University of Applied Sciences and Arts Northwestern Switzerland
School of Business

Master Thesis:

Learning frequent and periodic usage patterns in smart homes

Author:
D. Schweizer

Supervisors:
Prof. Dr. H. Wache
Prof. Dr. H. F. Witschel

Date:
Olten, 31 January 2014

Programme:
Master of Science in
Business Information Systems

LEARNING FREQUENT AND PERIODIC USAGE PATTERNS IN SMART HOMES

ABSTRACT

This paper discusses how usage patterns and preferences of smart home inhabitants can be learned efficiently. Such patterns as a baseline of what constitutes normal behavior of inhabitants allows future smart homes to autonomously achieve positive effects like comfort increases, energy savings or improved safety for elderly residents in assisted living homes.

The approach for learning usage patterns chosen by this research project, which was carried out as a Master Thesis at FHNW, uses frequent sequential pattern mining algorithms to mine the event data available in smart homes. While other authors have already published possible solutions or at least approaches to the problem, the information presented herein is unique because it is based solely on real-life smart home event data and not data collected in a laboratory trial and/or enriched by additional sensors. Furthermore the project does not only propose one solution but compares the performance of different algorithms regarding completeness/correctness of the results, run times as well as memory consumption and elaborates on the shortcomings of the different solutions.

To be able to solve the challenge of learning usage patterns, this project followed the research onion framework by Saunders, et al. (2009) and the design science research paradigm by Vaishnavi & Kuechler (2004): after a research design and a literature review was done, the available secondary data was analyzed in depth before different solutions (including a brute-force algorithm specifically designed for this project as well as adaptations of the three established frequent sequential pattern mining algorithms PrefixSpan, BIDE+ and GapBIDE) were designed, implemented as prototypes in Java and benchmarked against each other.

The main findings of the benchmarking done with the prototypes and of the project as such were:

- With all four algorithms a reasonable amount of frequent sequential patterns can be found with an input parameter set of pattern length = 2-5 events, minimum support = 0.01 – 0.001, overlapping patterns, wildcarding deactivated.
- The traditional frequent sequential pattern mining algorithms like PrefixSpan, BIDE+ or GapBIDE need pre- and post-processing to be able to mine smart home event data. Additionally, if different minimum and maximum lengths of patterns shall be mined, those algorithms need to be run multiple times to report the correct support count.
- The run times vary greatly for the different algorithms, BIDE+ being the slowest of the four algorithms. Both GapBIDE and especially PrefixSpan run significantly faster, however, they are outperformed by the brute-force algorithm WSDD developed for this project.
- Wildcarding could not fulfill the potential attributed to it at the beginning of the project because no significantly higher support counts can be found with wildcarding being activated.
- While WSDD as the fastest algorithms can be recommended without reservations regarding run times, all four benchmarked algorithms showed bad results regarding memory consumption for certain combinations of input parameters. This paper therefore contains six different propositions for lowering the memory consumption, should memory consumption be a concern.

- While the aspect of periodic sequential pattern mining was investigated as part of this research project and a manual analysis of the available data showed that periodic patterns exist in smart home events, no satisfactory mining results could be achieved and it is therefore suggested to look into this aspect in a follow-up research project (e.g. by adapting a state-of-the-art periodic (sequential) pattern mining algorithm to the specifics of smart home event data).

Overall it can be said that the thesis could prove part of its statement as true: it is possible to learn, in a run-time efficient way, frequent usage patterns and preferences of smart home inhabitants only from event data available within a smart home. With different use cases it could be shown in a theoretical way that the patterns can be used to achieve the aforementioned positive effects like saving electricity or increasing the comfort of smart home inhabitants.

