison to fossil fuels. An even smaller part of the energy production is provided by nuclear plants. This method involves a complex process and therefore comes with great safety risks, as proven by major accidents in Fukushima and Chernobyl. Because of all these risks, fossil fuels will remain to be the primary resource for energy in the coming years.

Since it is difficult to change the energy source we have to look into another solution, in other words consuming less energy instead. According to the International Energy Agency it is possible to keep the CO<sub>2</sub> levels under control until 2017, limiting the global warming with 2°C. However, without taking action the allowable CO<sub>2</sub> levels will be locked-in by the existing energy infrastructure at that time. Cutting the energy consumption growth in half by investing in energy saving solutions, postpones the lock-in to 2022 [2].

One method of saving energy is replacing the devices by one that requires less energy when it is used. For example, replacing halogen with more energy efficient light emitting diode (LED) lamps. Another method is to improve the energy consumption of consumers, by for instance giving them tools to analyse their energy consumption which allow them to find areas in which they can improve their energy consumption.

In our research we focus on the last method, so improving the energy consumption of consumers, by using technology to provide feed-

back to the user about his or her energy usage. The feedback consists of automatically generated optimal schedules for home appliances within a neighbourhood.

### 1.1 RESEARCH PROBLEM

Smart environments form an important part of the future energy infrastructure. Because they give households insight into their energy usage. Mark Weiser describes a smart environment as

"a physical world that is richly and invisibly interwoven with sensors, actuators, displays and computational elements, embedded seamlessly in everyday objects of our lives, and connected through a continuous network" [25].

This concept is already being applied in some research projects, such as the GreenerBuildings project [1], to reduce energy consumption.

We focus on collaboration between multiple households in a neighbourhood to schedule their energy consumption. Appliances in households can be divided into three categories namely: major appliances, small appliances and consumer electronics. Major appliances are difficult to move, such as refrigerators and washing machines. Small appliances are easy to move and have a low energy consumption, for example, toasters and coffee machines. Finally, consumer electronics are appliances such as televisions and personal computers, in general related to entertainment, communication and office activities. From a scheduling point of view is the first category the most interesting, given that these appliances have a high energy consumption and they can be scheduled freely. However, this category is responsible for only a small part of the energy consumption for most households. Therefore we do not look at a single households but a complete neighbourhood for reducing peaks, because a peak will have a great impact on the energy consumption. We reduce peaks by creating an algorithm and a web application that provides feedback for each household thus the consumer himself can prevent peaks to occur in the total energy consumption of the neighbourhood. Therefore our main research question is:

**How do we automatically provide households in a neighbourhood with a schedule to avoid power peaks?**

We will answer the research questions by asking the following sub questions:

1. What is the current state of the art regarding energy feedback and energy management in respect to households?

2. How to determine when an appliance is active based on the energy usage gathered by smart meters?
3. How to find the near-optimal combination of tasks in order to minimize peaks in the combined energy profile?
4. What kind of system is needed to automatically minimize peaks in a neighbourhood based on historical data?

## 1.2 THESIS CONTRIBUTION

In this thesis we propose a system for minimizing peaks in the energy consumption of a residential neighbourhood. The system analyses energy load profiles of the different categories of appliances during the day by searching for active periods. Once completed the active periods are assigned to a policy. Appliances may have more than one policy assigned to them, depending on the type of appliance. Finally the policies are used by the scheduling algorithm as a basis for creating new energy load profiles in which the peaks are minimized.

We describe the algorithms used for scheduling and provide a working system that can be accessed via a graphical user interface (GUI) and application programming interface (API). The proposed solution is tested for three data sets, where each data set contains energy load profiles of randomly chosen households on a random day. After analysing the results, we see a decrease in peaks for all three data sets. The system could therefore be an useful tool for managing the energy demand in a residential neighbourhood.

## 1.3 THESIS OUTLINE

The thesis consists of the following chapters;

CHAPTER 2: we explain the current state of the art in the field of household energy management tools. Furthermore, we look into scheduling appliances inside a household.

CHAPTER 3: we present the concept of our algorithm for minimizing peaks in neighbourhoods. Secondly the structure of the data set is presented in combination with algorithms that transform the initial measurement in tasks. Thirdly we look how the policies are assigned to the created tasks. Finally, we present the scheduling algorithm and give a short discussion about its complexity.

CHAPTER 4: we present the design and implementation of the concept created in [Chapter 3](#). Followed by an architecture overview is given and a description of the technologies used.

CHAPTER 5: we present the results of running the algorithm on three data sets. First we explain the data sets followed by the results and a small discussion.

CHAPTER 6: we give the conclusion where we reflect on the work presented in the thesis. Followed by a further work section where we give ideas in the area of appliance scheduling and energy awareness for households.

## SMART ENVIRONMENTS AND LOAD SHAVING

---

In the late 1980s Mark Weiser [25] described the smart environment as

"a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network."

Nowadays technology has turned the vision of Mark Weiser into a reality. Therefore making it is now possible for consumers to turn their house into a smart environment.

### 2.1 SMART ENVIRONMENTS IN HOUSEHOLDS

One of the areas where smart environments play an important role is the energy awareness field. This because smart environments enable either the users or the buildings to change their energy usage based on the data provided by the environment. The following projects have in common that they enhance the data of the smart environment to influence behaviour.

#### 2.1.1 *GreenerBuildings*

The goal of GreenerBuildings [1] is to make buildings aware of their energy usage and adapt their energy consumption accordingly. This project of the GreenerBuildings project has developed a framework which uses sensors to define human activity. Based on these activities, an ubiquitous layer of the system will adapt smart objects. These smart objects are light switches, plug-in power meters, blinds motor, and controllers.

A second layer on top of the ubiquitous layer allows the user to interact with the system and to receive feedback from the framework. The focus of the project is mainly on controlling the energy on a local level, such as rooms or offices, because global control of a large building tends to be rather complex. However, reducing energy usage on a local level will have an impact on a global level, so a local focus is an a worthwhile endeavour.

### 2.1.2 enerGQ I-CARE

The goal of enerGQ is to make organizations and households aware of their energy usage. One of the solutions they offer households is I-CARE [5]. I-CARE gives households a web portal which shows their energy usages. The energy usage data is retrieved from the smart meter installed in their household. Like with most other systems such as, Net2Grid and others, households are able to review their energy usages in different formats.

Nonetheless this solution is different from others, because it also tells households if they are saving energy by comparing the current energy usage with historical data. With this comparison time, day of the week, and weather conditions have are taken into account. In the interface color codes are used as indication if the energy usage differs or is equal to similar circumstances at a certain time (see Figure 1).



Figure 1: The web portal of enerGQ which displays the energy consumption.

### 2.1.3 Net2Grid Smartbridge

Net2Grid Smartbridge [21] is a system which give households insight into their energy usage. The main component of this system is the Smartbridge which is used as a hub for retrieving and displaying energy used by a household. In addition, the Net2Grid relies on a smart meter in combination with multiple Smart Plugs to retrieve the energy consumption (see Figure 2). When the Smartbridge has gathered data it becomes available for users by either an application on their smartphone or using the myNET2GRID portal.

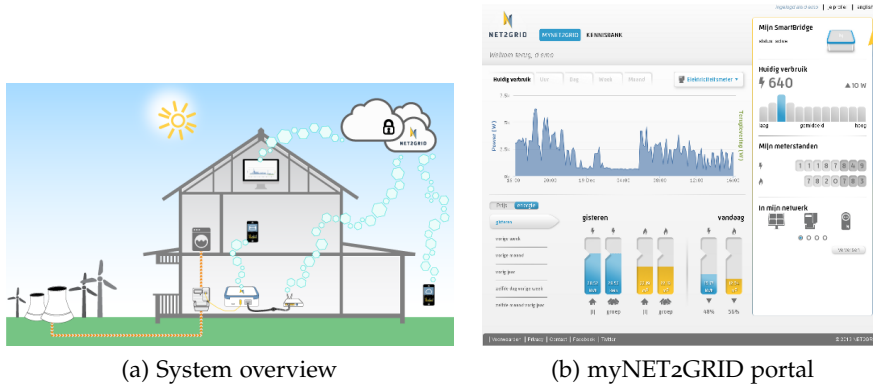


Figure 2: The Net2Grid Smartbridge overview

Apart from giving insight into the energy usage, the system allows users to set goals. An example of these goals is setting a maximum for the energy consumption during a day or month. When the goals are reached, the system will inform the user about their achievement which is giving the user extra motivation to reduce their energy consumption.

## 2.2 LOAD MANAGEMENT AND LOAD SHIFTING

Managing energy consumption is important for both consumers and producer, since the energy can not be stored, unlike other commodities, such as water, petrol, or gas, which can be stored. By using demand side management methods, energy producers are able to influence the consumers of energy, like for example households. One technique used by the energy producers is modifying the power curve [3], in which they influencing the energy demand of the consumers causing the load shape to change. For example, companies charge higher rates during peak hours than in off-peak hours, hoping the consumers will shift their energy consumption into the off-peak hours.

Instead of letting consumers plan their own appliances, research has looked into using computers for finding the optimal schedule. Research by Mangiatordi et al. [18] describes a method for reducing the Peak to Average Ratio (PAR) with a swarm intelligence system. This system calculates an average energy usage with the cost function of the algorithm by using the energy profiles of the devices inside the households. Another system proposed by Georgievski et al [8] schedules devices according to policies. This system is build for offices connected to the smart grid. Based on the type of devices and typical usage the researcher have created a set of policies. For example, a microwave or a laptop has a policy that forces the scheduling algorithm to plan the activities for these devices when the user requires the device, reducing the impact on the user. The scheduling algorithm uses

a Priority Queue (PQ) with Breadth-First Search (BFS) optimization for finding the optimal schedule.

Although there are many solutions in the field of energy awareness or load management, many focus on a single household. The novelty of our proposed work is that we look at a neighbourhood instead. Our algorithm can also be applied with minimal user input except with an energy profile for each appliance in the neighbourhood.



## Part II

### CONCEPT AND IMPLEMENTATION



## CONCEPT

---

The Cooperative Load Scheduling System (CLSS) is a tool for households in a neighbourhood to reduce peaks in energy demand. The system delivers an energy usage schedule for all the appliance inside a neighbourhood based on historical measurements. These measurements are gathered on a daily basis for each of the appliances by the CLSS. Based on this information an average energy consumption and set of tasks performed by the appliances are obtained. Once completed, the scheduling algorithm finds the optimal or near-optimal schedule. The CLSS gives the neighbourhood insight into their energy consumption and provides an incentive to adapt their usage of the appliances according to the optimal schedule. The scheduling procedure consists of the following steps;

- Gather measurements of appliances
- Create tasks and policies
- Find the optimal schedule for the given tasks and policies.

### 3.1 GATHERING HISTORICAL MEASUREMENTS

Creating a schedule is difficult without knowing when the appliances are active. So first of all, the CLSS have to create a load profile for the neighbourhood where the schedule is intended for. Each household  $h$ , which is part of the neighbourhood, is fitted with four smart meters that register the energy consumption of the appliance  $a$ , leading to the following definitions;

$H$ : Set of households which belong to the neighbourhood

$A_h$ : Set of appliances available for scheduling inside household  $h \in H$ .

Based on the two sets defined above we define an *energy consumption vector* [20]

$$x_{h,a} = [x_{h,a}^1, \dots, x_{h,a}^N] \quad (1)$$

where  $N$  is the total amount of hours a day, so 24. Equation 1 gives the hourly energy consumption of an appliance  $a$  of household  $h$ . So, the energy consumption of one appliance during the day is equal to  $x_{h,a}^n$  where  $n \in \{1, \dots, N\}$ . Since Equation 1 gives the energy consumption of each appliance, the following equations give the total

energy consumption of all appliances  $A_h$  inside household  $h$  or a neighbourhood  $H$  during hour  $n$

$$l_h^n = \sum_{a \in A_h} x_{h,a} \quad n \in N \quad (2)$$

$$l'_n = \sum_{h \in H} l_h^n \quad n \in N \quad (3)$$

where  $N = 24$ .