ACKNOWLEDGEMENTS

The author of this paper would like to acknowledge and thank the following persons for their valuable insights and contributions to this project. Without them it would not have been possible to bring this project to a successful conclusion:

- Prof. Dr. Holger Wache from FHNW
- Prof. Dr. Hans Friedrich Witschel from FHNW
- Miguel Rodriguez from aizo
- Andrea Helfenstein from aizo

CONTENTS

1 INTRODUCTION

1.1 INTRODUCTION TO CHAPTER 1

This first chapter is the introduction to the master thesis at hand. As its title "Learning frequent and periodic usage patterns in smart homes" suggests, this research project deals with a topic from the field of smart homes. Smart homes allow more comfortable habitations and therefore also more comfortable lives through automation, e.g. by allowing a central and remote management, monitoring and control of appliances of a building. This is achieved via „a communication network that connects the […] electrical appliances" (Intertek and the Department of Trade and Industry, 2003).

However, for such automation to work autonomously, a smart home has to know what the normal behavior of its inhabitants is and what preferences they have. One way for a smart home to learn normal behavior, is to analyze the available power consumption and event data and mine it for frequent and periodic usage patterns. The following subchapters are going to elaborate, on a high level, how this can be realized.

1.2 BACKGROUND

To achieve a positive effect like increasing the comfort of smart home inhabitants or to reduce the needed electricity in a smart home, it is necessary for a smart home to be able to predict future needs of the smart home inhabitants by applying rules. An example for such a rule could be: *if the light was turned on in the bed room, the venetian blinds were opened in the bed room and the light was turned on in the bath room, the light in the bed room can be turned off*. While such rules can be manually specified by the smart home inhabitants, this is a tedious task. A study by CKW (2012) substantiates that a manual specification of such rules is not desirable because of the high degree of interaction with and involvement of the inhabitants needed. Therefore, instead of only having a centrally but manually managed smart home, it would be better to have a fully automated smart home that e.g. minimizes the power consumption or maximizes the comfort of the inhabitants (or any other desirable positive effect within a smart home) by itself.

However, before the aforementioned rules can be derived and applied, it is necessary for a smart home to know the smart home inhabitants' preferences and usage patterns. These patterns are needed by the smart home to decide what constitutes as normal behavior. To recognize patterns, the smart home would analyze the power consumption and event data provided by the used devices and appliances like HVAC systems (the abbreviation HVAC stands for Heating, Ventilation, and Air Conditioning), lightning system or venetian blinds. Combined with other information (e.g. the current as well as future energy prices or current as well as future weather conditions) the usage patterns of the inhabitants can then be used to find the solution that e.g. increases the comfort the most or uses the least energy.

A pattern in the context of this research project is a sequence of activities. One scenario could be a smart home inhabitant that wakes up and then turns on the lights in the bed room, opens the venetian blinds, takes new cloths out of the closet before leaving the bed room to go to the bath room, where he turns on the lights. Because both the lighting as well as the venetian blinds are connected to the electrical grid of a smart home, in this scenario the activities recorded by a smart home are:

- Turning on the light in the bed room.
- Opening the venetian blinds in the bed room.
- Turning on the light in the bath room.

The pattern is these activities, sometimes also called events, in the correct sequential order. This could be represented as "Turning on the light in the bed room -> Opening the venetian blinds in the bed room -> Turning on the light in the bath room". The mentioned scenario would therefore result in one pattern that consists of three activities / events.

If a smart home knows all the frequent and periodic patterns of its inhabitants, it can decide what they want to do next and therefore achieve the best results in automating tasks like switching off the HVAC systems or switching on the lighting in a certain room.

1.3 PROBLEM IDENTIFICATION

As can be seen from the above explanations, a crucial step towards a smart home capable of achieving positive effects through automation is to mine frequent and periodic patterns. That is where this research project comes into play because it addresses the issues identified above and has the goal to learn usage patterns from smart home inhabitants autonomously, i.e. without the active participation of the smart home inhabitants, by mining for frequent and periodic patterns.

All this has to be achieved only from analyzing the power consumption and event data available within a smart home and without relying on additional sensors like cameras, motion- or pressure-sensors. The restriction to only use the power consumption and event data has mainly two reasons:

- Data about power consumption and events is readily available in smart homes (at least in the smart homes from which this project received its data, see also chapter 3.2.5.1).
- The installation and maintenance of additional sensor (e.g. cameras, motion- or pressure-sensors) is costly and increases the complexity, which are two undesirable properties.

1.4 THESIS STATEMENT AND RESEARCH QUESTIONS

All the previously mentioned aspects lead to the following thesis statement:

It is possible to learn, in an efficient way, frequent and periodic usage patterns and preferences of smart home inhabitants only from event data available within a smart home.

The different parts and aspects that make up the thesis statement can be defined as follows:

- To learn means that by employing data mining technics it is possible to derive new knowledge.
- Efficiency is measured in run times and memory consumption for this research project.
- Frequent means a pattern which occurs more often than others.
- Periodic means a pattern which occurs at a constant interval.

- A usage pattern (and included in it the preference) of a smart home inhabitant is defined as a sequence of activities like switching on a certain lamp before opening a certain venetian blind.
- Event data are the recordings of every manipulation of an appliance (lamp, venetian blind, etc.) in a smart home.
- Available within a smart home means that no additional components like motion sensor need to be installed in a smart home.

To prove that the thesis statement is true, the following research questions have to be answered:

1. What exactly is a pattern and what kind of patterns exist inside the data?
2. What regularities do the patterns have?
3. How can the data be encoded for the different algorithms?
4. What are suitable methods and algorithms to find patterns in the smart home area?

1.5 RESEARCH OBJECTIVES

As mentioned before, the goal is to find a way for a machine to learn usage patterns (which contain implicitly the smart home inhabitants preferences) without the active participation of the inhabitants (i.e. the inhabitants shall not need to specify when they are getting up, leaving the house, getting home again, etc.). This master thesis focuses mainly on the technical aspects of such a software artefact, its design and especially how it analyzes and evaluates the power consumption and event data collected by the smart home. To reach this objective, software prototypes will be realized.

1.6 SCOPE AND LIMITATIONS

The focus lies on the automation of private households and ways for a smart home to decide what is normal behavior of its inhabitants by mining for frequent and periodic patterns. Additionally, the inhabitants shall not be required to specify their usage patterns and preferences, what also spares them from monitoring their energy consumption via smart meters (the electronic devices recording the current power consumption of the appliances of an electric circuit) on a regular basis.

Moreover, this research project focuses solely on the aspect of learning patterns. The deducing of rules and the realization of the final software components needed for home automation is not within the scope of this research project but has to be researched in a separate research project that can follow up this master thesis.

1.7 TIME LINE

The research project at hand follows the research process for master theses as proposed by Saunders, et al. (2009, pp. 11):