As explained in the introduction of [Chapter 3](#), the system works at neighbourhood level. The households in a neighbourhood contain multiple appliances. All of these appliances have corresponding energy consumption vectors as defined by [Equation 1](#). Combining these vectors results in the matrix below

$$M_{a,n} = \begin{pmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,n} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{a,1} & m_{a,2} & \cdots & m_{a,n} \end{pmatrix} \quad (4)$$

where the rows correspond to the  $x_{h,a}$  vectors of appliance  $a$  inside household  $h$  of neighbourhood  $H$  and  $n$  is the time slot in the range  $\{1, \dots, N\}$ . Now that we have matrix  $M$  our goal is to create the optimal schedule by  $M'$  in which the [PAR](#) is minimized for each column, similar to research by Mangiatordi et al. [\[18\]](#).

### 3.2 TASKS AND POLICIES

The matrix  $M$  created in [Section 3.1](#) provides load profiles for neighbourhoods. The rows in this matrix contain the load profile of the appliances equipped with smart meters. Active periods are reflected by an increase in energy consumptions, defined as tasks. These tasks are used by the [CLSS](#) as building blocks for the creation of  $M'$ . In this research we make the assumption that tasks have to be completed once they have started, other restrictions are enforced by policies.

#### 3.2.1 Tasks

A task is defined as vector  $T_j = (a_j, s_j, e_j, r_j)$ , where  $a_j$  is the appliance,  $s_j$  is the start slot,  $e_j$  is the end slot and  $r_j$  is the energy used. Tasks are based on matrix  $M$  and are discovered by using the Min-Max Steady-State technique described by Lee et al. [\[16\]](#). Given the

values from cell  $m_{a,n}$  and  $m_{a,n-1}$  in  $M$ , the similarity between both cells is estimated as

$$f(m_{a,n}, m_{a,n-1}) = \begin{cases} \text{stable, if } |\text{avg}(m_{a,n}) - \text{avg}(m_{a,n-1})| < \varepsilon \\ \text{change, else} \end{cases} \quad (5)$$

where  $\text{avg}(m_{a,n})$  and  $\text{avg}(m_{a,n-1})$  are the average usage for cell  $m_{a,n}$  and  $m_{a,n-1}$  in  $M$ , and  $\varepsilon$  is the threshold for the similarity, in our case three times the standard deviation of  $m_{a,n}$ .

When looking at the load profile of an appliance in  $M$  we define three states, namely; active, standby, and off. As result, each row in  $M$  can be divided in periods where the appliance is in one of the three states. A period always starts and ends with two consecutive cells marked as *change* by Equation 5. The cells between the start and end cell decide the state where the appliance is in. When the appliance is in the off state it is not consuming energy at all. In both the active and standby state the appliance is consuming energy. However, the assumption is made that the energy consumption is low and uniform in the standby state, while this is not the case in an active state. To find the energy consumption corresponding to the standby state we first find the minimal energy consumption, for which the cell is marked as *change* by Equation 5. If this value is found in more than one consecutive cell we define it as threshold  $S_a$  for the standby state, where  $a$  is the appliance. When the value is only found in a single cell the threshold  $S_a$  is set to 0. When the energy consumption is getting in a cell above  $S_a$ , the appliance is turning into the active state.

Periods where the appliance is in an active state are characterized as Tasks, where  $r_j$  is calculated with Equation 6.

$$r_j = \frac{\sum_{i=s_j}^{e_j} m_{a,j,i}}{e_j - s_j} \quad (6)$$

### 3.2.2 Policies

Policies gives the schedule algorithm rules to follow when a task is planned for an appliance (Georgievski et al. [8]). Such a planning can be made for example a freezer that runs for a fixed duration every hour or a computer which needs to be active between two and three o'clock. Policies have at least the  $P_{\text{begin}}$ ,  $P_{\text{end}}$  and a  $P_{\text{duration}}$  variable, where  $P_{\text{begin}}$  is the start timeslot,  $P_{\text{end}}$  is the end timeslot, and  $P_{\text{duration}}$  is equal to  $P_{\text{end}} - P_{\text{begin}}$ .

Our system defines six types of policies, as specified in Table 1 and described in the following list.

POLICY TYPE	LINKED APPLIANCE	DESCRIPTION
DELAY	Freezer	Appliance should be active for $P_{duration}$ , but may be delayed by $P_{delay}$ .
REPEAT	Any appliance	Appliance should be active for $P_{number\_of\_cycles}$ of a given $P_{cycle\_duration}$ .
SINGLE	Wash machine, Dishwasher	Appliance should be active for $P_{duration}$ .
STRICT	Television, Computer	Appliance should be active between $P_{begin}$ and $P_{end}$ .
PROFILE	Any appliance	Appliance is active during all hours of the day.
SLEEP	Any appliance	Appliance is inactive during the day.

Table 1: Summary of the different policies their associated appliances.

**DELAY:** Appliances that fall under the DELAY policy are scheduled with a delayed start with respect to the original starting time:  $P_{begin}$ . The delay is defined by  $P_{delay}$  and is either positive or negative. While a task may be delayed for the appliance, the execution time defined by  $P_{duration}$  is maintained.

**REPEAT:** Appliances like a refrigerator or freezer are likely to switch on and off periodically. These fluctuations lead to a number of tasks with the same duration during the day. The REPEAT policy groups these tasks together under one policy. Instead of the  $P_{duration}$  variable, two new variables are used namely;  $P_{cycle\_duration}$  and  $P_{nr\_of\_cycles}$ .  $P_{cycle\_duration}$  tells the schedule algorithm the duration of one cycle, in which the appliance is active. The number of repetitions is stored in  $P_{nr\_of\_cycles}$ , the schedule complies with the policy when the number of cycles equals the  $P_{nr\_of\_cycles}$ .

**SINGLE:** Many appliances inside households run just one task during the day. The starting time  $P_{begin}$  is not important for this policy, because only  $P_{duration}$  matters. Typical examples of such appliances are washing machines and dishwashers. Tasks with this policy can be scheduled freely during the day, as long as the  $P_{duration}$  is respected.

**STRICT:** This policy is used for appliances which are needed at a specific time. The tasks for such an appliance start at  $P_{begin}$  and end at  $P_{end}$ . Typical appliances for this policy are televisions and computers. These appliances directly respond to the user interaction and delaying them would cause inconvenience for the users.

**PROFILE:** There are appliances that consume energy throughout the entire day. Changing the begin  $P_{begin}$  or  $P_{end}$  will lead to an incomplete task, as the duration is less than  $P_{duration}$ . However, these appliances still influence the schedule, especially when their energy consumption is high. So, they are taken into account by the scheduling algorithm when a schedule is created.

**SLEEP:** Appliances that are inactive throughout the day are assigned to the SLEEP policy. This inactivity of the appliances indicating to the scheduling algorithm that these appliances have no need at all to be scheduled. Because of this reason, appliances with this policy are excluded from scheduling by the scheduling algorithm.

### 3.3 CREATING A SCHEDULE

Given the set of tasks vectors  $T$  and policies  $P$  we want to minimize the **PAR**. The problem is formalized by reducing the distance between the energy consumption on hour  $n$  where  $n \in \{1, \dots, 24\}$  and the average energy consumption of all tasks  $T_j \in T$ , similar to research by Mangiatordi et al. [18]. For comparing schedules, the cost of a schedule is determined by the following equation:

$$\text{Cost}(n) = (\text{AvgP} - \sum_{T_j \in T} r_j \cdot \delta_{jn})^2 \quad (7)$$

where  $\text{AvgP}$  is the average energy consumption of all tasks and

$$\delta_{jn} = \begin{cases} 1, & \text{if } s_j \leq n \leq e_j \\ 0, & \text{else} \end{cases} \quad (8)$$

Besides scheduling the tasks  $T$  into the available slots, we also need to comply with the policies  $P$ . Hence the optimal schedule minimizes the cost for each time slot  $n \in N$ , while at the same time it should satisfy the policies in  $P$ . To find the optimal schedule we use a **BFS** in combination with a **PQ** optimization algorithm [22].

**BFS** is an algorithm for exploring a graph. The algorithm starts with discovering the adjacent nodes of the initial node by adding them to a queue. Using a queue guarantees that the nodes are sorted in a First In First Out (**FIFO**) order, which ensures that nodes are removed from the queue in the order in which they are discovered. In the following iterations the algorithm removes a node from the queue and discovers its adjacent nodes. The algorithm terminates when the queue is empty.

Another use of the **BFS** algorithm is searching for a specific node in a graph. In this case, the algorithm terminates when the queue returns a node matching to the search criteria. However, this is the first node that matched the search criteria, not necessarily the optimal node. To find the optimal node, a small alteration to the **BFS** algorithm is needed.

Instead of using a queue for storing the discovered nodes, a **PQ** is used. By using a **PQ**, the node with the lowest cost is removed in every iteration. Discovered nodes will either increase the cost or cause it to remain the same. This means that with the **PQ**, the first node matching the search criteria is the optimal node.

In our case each of the nodes represent an incomplete schedule. We fill the PQ with solutions for the first time slot. In order to reduce the search space we make two adaptations to the BFS algorithm. First, we only add the solution with the lowest cost to the PQ during each iteration. The other solutions are only increasing the cost of the complete schedule. Hence, they would only increase the size of the PQ because they would never be removed. Second, we omit solutions that contradict with the policies in P. For example, when an appliance has a SINGLE policy it only needs to be scheduled for the duration of the task, so solutions that extend the task beyond the duration are not expanded. Appliances with a STRICT policy only need to be scheduled between  $P_{\text{begin}}$  and  $P_{\text{end}}$ . Consequently, there is no need for expansion of schedules that plan a task for an appliance outside the interval  $P_{\text{begin}}$  and  $P_{\text{end}}$ . The algorithm stops when a complete schedule is returned by the PQ.

### 3.3.1 Terminates

The algorithm explained in Section 3.3 is given in pseudo-code by [algorithm 1](#). [algorithm 1](#) will terminate when either the PQ is empty or a complete schedule is removed from the PQ. When an empty set of appliances A is given as an input, the algorithm will exit the while-loop, causing terminating the algorithm directly. If appliances in set A have policies assigned to them, solutions for the first time slot are generated. Only solutions that comply with the policies of the appliances in set A are added to the PQ. Then, a solution is removed from the PQ and extended with the next slot. During this step [algorithm 2](#) and [algorithm 3](#) creates solutions that result in a valid schedule. Instead of including all the solutions only the solution with the lowest cost is added to the PQ. This process continues until the solution removed from the queue contains N slots, where  $N = 24$  and the algorithm terminates. Hence, the algorithm always terminates as solutions are extended with a new time slot in each iteration.

### 3.3.2 Completeness

In this subsection we discuss the completeness of [algorithm 1](#). During an iteration a solution for a schedule is extended with the another slot. In the expansion step the generated solutions are validated against three rules;

- a solution has to contain all the policies that have a duration equal to the number of slots left
- a solution has to finish a policy once it is started
- all the policies from set P should be obeyed.



Only the slot with the lowest cost and obeying all three rules mentioned above is added to the solution. So the solution for the schedule found will meet all policies  $p \in P$  for each appliance in set  $A$ .

The cost function assures that the costs are increasing when the solution is expanded with another slot. Consequently, the optimal solution for the schedule is always at the start of the PQ. Therefore the algorithm will always return an optimal or near-optimal schedule.

### 3.3.3 Complexity

The complexity of [algorithm 1](#) for an input of  $n_a$  appliances with  $m_p$  policies is calculated as follows. Before starting the main loop, a set of solutions is created by calling [algorithm 2](#). This algorithm creates the power set of the appliances and their policies, which contains  $2^{n_a m_p}$  elements. The set is added to the PQ in a loop taking  $O(2^{n_a m_p})$  time. At this point the PQ holds  $2^{n_a m_p}$  incomplete schedules. In order to complete the schedules, each iteration removes one schedule from the PQ. Next, a set of solutions for the new slot is generated by [algorithm 2](#), resulting in  $2^{n_a m_p}$  solutions. Finding the slot with the lowest cost takes  $O(2^{n_a m_p})$  time. When a slot is found, we have to validate all of the policies, taking  $O(n_a m_p)$  time. The total time of expanding a schedule with a slot is  $O(2^{n_a m_p} n_a m_p)$ . When simplified we get a complexity of  $O(n2^{2n})$ , where  $n = n_a m_p$ .

The worst case time complexity is influenced by the number of solutions generated by [algorithm 2](#). In the worst case this algorithm generates  $2^{n_a m_p}$  solutions for extending a slot, where  $n_a$  is the number of appliances and  $m_p$  the number of policies. But in most cases the number of solutions will be lower than  $2^{n_a m_p}$ .