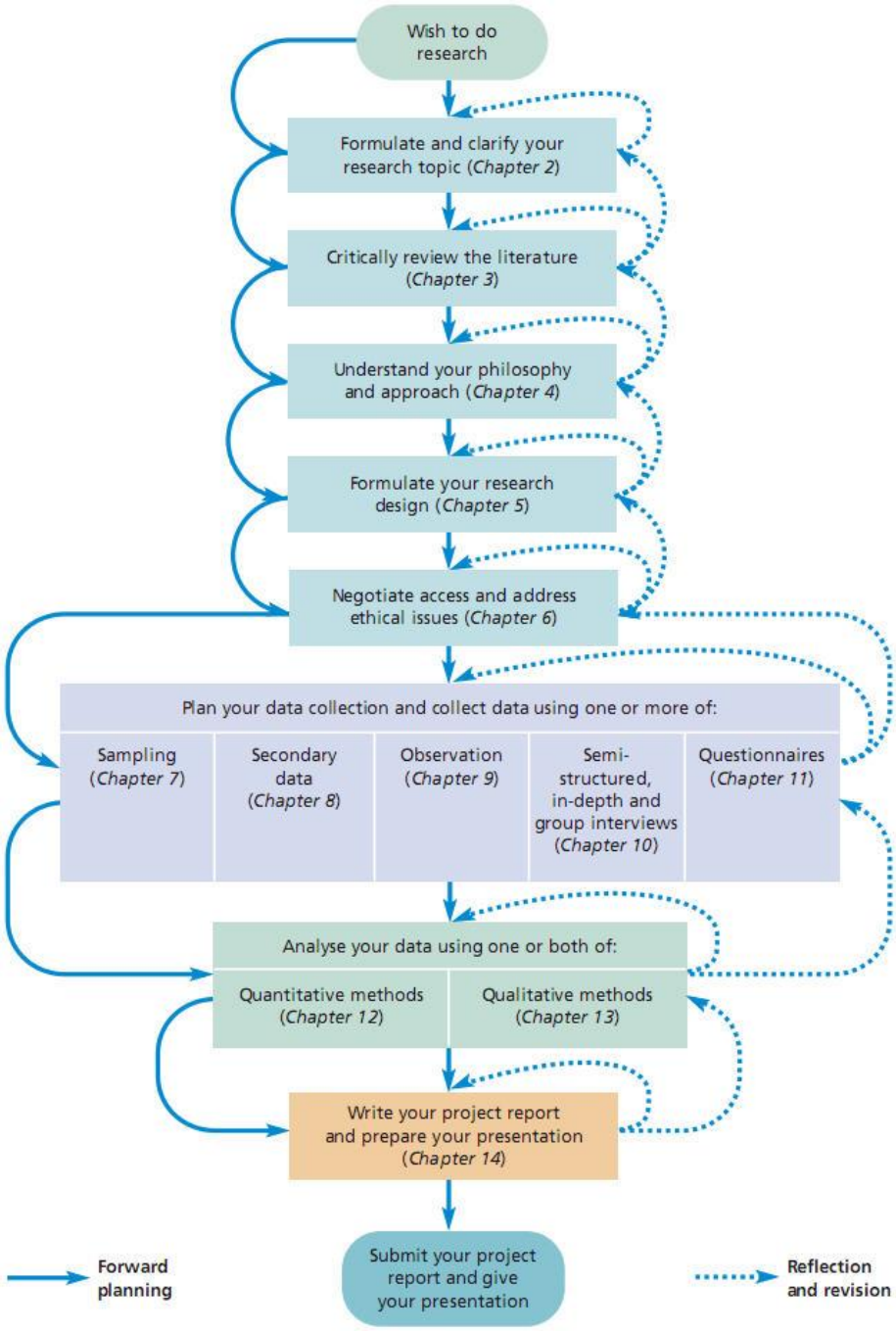


FIGURE 1: THE RESEARCH PROCESS SUITABLE FOR MASTER THESES BY SAUNDERS, ET AL. (2009).

This means that after clarifying the research topic (via the thesis statement and the scope), a literature review is done. This is followed by a study of the possible research approaches and methodologies available for a thesis.

Negotiating access to the data, planning the data collection, collect the data and analyzing the data are then the next steps. Finally, contributing the insights gained with this research project to the knowledge pool and coming up with a final conclusion of the topic by writing and submitting the project report as well as giving the final presentation are marking the end of this research project.

The time line for the research project looks as follows:

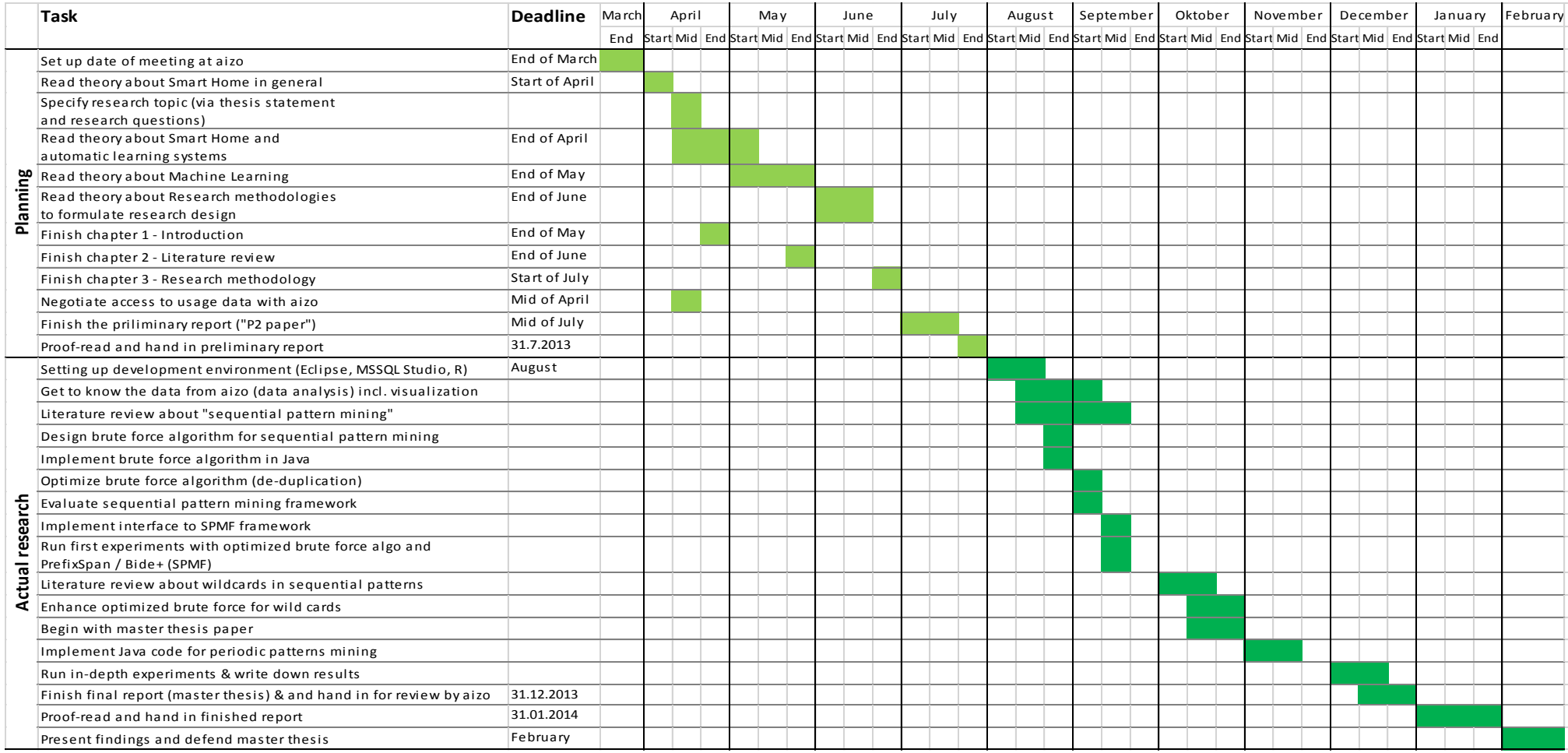


FIGURE 2: TIME LINE FOR THE RESEARCH PROJECT

1.8 STRUCTURE OF THE THESIS AND THESIS MAP

Based on the goals of this research project, the final report of the thesis is going to be structured according to the following thesis map:

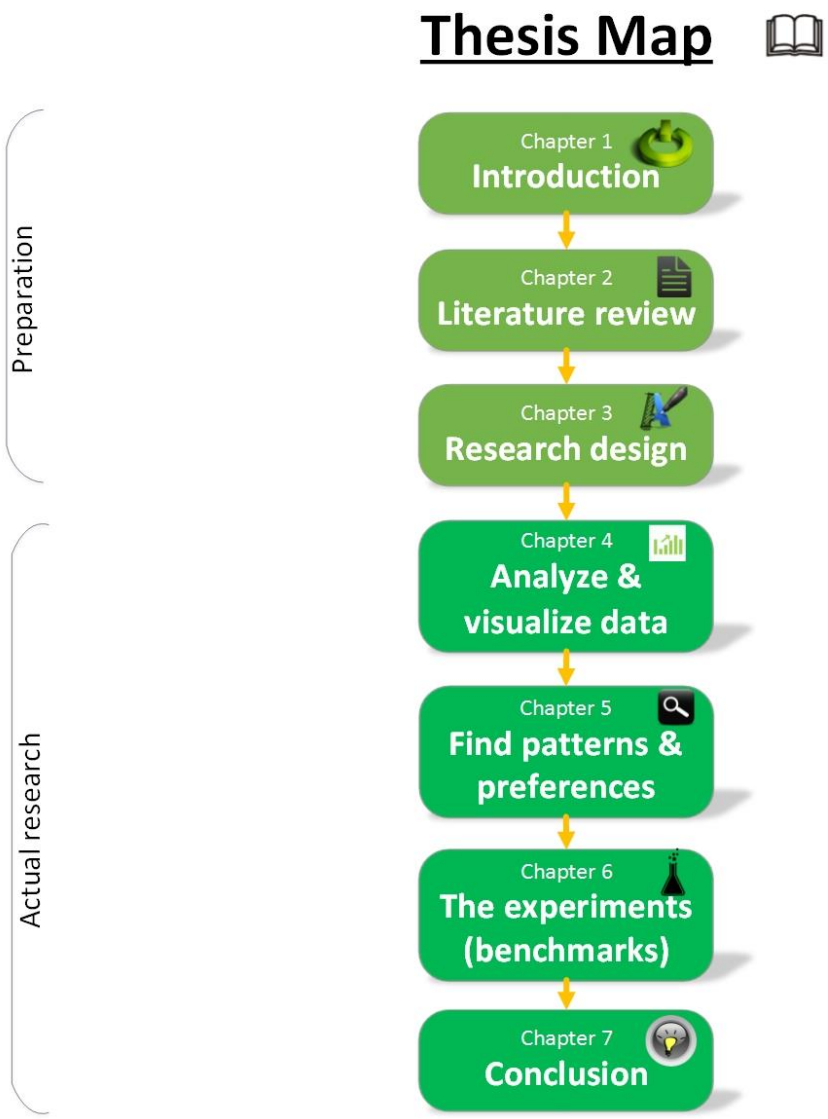


FIGURE 3: THE MAP OF THIS MASTER THESIS

The first chapter gives an introduction to the topic of the master thesis and the conditions to which it has to adhere.

The second chapter, called literature review, is going to present the available literature regarding topics like smart homes as such as well as the technics available to recognize patterns in general and especially in a smart home environment. Knowledge about these aspects are needed to be able to design and implement the software prototypes.

The chapter "research design" is next and elaborates how this master thesis and its challenges are going to be tackled. The research is designed according to the research onion proposed by Saunders, et al. (2009).

While the first three chapters deal with the preparation for the master thesis, the next chapters are concerned with the actual research and the presentation of the findings. The forth chapter is therefore called "Analyze & visualize data". It is about getting familiar with the available data and answers research questions 1 and 2 ("What kind of patterns exist?" and „What regularities do they have?"). Only after one is familiar with the data, one can make an informed decision about suitable algorithms for detecting patterns and therefore this chapter is of great significance.

The fifth chapter is concerned with finding a good way to detect patterns. The main goal is to find out how to use software to identify patterns out of a huge amount of power consumption and event data in a smart home. To achieve this, an appropriate way to decode the data (research question 3) as well as suitable methods and algorithms for finding patterns (research questions 4) are needed.

After that the next to last chapter is dealing with the experiments needed to compare the performance of the different prototypes developed. The goal behind the experiments is to benchmark the prototypes and see which performs best regarding run time, memory consumption and completeness of the patterns found.

As always with a research project such as a master thesis, the paper is going to be concluded by a summary of the work and the findings as well as a statement what the contribution to the body of scientific knowledge is. This is done in the seventh chapter called "conclusion".

1.9 CONCLUSION OF CHAPTER 1

In this first chapter the general aspects of the research project at hand are addressed. This includes the thesis statement and the research questions, which together with the definition of the scope and limitations establish the framework for the master thesis.

Additional aspects addressed in this chapter are the thesis map that provides the structure of the final report as well as a time line for the research project.