Most appliances will either not be used or only once throughout the day. There are exceptions such as a refrigerator or a freezer which turn on and off at specific intervals. Instead of creating multiple policies, the same policy is used multiple times, which means that in most cases  $m_p = n_a$ .

Policies belonging to the same appliances will not be combined together, after all an appliance can only execute one task at a time slot. For example, given an input of  $A = (A_1, A_2)$  where  $A_1$  has policies  $P_1, P_2$  and  $A_2$  has policy  $P_3$ , [algorithm 2](#) returns the incomplete power set  $( ), (P_1), (P_2), (P_3), (P_1, P_3), (P_2, P_3)$ .

Finally, [algorithm 2](#) will only create solutions containing incomplete policies. Once a policy is completed it will no longer be part of the solutions generated by the algorithm. As the number of slots in a solution increases, more policies are completed which in turn reduces the number of slots created by [algorithm 2](#). Thus in most cases the algorithm will perform better than the worst time complexity.

---

**Algorithm 1:** FindSchedule( $A$ )

---

**Data:** A set  $A$  with appliances containing policies to schedule.**Result:** Schedule  $S$  containing  $k_0 \dots k_{23}$  slots.

```

1  Let  $PQ$  be an empty Priority Queue
2  Let  $S$  be an empty schedule
3   $X \leftarrow \text{CreateSolutions}(A, S)$ 
4  for  $x \in X$  do
5  |    $PQ.\text{push}(x, x.\text{cost})$ 
6  end
7  while not  $PQ.\text{isEmpty}()$  do
8  |    $S \leftarrow PQ.\text{removeItem}()$ 
9  |   if  $S.\text{nrSlots}() == 24$  then
10 |      /* Return schedule  $S$ , if  $S$  contains 24 slots      */
11 |      return  $S$ 
12 |   end
13 |   /* Generate set of possible slots based on the
14 |      policies of each appliance in  $A$ .                      */
15 |    $K \leftarrow \text{CreateSolutions}(A, S)$ 
16 |   Let  $k_{\text{optimal}}$  be an empty solution
17 |   Let  $\text{cost}_{\text{prev}}$  be an empty variable
18 |   for  $k \in K$  do
19 |       if  $k.\text{getCosts}() < \text{cost}_{\text{prev}}$  then
20 |            $\text{valid} \leftarrow \text{True}$ 
21 |           /* Validate if the schedule still complies with
22 |              all the policies from the appliances in  $A$       */
23 |           for  $a \in A$  do
24 |               if not  $\text{validateSchedule}(S, a.\text{policies}())$  then
25 |                    $\text{valid} \leftarrow \text{False}$ 
26 |                   break
27 |               end
28 |           end
29 |           if  $\text{valid}$  then
30 |                $\text{cost}_{\text{prev}} \leftarrow k.\text{getCosts}()$ 
31 |                $k_{\text{optimal}} \leftarrow k$ 
32 |           end
33 |       end
34 |   end
35 |   /* Finally the optimal slot is added to the schedule
36 |      and is added back into the Priority Queue              */
37 |    $\text{extend } S \text{ with new solution } k_{\text{optimal}}$ 
38 |    $PQ.\text{add}(S)$ 
39 end

```

---

---

**Algorithm 2:** CreateSolutions( $A, S$ )

---

**Data:** A set  $A$  containing appliances to schedule  
and an either empty or an incomplete schedule  $S$ .

**Result:** List with *solutions* for expanding schedule  $S$ .

```

1 Let fixed be an empty set
2 Let schedulable be an empty set
3 Let policySets be an empty list
4 for  $a \in A$  do
5   for  $policy \in a.policies()$  do
6     if policy is already completed in S then
7       | continue to next policy
8     end
9     if policy is schedulable then
10      | schedulable.add(policy)
11    else if policy is active the next timeslot of S then
12      | fixed.add(policy)
13    end
14  end
15  if number of policies in a.policies() > 1 then
16    | policySets.add(a.policies)
17  end
18 end
19 return CombinePolicies(fixed, schedulable, policySets)

```

---

---

**Algorithm 3:** CombinePolicies(*Fixed*, *Schedulable*,  $P_{sets}$ )

---

**Data:** A set *Fixed* containing required policies for this slot,  
a set *Schedulable* containing optional policies for this slot, and  
a set  $P_{sets}$  containing sets of policies that cannot be combined.

**Result:** List *solutions* with slot solutions for the given policies.

```

1 Let solutions be an empty list
2 Add a new slot with policies from Fixed to solutions
  /* generateCombinations creates the power set of
    Schedulable policies without the empty set */
3 for  $x \in \text{generateCombinations}(\text{Schedulable})$  do
4   valid  $\leftarrow$  True
5   for  $set \in P_{sets}$  do
6     if number of elements in  $set \cap x > 1$  then
7       valid  $\leftarrow$  False
8       break
9     end
10  end
11  if valid then
12    solution  $\leftarrow x \cup \text{Fixed}$ 
13    add new slot with policies from solution to solutions
14  end
15 end
16 return solutions

```

---

## IMPLEMENTATION

In [Chapter 3](#) we discussed the concept behind the [CLSS](#), now we continue with the implementation of the system. The [Section 4.1](#) gives an overview of the system, followed by a short description of the tools that were used during the development. In [Section 4.2](#) the important frameworks on which the [CLSS](#) is built are discussed. Finally we examine the system more thoroughly and look at the individual components that make up the [CLSS](#).

### 4.1 SYSTEM OVERVIEW

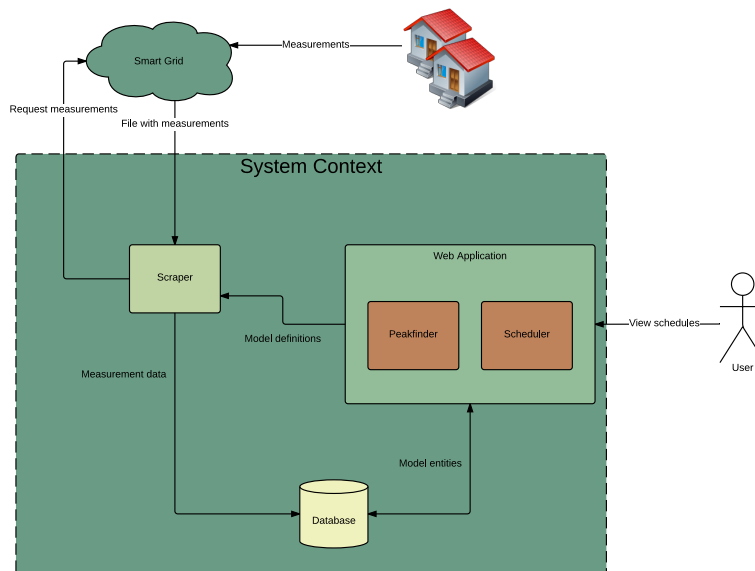


Figure 3: Components overview of the [CLSS](#).

The [CLSS](#) is divided into the following components as illustrated by [Figure 3](#), that are discussed next.

**SCRAPER:** retrieves a file from the Smart Grid ([SG](#)) holding measurements from appliances. After removing the incorrect measurements the data is stored into the database.

**DATABASE:** holds the data needed by the different components. The data stored is from measurements, information about smart meters, and schedules generated by the system.

**WEB APPLICATION:** is the component that holds the logic for generating schedules. Besides the logic, this component also provides

a web interface making it possible to view schedules and manage the database. External access to the component is facilitated by an [API](#).

#### 4.2 DEVELOPMENT TOOLS

The [CLSS](#) is developed on a Toshiba Satellite L500. For the integrated development environment (IDE) is Eclipse Juno [7] used in combination with the PyDev plug-in [17]. The *Scraper* component is build with the Scrapy framework [24]. Data required by the system is stored in the *Database* component using MongoDB [14]. The *Web application* is build in the Django framework [6]. Styling is provided by Bootstrap framework [19], except the graphs which are implemented implemented with the free version of Highcharts JavaScript library [11]. Now we will examine the main components discussed before more thoroughly.

#### 4.3 SCRAPER

The scraper component makes use of the Scrapy framework [24] written in Python. This framework offers a complete solution for turning the content of web pages into structured data that can be analysed and stored. Scrapy consists of the following components (see [Figure 4](#)) which are working together in order to scrape a web page;

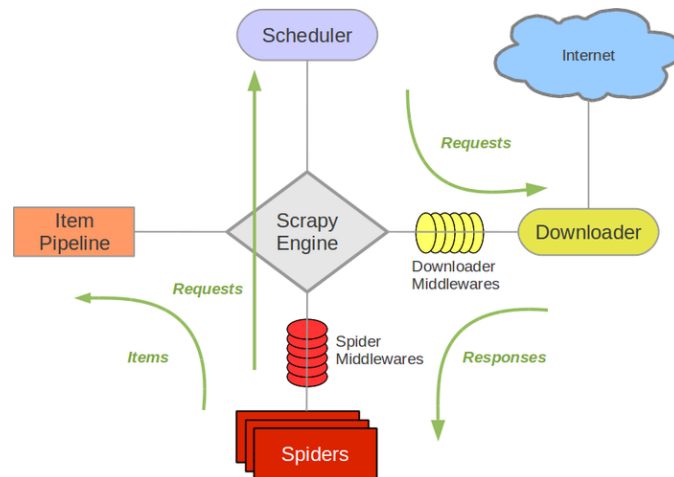


Figure 4: Components overview of Scrapy [23].

**SCRAPY ENGINE:** controls the data flow between all the different components within Scrapy.

**DOWNLOADER:** fetches web pages from the internet and passes them to the Scrapy Engine.

**SPIDERS:** are responsible for parsing the responses and extract items. They are custom written classes that inherit functions from a default spider. Allowing customised spiders so that specific content can be retrieved from a web page. Besides extracting items, spiders can also create new requests based on content from the web page.

**ITEM PIPELINE:** processes items that are extracted by a Spider. They are typically used for filtering, removing corrupted data or storing an item in a database.

**DOWNLOADER AND SPIDER MIDDLEWARE:** are specific hooks between the Scrapy Engine component and Downloader or Spider(s). The hooks can be used for custom code, such as catching or sending or receiving Hypertext Transfer Protocol Secure ([HTTPS](#)) requests.

#### 4.3.1 The spider

The spider is responsible for creating the [HTTPS](#) request and parsing the file containing measurements from appliances in households, available from a portal provided by the [SG](#). Users can specify the date and type of the measurements with a form on the download page. Once the user submits the form, a file with measurements for the specific date and type are available for downloading. When the spider sends a normal [HTTPS](#) request the response contains only the download page. Therefore, we use a *FormRequest* that contains a dictionary with parameters for the form. Instead of the download page the response now contains the requested file. The measurements of the requested file are distinguished in two records for a single smart meter: Instantaneous Demand Delivered ([IDD](#)) and Instantaneous Demand Received ([IDR](#)) measurements within an interval of ten seconds. Besides the measurements, each record also holds the postal code from the household, media access control ([MAC](#)) address of the smartbridge, [MAC](#) address of the smart meter, category of an appliance, and the type of measurement. Records are parsed into *RowItems* where after they are sent to the pipeline for further processing.

#### 4.3.2 The pipeline

The *pipeline* consists of three different components, as shown in [Figure 5](#). Firstly *RowItems* received from the *spider* component start at the *ParseDataRow* component. *RowItems* consist of multiple types of data that are divided in the following items;

**HOUSEHOLDITEM:** an item that stores the postal code and the [MAC](#) address of the smartbridge that is present in the household.

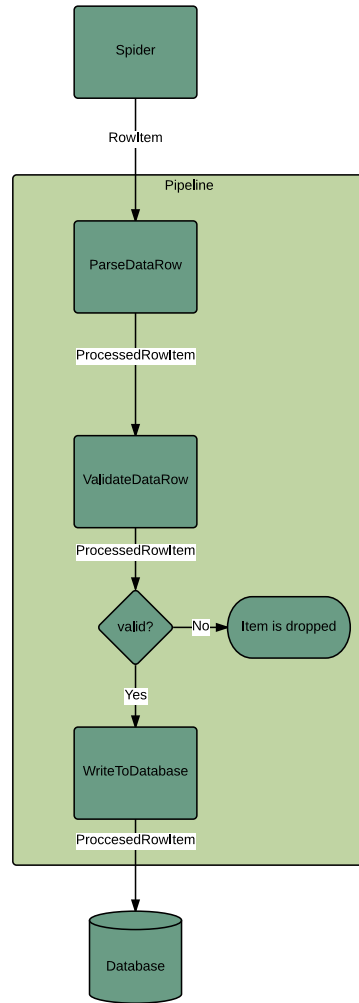


Figure 5: Dataflow diagram of the scraper component.

**SMARTMETERITEM:** an item that stores the [MAC](#) address of the smart-bridge, the smart meter [MAC](#) address, and the category.

**PROCCESSEDROWITEM:** an item that stores the type of measurement, the *HouseholdItem*, the *SmartmeterItem*, and the measurements. Only the *ProccesedRowItem* will continue to the next stage of the pipeline.

The *ValidateDataRow* component drops the items from the pipeline according to the following rules;

- the items needs to contain a *HouseholdItem* with postal code and smartbridge [MAC](#) address
- the category of the smart meter needs to be supported by the [CLSS](#)
- the description in the *ProccesedRowItem* has to match [IDD](#), indicating the measurements are the energy consumption measured by the smart meter.



Items that break one of the three rules are dropped from the pipeline. Finally the *WriteToDatabase* component makes a connection with the database and inserts the items into the database. Both the *HouseholdItems* and *SmartmeterItems* are written in a single database action. However, in order to reduce the database load the measurements are processed in batches.

#### 4.4 WEB APPLICATION

The web application is the central component of the *CLSS* containing the implementation of the algorithms discussed in [Chapter 3](#). While also containing a *GUI*, allowing users to interact with the *CLSS*. There are multiple frameworks available for building web applications, one of them is Django [6]. This framework is written in the Python programming language. The Web application contains three layers, namely a model, view, and template layer.

**MODEL LAYER:** an abstract layer between the web application and the database. In most cases a model maps to a single entity within the database. Attributes of an entity are represented in the model by fields. Entities are manipulated with the help of custom functions within the model.

**VIEW LAYER:** is responsible for handling requests from the user and returning responses. When a request is made by the user, the Uniform Resource Locator (*URL*) pattern is mapped to a view. Views are Python functions or classes that accept a web request and return a web request. The logic needed for returning web request lives in the view.

**TEMPLATE LAYER:** provides a syntax for rendering the views into web pages. Templates are simple text files that can generate any text-based format, such as Hypertext Markup Language (*HTML*), Comma Separated Values (*CSV*), etc.

The web application of the *CLSS* is divided into a *peakfinder* and, the *scheduler* application.

##### 4.4.1 *Peakfinder application*

The *peakfinder* application is built to provide a set of tasks that can be schedule based on data retrieved by the *scraper*. Metadata of the measurements is stored into *household*, *smartmeterCat*, and *smartmeter* models. Initially measurements for a single appliance are stored as an array into the *measurements* model by the *scraper*. At this point the measurements have an interval of ten second, instead of a hour interval as described in [Section 3.1](#). So, Map-Reduce [13] is used to

reduce the measurements to the correct interval. Once in the correct interval, they are stored into *HourlyMeasurement* models. For quick access the *HourlyMeasurement* model of a single day are embedded into a *DayMeasurement* models. Policies are stored into *Policy* models. The model contains the type of policy, date on which the policy is active, smart meter [MAC](#) address the policy is assigned to, and other fields depending on the type of policy (see [Section 3.2.2](#)).

Models only handle mutations on data that belong to the models itself. Managers are used for mutations on a collection of models. The *peakfinder* application holds the following managers;

**SMARTMETERMANAGER:** allows the system to quickly retrieve a list of smart meters for a household, based on the smartbridge *MAC* address.

**MEASUREMENTMANAGER:** provides a method for starting the Map-Reduce process of reducing the interval of the measurements from ten seconds into one hour.

**REDUCEDMEASUREMENTSMANAGER:** provides a method for calculating the energy consumption when an appliance is standby.

**DAYMEASUREMENTMANAGER:** has a method for retrieving *DayMeasurement* models for a specific smart meter and date.

**POLICYMANAGER:** has two methods for creating policies as described in [Section 3.2.2](#).

The [GUI](#) of the *peakfinder* application offers an overview of the households and detailed view of the households in the [CLSS](#).

**HOUSEHOLDLIST:** shows a table with households in the [CLSS](#) based on the *Household* model (see [Figure 20](#)).

**NEIGHBOURHOODHOUSEHOLDLIST:** is similar to the *HouseholdList* view, except showing only the household part of a given *Neighbourhood* model (see [Figure 21](#)).

**HOUSEHOLDDETAIL:** shows general information of the *Household* model and the smart meters inside the household. Besides this information, a schedule from the scheduler application can be viewed (see [Figure 22](#)).

#### 4.4.2 Scheduler application

The scheduler application is responsible for generating the schedules based on the task and policies defined by the *peakfinder* application (see [Section 4.4.1](#)). This application contains the following models;

**NEIGHBOURHOOD:** holds the name of the neighbourhood and a list of keys of related *Household* models.

**TASK:** Defines a single task for a smart meter in a household. Other fields in this model are the date on which the task is performed, the start time slot, and end time slot.

**SCHEDULEDTASK:** defines a scheduled task for a smart meter in a household. Also contains a date, reference to the policy leading to this task, the begin time slot, the end time slot, and energy consumption field.

Similar to the *peakfinder* application, each model has a manager for the interaction between models. Now we describe each of the managers;

**NEIGHBOURHOODMANAGER:** contains a method for retrieving a *neighbourhood* of a specific household.

**TASKMANAGER:** this manager has a method called *getTask* which returns a list of tasks that belong to a set of smart meters on a specific date. Energy consumption of multiple tasks can be requested with the method *getTaskTotalForNeighbourhood*, with a list of smart meters and a date as argument. Similar to the previous method, is the *getTaskSlotTotals* method. This method returns the sum of the energy consumption by the tasks for each slot. This method is also used by the *getTaskSlotsCost* that performs the cost calculation for each slot.

**SCHEDULEDTASKMANAGER:** contains method *createScheduleTasks*, that creates tasks based on slots objects from a schedule. Multiple *ScheduledTask*, that belong to a set of smart meters on a specific date can be retrieved with the *SmartmeterSet*. The *getScheduledTaskEnergyBySlot* returns the energy consumption per smart meter for each slot during a day, for a given set of smart meters. Similar to the *TaskManager*, it is also possible to get the total energy consumption or cost per slot for a set of smart meters at a specified date.

This application contains a number of overview views as well as the views for the index and login screen. The list below gives a short description of each of these views.

**INDEX:** view for rendering the home page of the web application (see [Figure 23](#)).

**USER\_LOGIN:** view for rendering the login form and handling the login process ([Figure 24](#)).

**LISTNEIGHBOURHOODVIEW:** view for rendering a list of neighbourhoods available in the system. This view is only available when the user is authorized to view the page (see [Figure 25](#)).

**DETAILNEIGHBOURHOODVIEW:** view for rendering the neighbourhood page for a given neighbourhood. The page contains an overview of the historical measurements used for creating a schedule, the tasks, the policies, and the schedule for a given date. Besides the tables, there are also a number of charts available. Tables are filled by retrieving the corresponding models with the help of the managers. Data for the charts is retrieved by using the REST API (see Section 4.4.3). As the charts are generated with Javascript, the view only includes a date for the template to parse (see Figure 26).

This application also contains the implementation of algorithm 1, that is implemented in a separate class called *ScheduleFinder*. Solutions created by algorithm 2 are implemented in the *ScheduleNode* class, which contains a list with timeslots and a method for calculating the cost of the solution. Each timeslot is an instance of the *SlotSolution* class, which contains a number indicating the timeslot and a list with smart meter MAC addresses of the appliances that are scheduled in the respective timeslot.

#### 4.4.3 REST API

The Django REST Framework [4] is a framework for building a web API. The framework is built on top of the Django framework [6]. APIs create an abstract layer to other devices, such as smartphones, which enables them to communicate with the actual application. This layer hides the logic for storing and retrieving the information. Devices only need to give the correct parameters and the API returns the requested information in either the JavaScript Object Notation (JSON) or the HTML format.

When data from a model is requested, the framework provides serializers for serializing and deserializing a Django model entity into JSON or HTML. Besides using direct requests, the framework offers in addition a browsable API (see Figure 6).

The framework also offers a multiple authentication methods such as;

**BASIC AUTHENTICATION:** which uses a combination of username and password for authentication.

**TOKEN AUTHENTICATION:** where a token is used as a key for authentication.

**SESSION AUTHENTICATION:** where the Django session back-end is used.

**OAuth 1.0a AUTHENTICATION:** where the OAuth 1.0a protocol [9] is used for authentication.

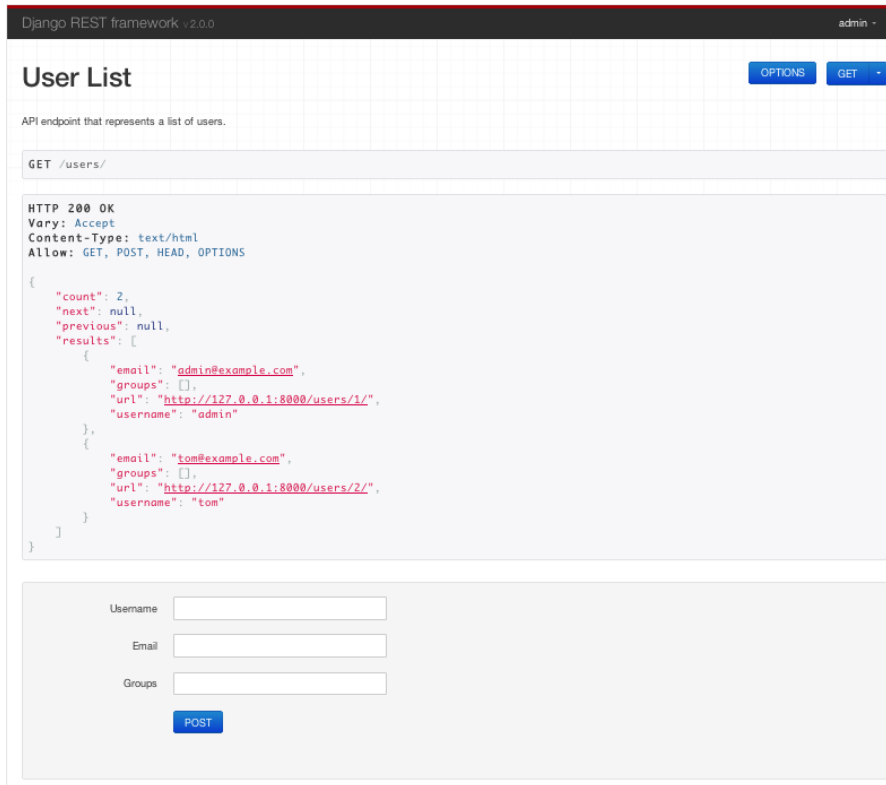


Figure 6: Screenshot from the browsable API [4].

**OAUTH 2.0 AUTHENTICATION:** where the OAuth 2.0 protocol [10] is used for authentication.

## 4.5 DATABASE

The database is MongoDB [14], a NoSQL, document based database. Compared to the Structured Query Language (SQL) solutions, MongoDB provides a more flexible method of storing information. The data is stored into a document, which has one or more fields. Fields have a key and a value, of which the key is similar to a column name of a table in SQL. Unlike SQL solutions, values can hold both a single value or another document (see Figure 7). In the CLSS the embedded data model is only used in the peakfinder application for the *Measurements*, the *ReducedMeasurements*, and *DayMeasurement*. In which the embedded document holds the energy consumption at a given time. Documents are stored into collections, equivalent to tables in a SQL solution. The collections may contain documents with different fields, however in most cases the documents in a collection have the same structure.

Two other important functions of MongoDB are sharding and replication of data. Sharding allows distribution of the database across multiple machines. With sharding the data is divided between servers,

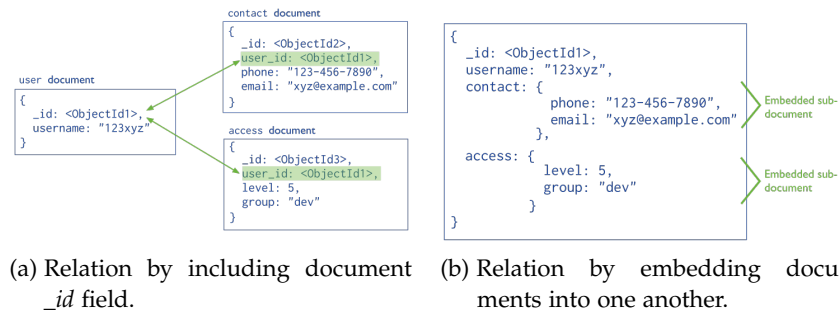


Figure 7: Different data model types [14].

also called shards. A shard is an independent server running a MongoDB instance with storing a slice of the database. All shards together form a single database. The function of sharding is used to solve two problems associated with large data sets. First, the number of operations, such as inserting data, is shared between shards. For increasing the capacity and throughput of the database, more shards can be added. Second, the size of the data set complete stored on the shard can be smaller than the complete data set. This is because all the shards have an equal slice of the data set.