In the end it can be concluded that this research project aims at helping to enable smart homes in the future to autonomously achieve the aforementioned positive effects (like increasing the comfort or reducing the power consumption) by learning what the normal behavior of smart home inhabitants is. This is done by providing a software prototype that is able to recognize smart home inhabitants' usage patterns.

2 LITERATURE REVIEW

2.1 INTRODUCTION TO CHAPTER 2

The second chapter is looking at the available literature regarding the different topics affecting this research project. Each subchapter is addressing one of the main topics.

The literature review is an important step for a researcher to get acquainted with the research topic and to expand his / her knowledge on which he / she can build the research. Additionally it allows the reader of the report, who is not familiar with the background of the research, to get the necessary information in a short overview.

Since different researchers use slightly different names for describing identical or nearly identical things, in the following the terms learning (frequent) patterns, (frequent) pattern recognition, (frequent) pattern detection, and (frequent) itemset mining as well as subsequence mining are used as synonyms.

2.2 LITERATURE REGARDING SMART HOMES IN GENERAL

There is a lot of literature available that deals with the topic smart homes, because smart homes are no new phenomenon. They exist since the mid 80ies of the last century already and e.g. Ruth Youngblood wrote in 1986 that in a few years, smart homes would help making the lives of inhabitants easier by starting the coffee maker or washing machine when asked to. This makes obvious what the expectations were back then: enhance the comfort level of the inhabitants by introducing home automation and offering a possibility to centrally control the devices and appliances inside a habitation. When talking about appliances in the smart homes environment, normally devices for illumination and heating, cooling as well as improving the indoor air quality (summarized as HVAC) are meant. Some authors (e.g. Dounis and Caraiscos, (2009) also include appliances that deal with shading and security and / or additional devices like coffee maker or washing machine (e.g. Youngblood (1986)).

Over the years those expectations and the technologies evolved and grew more sophisticated. For a good overview of the developments, see the paper from Alam, et al. (2012) called "A Review of Smart Homes – Past, Present and Future". However, until about the year 2000 the core idea of smart homes stayed the same and only in the new millennium additional aspects came into focus, as can be seen when reading the following definition of a smart home by Intertek and the British Department of Trade and Industry from 2003: "A [smart home is a] dwelling incorporating a communications network that connects the key electrical appliances and services, and allows them to be remotely controlled, monitored or accessed". While at a first glance this still seems to deal with the aspect of home automation only, the definition (Intertek and the Department of Trade and Industry, 2003) goes on by stating benefits that a smart home can have regarding energy management like running appliances when energy is cheapest or controlling the air conditioning/heating for maximum efficiency when the house is busy or empty.

This indicates that besides raising the level of comfort, energy saving is nowadays also an aspect that can be addressed with smart home technology. A third field where smart home technology is used, is in the assisted living area (e.g. Rashidi, et al. (2011)). What all these different fields have in common is that something like a baseline of the normal behavior is needed for the smart home to be able to provide useful autonomous automation.

2.3 EVALUATION CRITERIA FOR THE ALGORITHMS

One way to get the aforementioned baseline of normal behavior is the use of the data mining task of pattern recognition. To be more specific, an algorithm for sequential and periodic pattern mining is needed, preferably one that can deal with wildcards.

Such an algorithm for pattern recognition in the smart home area should therefore fulfill the criteria listed in the following figure:

	Apriori	PrefixSpan	BIDE+	GapBIDE	MAIL	PMBC	GA
Frequent sequential pattern mining	✗	✓	✓	✓	✓	✓	✓
Periodic pattern mining	✗	✗	✗	✗	✗	✗	✗
Wildcards (1)	✗	!	!	✓	✓	✓	✓
Output wildcard position (2)	✗	✗	✗	✗	!	!	✓
No adherence to one-off condition	✓	✓	✓	✓	✗	✗	✓
Single sequence as input (3)	!	!	!	!	✓	!	!
Efficiency for smart home events (4)	!	!	!	!	!	!	!

FIGURE 4: THE EVALUATION CRITERIA FOR THE ALGORITHMS

[1] PrefixSpan and BIDE+ only support pseudo-wildcarding, which can neither be deactivated nor controlled regarding the length of the wildcards.