Replication increases the data availability by storing copies of the the data on different database servers. In this way, the loss of data when a single server goes down will be avoided. At the same time replication can be used to increase the read capacity of the system. As clients have the choice to which server they write their operations to.

The CLSS is at the moment a single machine system, where the data is stored on a single MongoDB instance. However when needed, the CLSS has the possibility to use more MongoDB instances.

### Part III

## SIMULATION AND VALIDATION





## RESULTS

---

We discuss the results of our research on the scheduling of household appliances. The first section explains which data is used for the test and which matrices we are interested in. Once the test setup is completed, the running of the algorithm and results are explained in [Section 5.2](#). After [Section 5.1](#) we analyse the results obtained in [Section 5.2](#). Finally, we have a short discussion over the results found.

### 5.1 SETUP

Before we start testing our algorithm, we first need to define a data set that can be used as input. Once the data set is defined, we need to find a metric that enables us to define the quality of the results.

#### 5.1.1 *The data*

The data sets are based on the measurements gathered by the project "Smart grid: Rendement voor iedereen" [[12](#)]. During the project, pilots are held in Amersfoort and Utrecht. In these pilots, household members are encouraged to make smart choices when it comes to their energy consumption. This encouragement is based on raising awareness of these people about the energy consumption and energy production of their household. Therefore, households are fitted with smart meters for measuring the energy consumption in watts. Each household is fitted with up to five smart meters that connect to different appliance inside the household. The energy consumption is measured in watts with an interval of ten seconds, gathering 6 measurements in a second,  $6 \cdot 60 = 360$  measurements in a hour, and  $6 \cdot 60 \cdot 24 = 8460$  measurements in a 24 hour period per appliance.

In order to test our algorithm, we created three data sets namely; data set *A*, *B*, and *C*. We assigned randomly picked households from the pilots to the neighbourhoods. An overview of the appliances is given in [Table 2](#).

Data set *A* and *B* contain measurements collected on 1 February 2014, while measurements in data set *C* are from 17 October 2014.

#### 5.1.2 *The metrics*

Now that the data sets are defined, we need a metric to validate the results. The goal of our system is to minimize peaks in the energy demand of a group of households. In order to quantify the peaks,

APPLIANCE	A	B	C
Dishwasher	1	1	1
Freezer	1	0	2
Remaining / Unknown	7	17	4
TV / Video	1	0	2
Was machine	3	0	2
Was dryer	1	1	1

Table 2: Appliances available in each of the three data sets.

we calculate the difference between the local energy demand and the average energy demand required per timeslot. The energy demand is the total of energy required by all appliances occurring in the data set. This demand is calculated by using a cost function, which is defined in Equation 7 (see Section 3.3).

As a final metric, to compare the input energy consumption against the scheduled energy consumption, we use the following function:

$$\text{CostDiff}(n) = 100 \cdot \frac{\text{Cost}_{\text{scheduled}}(n) - \text{Cost}_{\text{input}}(n)}{\text{Cost}_{\text{input}}(n)} \quad (9)$$

where  $\text{Cost}_{\text{input}}(n)$  is the cost for the original input and  $\text{Cost}_{\text{scheduled}}(n)$  is the cost after scheduling the original input for timeslot  $n$ .

## 5.2 RUNNING

We ran our algorithm on the data sets defined earlier. Here we describe how the algorithm runs with the data sets and the results after using the algorithm.

### 5.2.1 Generating tasks and policies

During the initial phase of running the algorithm, tasks and policies are created for each data set. Not every appliance in the data set is associated with a task. But on the other hand, a task is always associated with one or more policies.

The results of running the algorithm against the data sets are shown in Figure 8, Figure 9, and Figure 10. These figures give an overview of the energy consumption during the day. A single row displays the MAC address and the appliance type, followed by the policies, energy consumption per timeslot, and the standby energy consumption. The policy abbreviations stand for Profile (PR), Single (SI), Strict (ST), Repeating (RP) and Delay (DE). Colour codes indicate the state of the appliance in the corresponding timeslot, namely:



Table colors legenda: Active / Standby Changing state Off

Smartmeter Eui64	Category	Policies	Hourly energy consumption (Watt)																								Standby Usage (Watt)
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
84df-0c00:1000:0099	Overig	RP, SI	12	23	25	9	38	0	42	0	39	0	45	0	45	0	40	0	57	1	52	0	46	0	37	0	1
84df-0c00:1000:00ba	Overig	PR	218	30	20	12	16	10	9	22	25	17	13	84	33	19	13	25	16	16	28	14	17	16	27	12	n/a
84df-0c00:1000:009e	Onbekend	RP, RP, SI	5	27	0	40	0	24	11	0	39	0	33	0	29	2	32	5	14	19	0	34	0	37	0	36	n/a
84df-0c00:1000:00a3	Onbekend	PR	22	12	44	38	40	36	18	25	44	42	30	19	18	38	34	36	35	35	22	25	37	42	42	35	n/a
84df-0c00:1000:00cf	Onbekend	SI	0	0	0	0	0	0	0	0	0	0	49	335	4	4	0	0	0	0	0	0	0	0	0	0	4
84df-0c00:1000:00db	Onbekend	SI, SI	0	0	0	0	0	0	0	0	0	0	233	1570	108	1	598	600	1	1	1	1	1	1	1	1	1
84df-0c00:1000:009b	Overig	SI	60	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
84df-0c00:1000:0099	Overig	PR	81	70	61	114	124	84	78	67	60	65	58	76	98	95	75	76	77	74	75	107	133	116	86	86	n/a
84df-0c00:1000:00ef	Overig	SI	430	355	617	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	n/a
84df-0c00:1000:0091	Atwasmaschine	RP	405	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	1
84df-0c00:1000:0093	Overig	SI	24	20	21	16	16	23	24	19	51	46	44	35	25	18	21	42	42	35	39	38	35	36	32	7	7
84df-0c00:1000:00b2	Overig	PR	220	41	36	48	23	57	28	49	39	39	56	31	47	61	59	77	64	66	64	55	69	80	71	41	n/a
84df-0c00:1000:00f1	Overig	SI	0	0	0	0	0	0	0	0	0	0	0	261	298	497	460	93	270	4	0	0	0	0	0	0	4
13 day measurements																											

Figure 9: Energy consumption of the appliances in data set B.

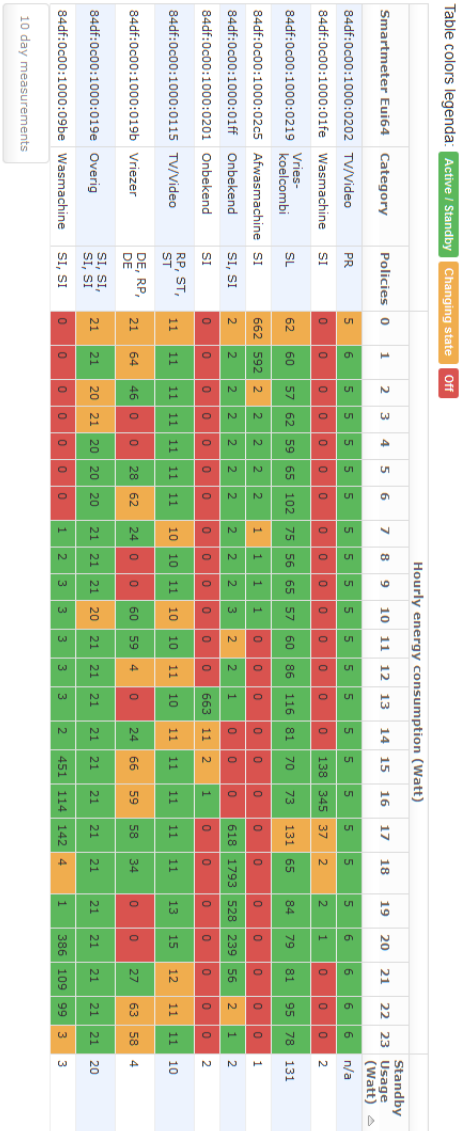


Figure 10: Energy consumption of the appliances in data set C.

### 5.2.2 Generating schedules

The policies created in [Section 5.2.1](#) are the input for the scheduling algorithm. At the start of the calculation, the average energy consumption of a slot is calculated for each data set. After that the scheduler algorithm will look for a schedule that complies with the policies and approaches the energy consumption average for each slot.

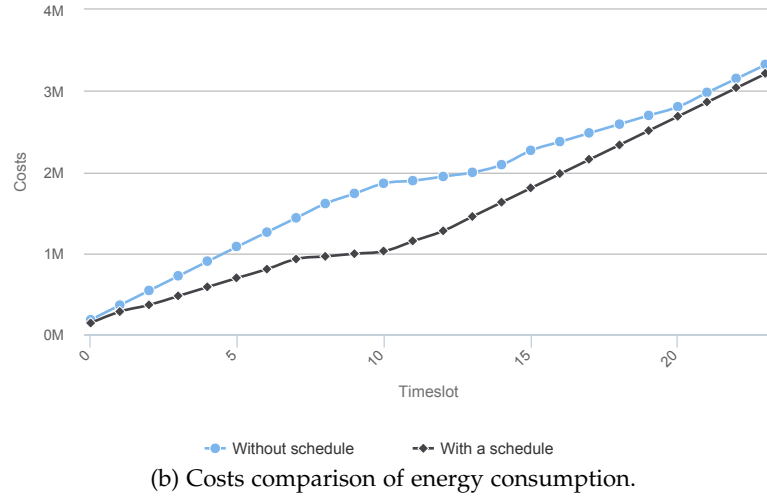
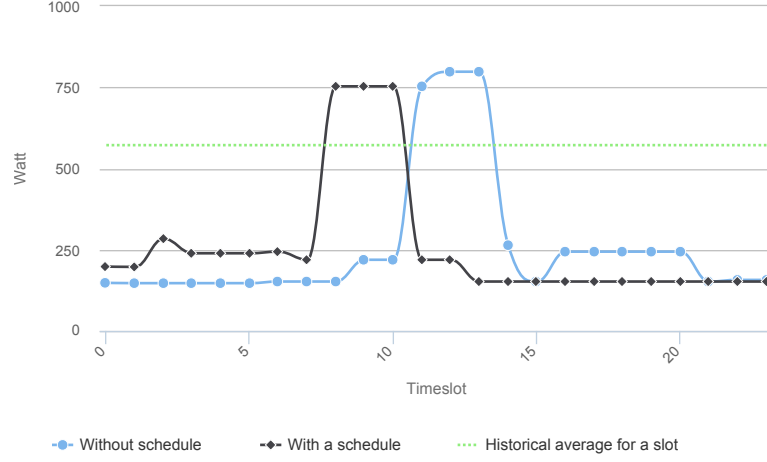
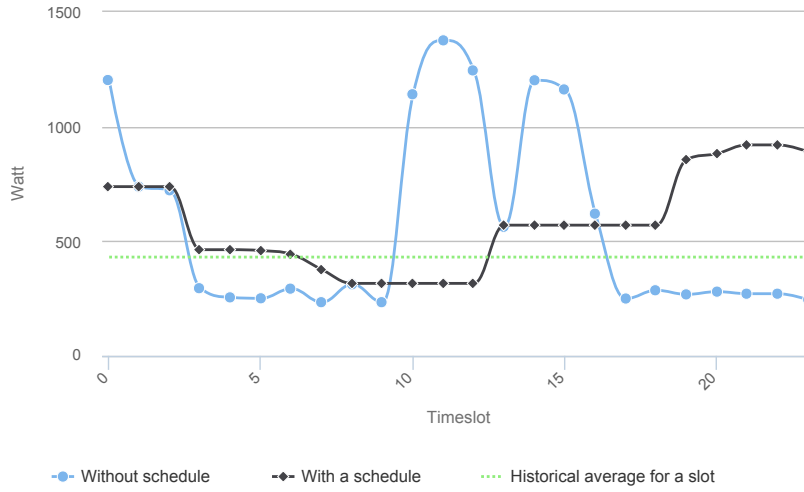
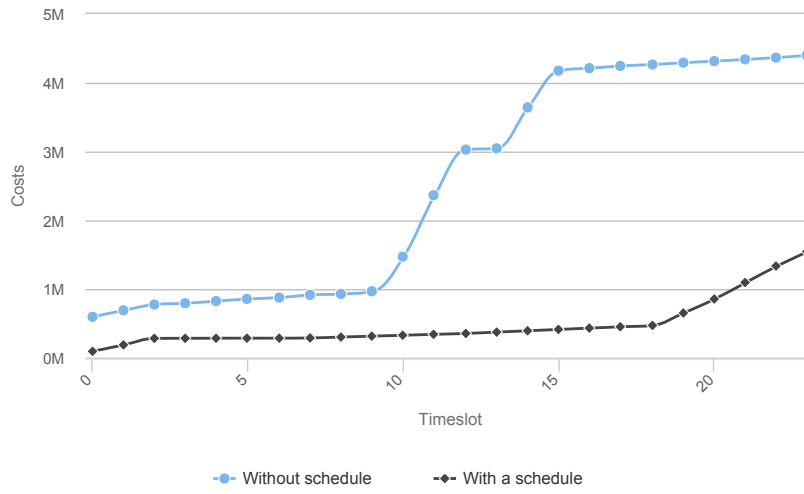


Figure 11: Energy consumption and cost charts for data set A.

Results of the scheduling algorithm are shown in [Figure 11](#), [Figure 12](#), and [Figure 13](#). The first chart shows the energy consumption per timeslot for the respective data set, which is the sum of all the appliances. The horizontal axis indicates the timeslot, while the vertical axis displays the energy consumption in watts. Included in the chart is the average energy consumption of a slot, represented as a green dotted line. The second chart shows the cost for each timeslot during the day. Similar to the first chart, the horizontal axis indicates the timeslot. On the vertical axis are the cost for the timeslot visible, calculated with the [Equation 7](#) (see [Section 3.3](#)).

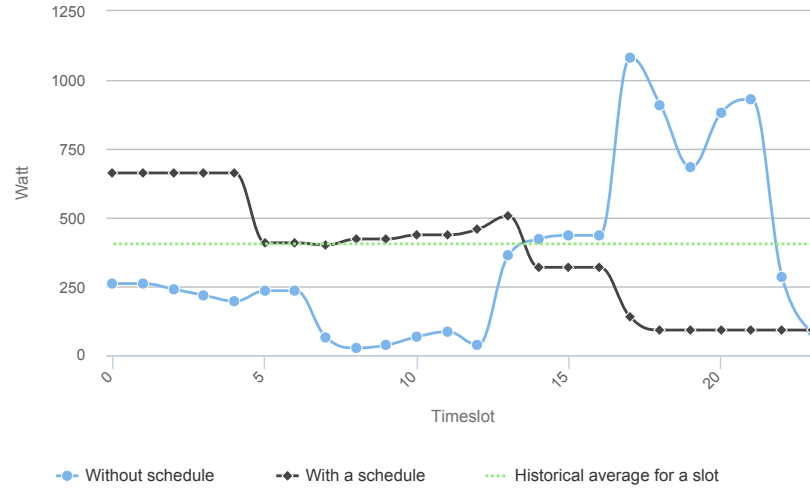


(a) Energy consumption comparison.

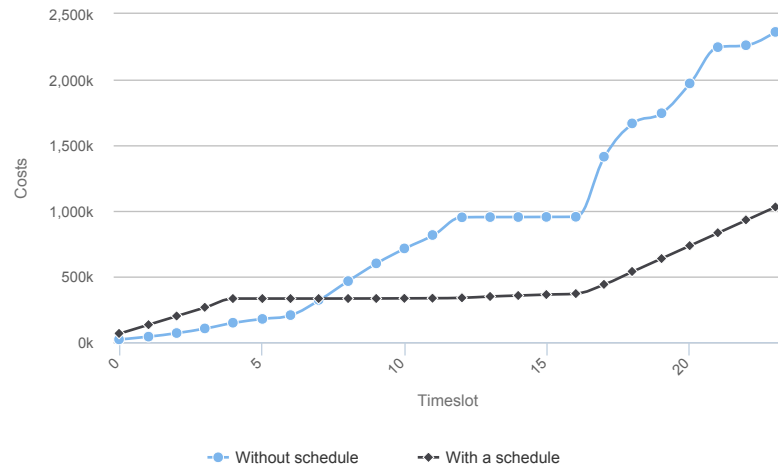


(b) Costs comparison of energy consumption.

Figure 12: Energy consumption and cost charts for data set *B*.



(a) Energy consumption comparison.



(b) Costs comparison of energy consumption.

Figure 13: Energy consumption and cost charts for data set C.



An overview of the schedule energy consumption is shown in Figure 14, Figure 15, and Figure 16. Each row of these tables shows the energy consumption of an appliance performing a task. The first column gives the unique MAC address of the smart meter coupled to the appliance, followed by the category. After the MAC address and category, the hourly energy consumption is shown in watts, where orange indicates the original consumption and green consumption after scheduling. In the last column we show how many timeslots the task has shifted.

Legend:		Optimized slots		Critical slots		Hourly energy consumption (kWh)																								Task	
																									Delay						
Smartmeter Enl64	Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23						
84df:0c00:1000:00e4	Overig	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	19	0	22 ↓				
84df:0c00:1000:00a9	Wasmachine	45	45	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12 ↑				
84df:0c00:1000:00e9	Onbekend	0	0	0	0	0	0	0	67	67	67	67	67	67	0	0	0	0	0	0	0	0	0	0	0	0	2 ↑				
84df:0c00:1000:00a1	TV/Video	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	0				
84df:0c00:1000:00a1	TV/Video	0	0	0	0	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0	0				
84df:0c00:1000:009c	Overig	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	0	0				
84df:0c00:1000:00e4	Overig	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	43	0	0				
84df:0c00:1000:00bd	Vries-	0	0	0	0	0	0	0	534	534	534	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3 ↑				
84df:0c00:1000:00e9	Koelcombi	5	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22 ↑				
84df:0c00:1000:00e6	Overig	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	0	0				
84df:0c00:1000:00e9	Onbekend	0	0	92	92	92	92	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14 ↑				

Legenda: Optimized slots Original slots

Figure 14: Energy consumption for data set A with and without schedule.

Legend:		Optimized slots		Original slots		Hourly energy consumption (kWh)																								Task Delay $\Delta$
Smartmeter Eui64	Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23					
84df:0c00:1000:00d9	Overig	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	84	0				
84df:0c00:1000:0093	Overig	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	30	0				
84df:0c00:1000:00ef	Overig	467	467	467	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
84df:0c00:1000:00a9	Onbekend	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	32	0				
84df:0c00:1000:009b	Overig	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7 $\uparrow$				
84df:0c00:1000:00db	Onbekend	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	637 637 637 11 $\uparrow$				
84df:0c00:1000:00b2	Overig	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	0				
84df:0c00:1000:009e	Onbekend	16	16	16	16	16	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12 $\uparrow$				
84df:0c00:1000:009e	Onbekend	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	26	26	26	20 $\uparrow$				
84df:0c00:1000:0091	Afwasmachine	0	0	0	0	0	209	209	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5 $\uparrow$				
84df:0c00:1000:009e	Onbekend	0	0	0	0	0	0	0	59	59	59	59	59	59	0	0	0	0	0	0	0	0	0	0	0	4 $\uparrow$				
84df:0c00:1000:00f1	Overig	0	0	0	0	0	0	0	0	0	0	0	0	0	313	313	313	313	313	0	0	0	0	0	0	2 $\uparrow$				
84df:0c00:1000:00db	Onbekend	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	599	599	0	0	0	5 $\uparrow$				
84df:0c00:1000:00ba	Overig	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	29	0				
84df:0c00:1000:00cf	Onbekend	0	0	0	192	192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7 $\uparrow$				
84df:0c00:1000:0099	Overig	21	21	21	21	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
84df:0c00:1000:0099	Overig	0	0	0	0	0	0	0	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	1 $\uparrow$				

Figure 15: Energy consumption for data set  $B$  with and without schedule.

Legend:		Optimized slots		Original slots		Hourly energy consumption (kWh)																							Task Delay $\Delta$
Smartmeter Eui64	Category	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23				
84df:0c00:1000:0202	TV/Video	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0			
84df:0c00:1000:019b	Vriezer	0	0	0	0	0	0	0	0	59	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2 $\uparrow$			
84df:0c00:1000:0115	TV/Video	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	11	11	11	11	11	11	11	11	11	0			
84df:0c00:1000:019b	Vriezer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	48	48	48	0	0	0	0	0	0	1 $\uparrow$			
84df:0c00:1000:019b	Vriezer	0	0	0	0	0	0	0	0	0	0	0	0	0	0	54	54	54	54	54	54	54	54	54	54	10 $\uparrow$			
84df:0c00:1000:019e	Overig	0	0	0	0	0	0	21	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6 $\uparrow$			
84df:0c00:1000:019e	Overig	0	0	0	0	0	0	0	0	0	0	21	21	21	21	21	21	21	21	21	21	21	21	21	21	0			
84df:0c00:1000:019e	Overig	0	0	0	0	0	0	0	0	21	21	21	0	0	0	0	0	0	0	0	0	0	0	0	0	1 $\uparrow$			
84df:0c00:1000:02c5	Overig	0	0	0	0	0	0	0	0	0	0	180	180	180	180	180	180	180	0	0	0	0	0	0	0	10 $\uparrow$			
84df:0c00:1000:09be	Wasmachine	0	0	0	0	0	0	0	0	0	0	177	177	177	177	0	0	0	0	0	0	0	0	0	0	5 $\uparrow$			
84df:0c00:1000:09be	Wasmachine	0	0	0	0	0	198	198	198	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15 $\uparrow$			
84df:0c00:1000:019e	Overig	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2 $\uparrow$			
84df:0c00:1000:0115	TV/Video	0	0	0	0	0	0	0	0	0	0	0	0	21	21	0	0	0	0	0	0	0	0	0	0	3 $\uparrow$			
84df:0c00:1000:0201	Onbekend	0	0	0	0	0	0	0	0	337	337	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5 $\uparrow$			
84df:0c00:1000:01ff	Onbekend	646	646	646	646	646	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17 $\uparrow$			
84df:0c00:1000:01ff	Onbekend	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3 $\uparrow$			
84df:0c00:1000:0115	TV/Video	11	11	11	11	11	11	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
84df:0c00:1000:01fe	Wasmachine	0	0	0	0	0	173	173	173	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10 $\uparrow$			

Figure 16: Energy consumption for data set C with and without schedule.

### 5.3 ANALYSIS

In [Section 5.2](#) we ran the algorithm against the three data sets *A*, *B*, and *C*. Now we analyse the different results presented in the previous section.

#### 5.3.1 Task and Policies

The first part of our algorithm is required to determine the tasks and policies for the appliances in a data set. This result of the first part of this calculation forms than the input for the scheduling algorithm. Tasks are defined as periods with higher energy demand compared to the standby energy demand. Once the tasks are identified one or more policies are assigned.

DATA SET	PROFILE	SINGLE	STATIC	DELAYED	REPEATING
<i>A</i>	4	4	2	1	0
<i>B</i>	3	10	0	0	4
<i>C</i>	1	11	2	2	2

Table 3: Policies assigned in each of the three data sets

An overview of the number of policies per data set can be found in table [Table 3](#).

#### 5.3.2 Before and after scheduling

Data set *A* has an average energy consumption of 571.9 watt. In most of the timeslots, as well as before as after scheduling, the energy consumption stays below the average energy consumption (see [Figure 11a](#)). Gathered measurements show a peak between timeslot 11 and 13 and during this peak a maximum of 798 watt consumed. After running the scheduling algorithm, we still have a peak between timeslot 8 and 10. However, the energy consumption of the peak is reduced by 219 watt at most.

Data set *B* has an average energy consumption of 430.69 watt. During most of the timeslots the energy consumption stays around the average. The original measurements show three peaks. The first one at timeslot 0, the second at timeslot 11, and the third at timeslot 14. These peaks have, in order it has been sequenced, an energy consumption of 1203, 1376, and 1202 watt. After scheduling the energy consumption stays closer to the average, as can be seen in [Figure 12a](#). The initial peak at timeslot 0 still exists, but the energy consumption is now 738 watt. As for the other two peaks, they are gone now and

instead the energy consumption increases near the end from 570 watt at timeslot 13 to 920 watt at timeslot 21.

Data set C shows an average energy consumption of 404 watt, as can be seen in [Figure 13a](#). We also see that the energy consumption is below the average energy consumption until timeslot 14. After this timeslot, the energy consumption increases rapidly, with a peak at timeslot 17 of 1081 watt. This rapid increase continues until timeslot 21 with an energy consumption of 930 watt. If we look at the energy consumption after scheduling, we see the reverse image. In the first few timeslots the energy consumption is above average namely at 622 watt. However at timeslot 5, the energy consumption drops down to the average. The consumption continues to decline and is even below average at timeslot 14, which was initially 319 watt, but dropped after scheduling to 91 watt.

The percentage of change as described in [Equation 9](#) can be found in [Figure 17](#), [Figure 18](#), and [Figure 19](#). In these, the horizontal axis shows the timeslot and the vertical axis shows the cumulative percentage of change. A higher percentage indicates a lower cost (see [Section 5.1.2](#)) after scheduling, except when the percentage is above the hundred percent. The value on top of the bars give the exact percentage of change in the cost for the given timeslot.

[Figure 17](#) shows a growing reduction in cost for data set A. The cost reduction continues until timeslot 10. After timeslot 10, the costs of the schedule are rising which is displayed in the chart by a steady decline of the difference between the input and the schedule. In the end, only a 3,31% improvement over the original input is left.

Opposite to data set A, data set B shows a steady percentage of change during the complete period, shown in [Figure 18](#). The cost without scheduling are quit high from the start. In addition, the costs are also increasing at a higher rate than the cost with the input in comparison to the schedule. Because of these differences between the input and schedule, the final result shows a 64% improvement over the original input, so this schedule is less costly than the original.

Data set C is the only data set in which the first couple of timeslots have a higher costs with a schedule than without. This higher cost is visible in [Figure 19](#) by the percentage of change being over the 100 for timeslot 0 to timeslot 4. After this, the percentage of change decreases to 87,73 and it remain below 100 percent with a final cumulative result of 56,43 percent. Hence after the four timeslots, the schedule produces less cost in comparison to the end.

## 5.4 DISCUSSION

The goal of our application is to improve the ratio between peaks and the average energy consumption for a given data set for one day. Because the data sets only contain measurements, we first need to

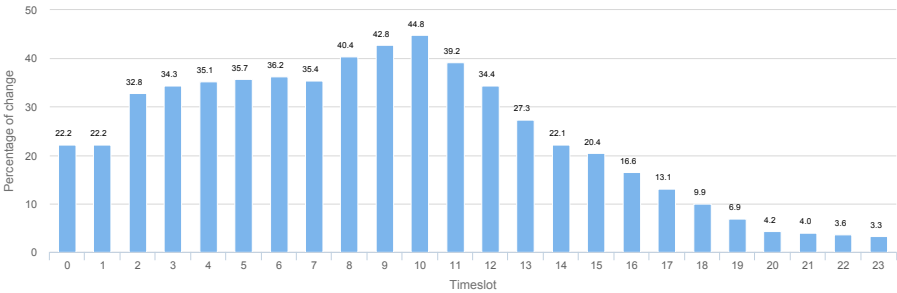


Figure 17: Cumulative percentage of change according to Equation 9 after scheduling for data set A.

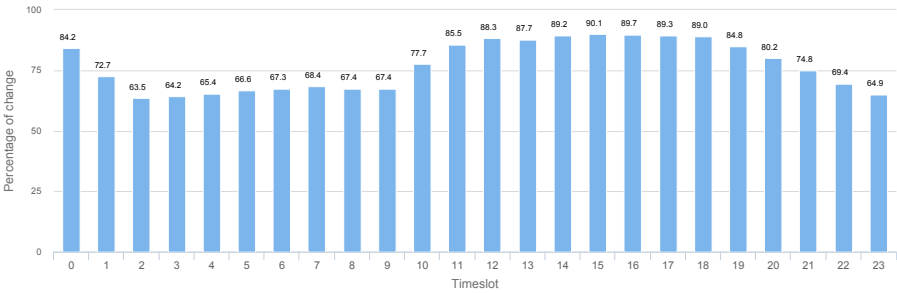


Figure 18: Cumulative percentage of change according to Equation 9 after scheduling for data set B.

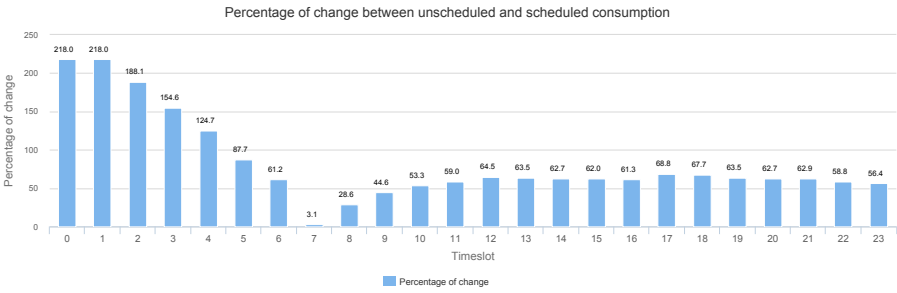


Figure 19: Cumulative percentage of change according to Equation 9 after scheduling for data set C.

find the active periods, which are mentioned as tasks and we have to assign policies. In [Section 5.3](#) we showed that we are successful in finding tasks and assigning policies. However the standby energy value is not determined for all appliances. If the standby energy consumption is detected, it allows the application to reduce the duration of the task. For example, in [Figure 11a](#) we see a standby consumption of 2 watt by the appliance with MAC address `84df:0c00:1000:00e9`. During the task creation of this appliance, the standby energy consumption will lead to two tasks with the SI policy, instead of a single task with a PR policy, this makes this appliance schedable.

Regarding to the actual scheduling of the data sets, we see an overall improvement in each of the three data sets. Data set A has the smallest percentage of change out of the three with a 3,3 percentage of change according to [Figure 17](#). So the schedule created for data set A is only slightly better than the original input. Looking at [Figure 11a](#), one would conclude that the algorithm fails to remove the peak, because the peak in the energy consumption still exist in the schedule although in a reduced and delayed form. So an important question is, why is this peak not removed but only reduced? The explanation can be traced back to the appliances causing the peak. In this case there are three appliances active during the peak (see [Figure 8](#)). In [Table 4](#) we calculated the energy used by the appliances. This table indicates that the peak is caused by the appliance with smart meter MAC address `84df:0c00:1000:00bd`. Hence, by placing the other appliances in different timeslots there will still be a peak caused by MAC address `84df:0c00:1000:00bd`.

SMART METER MAC	CATEGORY	ENERGY USED (IN WATT)
<code>84df:0c00:1000:00aa</code>	Wasmachine	137
<code>84df:0c00:1000:00e9</code>	Unknown	399
<code>84df:0c00:1000:00bd</code>	Freezer	1604

Table 4: Energy used by appliances that can be scheduled active between timeslot 11 and 15, according to [Figure 8](#).

Data set B has an improvement of 64,9% as shown in [Figure 18](#). This is also reflected in [Figure 12a](#), where we see that the schedule stays closer to the green dotted line, which is indicating the optimal energy consumption. At peak periods the percentage of change lies between the 84 and 90,10 percent. This peak is caused in the original data set by two appliances, using between the 261 watt and 1570 watt in a single slot (see [Table 5](#)). In order to remove the peak one of the appliances is advanced to timeslot 19 until 23.

Therefore the schedule is consuming more energy in this part of the day than in comparison to the input, as shown in [Figure 12a](#). Despite, the increase in energy consumption during time slot 19 until 23 com-

SMART METER MAC	CATEGORY	ENERGY USED (IN WATT)
84df:0c00:1000:00cf	Unknown	384
84df:0c00:1000:00f1	Remaining	1350
84df:0c00:1000:00db	Unknown	3099

Table 5: Energy used by appliances that can be scheduled with active between timeslot 11 and 16, according to Figure 9.

pared to the original input, remains the schedule an improvement as the peaks are gone.

Data set C ends up with an improvement of 56,4% over the original input as shown in Figure 19. The energy consumption in the original input stays well below the average energy consumption until timeslot 14 (see the green dotted line in Figure 13a). In the following timeslots we see a clear increase in energy consumption, with a maximum of more than 1000 watt in timeslot 17. The algorithm moves the appliances responsible (see Table 6) for this increase to earlier timeslots.

SMART METER MAC	CATEGORY	ENERGY USED (IN WATT)
84df:0c00:1000:09be	Wasmachine	741
84df:0c00:1000:01ff	Unknown	3236

Table 6: Energy used by appliances that can be scheduled with active between timeslot 17 and 22, according to Figure 10.

This causes the algorithm to show a higher energy consumption in the first five timeslots, as shown in Figure 13a. This also explains the high percentage of change, over 100, in Figure 19. For the remaining timeslots the cost of the algorithm stays around the average (see Figure 13a). Only in the last timeslots the algorithm has little tasks left to schedule for appliances, causing the energy consumption to move away from the average. Hence, the increase in costs starting around timeslot 15 shown in Figure 13b.

On average the algorithm shows a percentage of change of 44.4% over the original input. We also see that the percentage of change mainly depends on the amount of appliances and assigned policies causing the peak. For example, in data set A the peak is caused by a single appliance with a high energy consumption. By moving around tasks from other appliances the algorithm still achieves an improvement of 3.3%. However, when there are more appliances responsible for a peak, we see the algorithm successfully removing or at least reducing the peak, as it is the case for data set B and C.



## CONCLUSION

---

The presented work gives an algorithm and implementation for removing peaks from the energy profile for multiple households.

First we looked at similar work within the field of creating energy awareness. With the introduction of smart meters, mobile devices, and smart appliances, solutions are created to give users insight into their energy consumption. However, most of these solutions focus on a single household, rather than a group of households. Our solution focusses on multiple households and tries to minimize peaks in energy consumption of neighbourhoods.

After looking into the related work, we defined the concept for our peak minimizing algorithm. The basis of this algorithm is a data set of measurements gathered from appliances of households. For our research, the measurements in the data set span 24 hours with an interval of a hour. From the data set we create tasks, these are found using the MinMax Steady-state technique, described by Lee et al [16].

The proposed scheduling algorithm uses a PQ in combination with a BFS optimization algorithm, as proposed by Georgievski et al. [8]. We are also assigning policies to tasks in order to reduce the number of combinations possible. These combinations are generated by the scheduling algorithm when searching for the optimal schedule.

But when talking about an optimal schedule a metrics is needed for comparing schedules. In this research the metric is a cost function that describes the ratio between the peaks and average energy consumption, proposed by Mangiatordi et al. [18]. We chose for this function, because the calculation has a low impact on the complexity of the scheduling algorithm and leads to a flat energy consumption, which is after all our aim of research.

The CLSS provides households in a neighbourhood with schedules for a single day. The web application is built with the Django framework [6] and uses MongoDB [14] for storing data. The GUI provides tables and graphs showing the energy consumption before and after scheduling. A REST API allows sharing data with other devices.

Applying our solution to three different data sets showed a percentage of change up to 64,9 in the costs. Hence, the PAR was reduced in each case. We therefore conclude that it is possible to provide households in a neighbourhood with a schedule that minimizes peaks in the energy consumption and contribute to slowing down global warming.

### 6.1 FUTURE WORK

In the present work, we optimize the energy consumption profile over a day by providing households with a schedule. Currently, the impact of the schedule on the user has not been taken into account. The influence of the users on the schedule is limited, considering no parameters are available that can be set by the user, therefore limiting the influence users have on the scheduling algorithm. By adding the option for custom policies, the household could specify when an appliances may be scheduled. In this way, the impact on his or her comfort is reduced. We also think that tasks with high energy consumption have to be scheduled first. As tasks with small energy consumption have less impact on the cost. In consequence scheduling tasks with high energy consumption first will make it easier for the algorithm to make a plan without peaks in the energy consumption.

Another future work could be extending the range of historical data. For the scope of this thesis we focused on a day. By looking at multiple days a pattern may emerge, which allows the program to improve the detection of peaks while at the same time making the task detection more accurate.

The algorithms presented do not require input from the consumer. However, an interesting addition would be to give the consumer more control over the scheduling process. For example creating custom policies, restricting the hours an task can be planned. As policies are already integrated, it does not require any changes to the scheduling algorithm itself.

Part IV

APPENDIX



## GRAPHICAL USER INTERFACE OF THE SYSTEM

Here we show screenshots taken from the [GUI](#) of the [CLSS](#). The first section shows the views of the *Peakfinder application*, while the second section shows those of the *Scheduler application*.

### A.1 PEAKFINDER APPLICATION

Cooperative load scheduling tool		
Home	Households	Neighbourhoods
Control Panel	Logout	

Overview of households

ID	Postalcode	Smartbridge Eui64
532874d7a2b83a064cb014c6	3824WG	84df:0c00:0000:15c1
532874d8a2b83a064cb014c7	3811GB	84df:0c00:0000:15d6
532874daa2b83a064cb014c8	3824WX	84df:0c00:0000:190e
532874dba2b83a064cb014c9	3824WK	84df:0c00:0000:15c8
532874ddca2b83a064cb014ca	3824WE	84df:0c00:0000:1c3f
532874ddca2b83a064cb014cb	3824WX	84df:0c00:0000:20ca
532874dea2b83a064cb014cc	3824GB	84df:0c00:0000:185e
5328753ca2b83a064cb014cd	3824WT	84df:0c00:0000:1aca
5328753ca2b83a064cb014ce	3824WC	84df:0c00:0000:20d6
5328753da2b83a064cb014cf	3824WG	84df:0c00:0000:1c12
5328753da2b83a064cb014d0	3824GM	84df:0c00:0000:1856
5328753ea2b83a064cb014d1	3824GL	84df:0c00:0000:20cc
5328753fa2b83a064cb014d2	3824LJ	84df:0c00:0000:1a56
53287540a2b83a064cb014d3	3824GH	84df:0c00:0000:15d0
53287541a2b83a064cb014d4	3824KK	84df:0c00:0000:15e0
5328759aa2b83a064cb014d5	3824WC	84df:0c00:0000:1900
5328759ca2b83a064cb014d6	3823QN	84df:0c00:0000:1a5e
5328759da2b83a064cb014d7	3824WN	84df:0c00:0000:1c87
5328759ea2b83a064cb014d8	3824GL	84df:0c00:0000:1b12
5328759fa2b83a064cb014d9	3824CC	84df:0c00:0000:1553
532875fa2b83a064cb014da	3824WL	84df:0c00:0000:1c87
532875fa2b83a064cb014db	3824GN	84df:0c00:0000:1c0b
532875fa2b83a064cb014dc	3817GT	84df:0c00:0000:1c1e
532875fa2b83a064cb014dd	3824GS	84df:0c00:0000:1ce0
532875fa2b83a064cb014de	3812GA	84df:0c00:0000:1cb0

Page 1 of 2   Next   25 of 29 households

Figure 20: Screenshot showing a list of households.

Cooperative load scheduling tool		
Home	Households	Neighbourhoods
Control Panel	Logout	

Overview of households

ID	Postalcode	Smartbridge Eui64
532874d7a2b83a064cb014c6	3824WG	84df:0c00:0000:15c1
532874d8a2b83a064cb014c7	3811GB	84df:0c00:0000:15d6
532874daa2b83a064cb014c8	3824WX	84df:0c00:0000:190e

3 households

Figure 21: Screenshot showing a list of households for a specific neighbourhood.

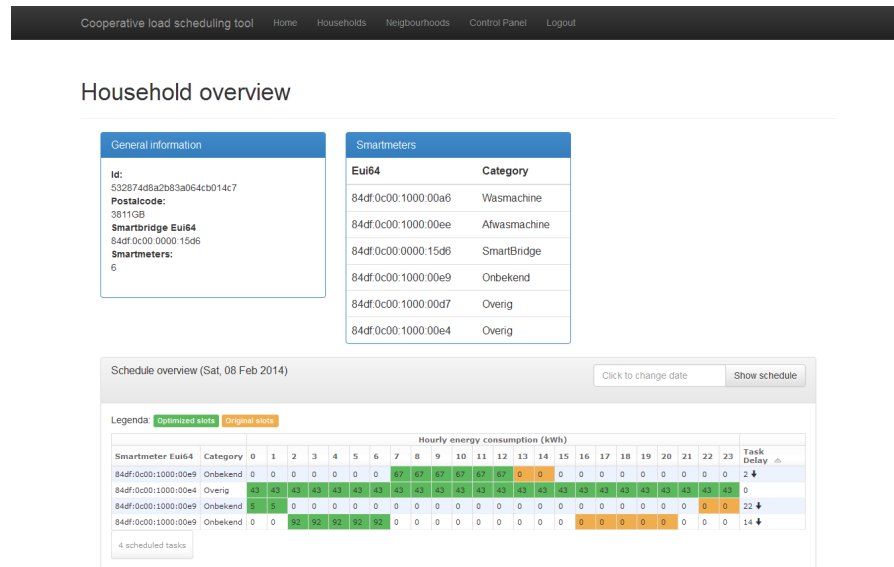


Figure 22: Screenshot showing an overview of the household.

## A.2 SCHEDULER APPLICATION

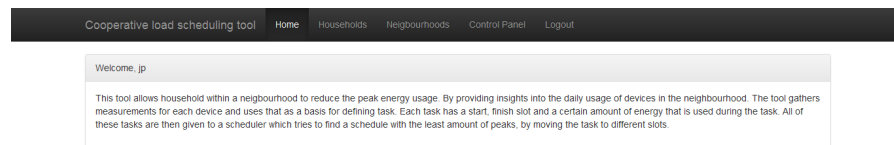


Figure 23: Screenshot showing the index page of the CLSS.

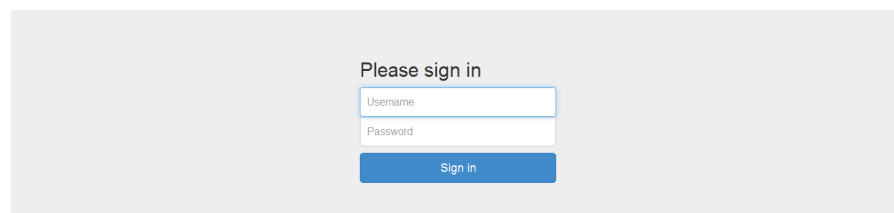


Figure 24: Screenshot showing the login page.

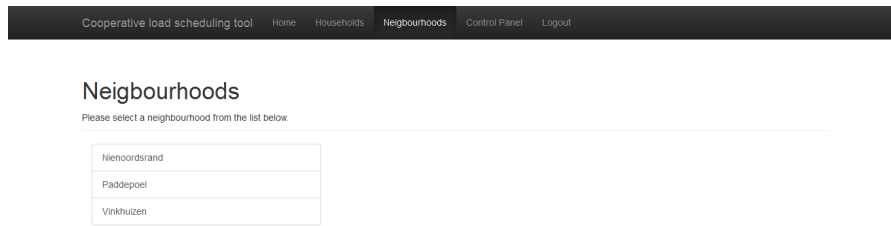


Figure 25: Screenshot showing a list with neighbourhoods.



Figure 26: Screenshot showing the details of a neighbourhood.





## BIBLIOGRAPHY

---

- [1] GreenerBuildings consortium 2013. *GreenerBuildings project website*. 2013. URL: <http://www.greenerbuildings.eu/> (visited on 06/2015).
- [2] International Energy Agency. *World Energy Outlook 2012*. November 2012. URL: <http://www.iea.org/publications/freepublications/publication/English.pdf> (visited on 02/2013).
- [3] G.T. Bellarmine. "Load management techniques." In: *Southeastcon 2000. Proceedings of the IEEE*. 2000, pages 139–145. DOI: [10.1109/SECON.2000.845449](https://doi.org/10.1109/SECON.2000.845449).
- [4] Tom Christie. *Django REST framework*. February 2015. URL: <http://www.django-rest-framework.org> (visited on 02/2015).
- [5] EnerGQ. *EnerGQ i-CARE*. 2013. URL: <http://www.energq.com/consument> (visited on 12/2013).
- [6] Django Software Foundation. *Django documentation*. September 2014. URL: <https://docs.djangoproject.com/en/1.8/> (visited on 04/2015).
- [7] The Eclipse Foundation. *Eclipse.org - Juno Simultaneous Release*. August 2015. URL: <https://eclipse.org/juno/project.php> (visited on 08/2015).
- [8] I. Georgievski et al. "Optimizing Energy Costs for Offices Connected to the Smart Grid." In: *Smart Grid, IEEE Transactions on* 3.4 (December 2012), pages 2273–2285. ISSN: 1949-3053. DOI: [10.1109/TSG.2012.2218666](https://doi.org/10.1109/TSG.2012.2218666).
- [9] E. Hammer-Lahav. *Request for Comments (RFC) 5849 - The OAuth 1.0 Protocol*. April 2010. URL: <http://tools.ietf.org/html/rfc5849> (visited on 02/2015).
- [10] D. Hardt. *RFC 6749 - The OAuth 2.0 Authorization Framework*. October 2012. URL: <http://tools.ietf.org/html/rfc6749> (visited on 02/2015).
- [11] Highcharts. *Interactive JavaScript charts for your webpage*. January 2015. URL: <http://www.highcharts.com> (visited on 04/2015).
- [12] E. Ter Horst. *Smart Grid | Rendement voor iedereen*. May 2013. URL: <http://www.smartgridrendement.nl> (visited on 05/2015).
- [13] MongoDB inc. *Map-Reduce - MongoDB Manual 2.6*. September 2014. URL: <http://docs.mongodb.org/v2.6/core/map-reduce> (visited on 07/2015).
- [14] MongoDB inc. *The MongoDB 2.6 manual*. October 2014. URL: <http://docs.mongodb.org/manual/> (visited on 02/2015).

- [15] IPCC. "Energy Systems." In: *Climate Change 2014: Mitigation of Climate Change*. Edited by O. Edenhofer et al. Cambridge, United Kingdom and New York, NY, USA: Cambridge University Press, 2014, 518–568. URL: [www.climatechange2013.org](http://www.climatechange2013.org).
- [16] Seungwoo Lee et al. "Automatic Standby Power Management Using Usage Profiling and Prediction." In: *IEEE Transactions on Human-Machine Systems* 43.6 (November 2013), pages 535–546. ISSN: 2168-2291. DOI: [10.1109/THMS.2013.2285921](https://doi.org/10.1109/THMS.2013.2285921).
- [17] Brainwy Software Ltda. *PyDev*. July 2015. URL: <http://www.pydev.org> (visited on 07/2015).
- [18] F. Mangiatordi et al. "Power consumption scheduling for residential buildings." In: *2012 11th International Conference on Environment and Electrical Engineering (EEEIC)*. May 2012, pages 926–930. DOI: [10.1109/EEEIC.2012.6221508](https://doi.org/10.1109/EEEIC.2012.6221508).
- [19] Jacob Thornton Mark Otto. *Bootstrap The world's most popular mobile-first and responsive front-end framework*. January 2014. URL: <http://getbootstrap.com/> (visited on 04/2015).
- [20] A.-H. Mohsenian-Rad et al. "Optimal and autonomous incentive-based energy consumption scheduling algorithm for smart grid." In: *Innovative Smart Grid Technologies (ISGT), 2010*. January 2010, pages 1–6. DOI: [10.1109/ISGT.2010.5434752](https://doi.org/10.1109/ISGT.2010.5434752).
- [21] Net2Grid. *Net2Grid*. 2013. URL: <https://www.net2grid.com/> (visited on 12/2013).
- [22] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.
- [23] Scrapy. *Architecture overview*. February 2015. URL: <http://doc.scrapy.org/en/latest/topics/architecture.html> (visited on 02/2015).
- [24] Scrapy. *Scrapy | A Fast and Powerful Scraping and Web Crawling Framework*. February 2015. URL: <http://scrapy.org/> (visited on 02/2015).
- [25] M. Weiser, R. Gold, and J. S. Brown. "The Origins of Ubiquitous Computing Research at PARC in the Late 1980s." In: *IBM Systems Journal* 38.4 (December 1999), pages 693–696. ISSN: 0018-8670. DOI: [10.1147/sj.384.0693](https://doi.org/10.1147/sj.384.0693).