[2] The ability of MAIL and PMBC to output the positions of wildcards is unknown.

[3] All listed algorithms (except MAIL, which accepts a single sequence as input) can deal with the single continuous input sequence of smart home data only after it was preprocessed.

[4] For all algorithms their efficiency in real-life smart home data is unknown. Additionally, for genetic algorithms it is questionable that they find all the patterns in an efficient way.

2.4 LITERATURE REGARDING PATTERN RECOGNITION IN SMART HOMES

Using a suitable algorithm allows a smart home to learn its inhabitants' usage patterns autonomously, meaning that the inhabitants do not have to specify these things manually. This was also recognized e.g. by Rashidi and Cook (2009) who proposed in their paper "Keeping the resident in the loop: adapting the smart home to the user" to use a modified version of the Apriori algorithm to automatically detect reoccurring patterns in event data collected in smart homes.

As described by Agrawal & Srikant (1994) the Apriori algorithm was originally designed with the goal of association rule mining in mind. Association rule mining in general needs a set of transactions to find rules that can predict the occurrence of a specific item in a certain transaction. As can be seen from this description, the naming of the components of association rule mining is influenced by the shopping cart / basket analysis, for which it is often used. There, the term item means a single object, e.g. one item in a shopping basket. Analogous an item can also be one press of a button in a smart home to switch on an appliance. On the other hand a transaction, sometimes also called an itemset, is a set of items, e.g. one specific shopping basket (or in the smart home analogy a series of switching on and / or off of certain appliances, what is called a pattern in this paper). From these elaborations it can be seen that, while the Apriori algorithm gained notoriety in the field of shopping basket analysis, it can also be used for other areas than shopping basket analysis like e.g. for pattern recognition in a smart home environment.

Rules in association rule mining are always made up in the form "{X} -> {Y}", read "X implies Y". One example of such a rule could be "{Milk, Bread} -> {Butter}", which means that "Buying milk and bread implies buying butter". Such rules are always read from left to right and are not true the other way

around, or in other words the expression on the left side of the "->" is the antecedent, the expression on the right side of the „->" the consequent. Additionally, the sequence in the antecedent is not important: the above rule could also be written as "{Bread, Milk} -> {Butter}" (Agrawal & Srikant, 1994).

This property of the original Apriori algorithm, that it does not care about sequence in the antecedent, is not desirable in the smart home area because here it is important whether appliance A was started before appliance B or if it was the other way around. Therefore Rashidi and Cook (2009) only borrowed the main idea of the Apriori algorithm (which also includes the idea of pruning, i.e. excluding, infrequent patterns from the search to keep the algorithm fast) but adapted it to deal with the sequences available in the smart home data they used.

When talking about the Apriori algorithm and association rule mining in general, an important notion are support and confidence. Support is the fraction of transactions that contain both X as well as Y, while confidence describes how often the item(s) in Y appear in transactions that contain X. For both a minimum threshold, called minSup and minConf respectively, is defined and has to be fulfilled by each rule. The minSup is important for pruning: because a superset (e.g. the itemset Bread, Milk and Butter) can never be more frequent than its subset (e.g. the itemset Bread and Milk), all supersets containing an infrequent subset can be pruned / excluded from the search (Agrawal & Srikant, 1994).

Association rule mining follows a two-step approach:

1. Generate all transaction for which the following holds: support >= minSup. Those are called frequent transactions.
2. Generate all high confidence rules for each frequent transaction for which the following holds: confidence >= minConf. This second step is not applied by Rashidi and Cook (2009), who only use step 1 of Apriori for generating frequent itemsets.

Rashidi and Cook (2009) differentiate between frequent as well as periodic patterns because both are interesting for later automation. The Apriori algorithm in its original form is only able to find frequent patterns, but not periodic ones and was therefore adapted even further by Rashidi and Cook (2009). They look for patterns (transactions) that are at least two activities (items) long and then extend the length of the patterns by one until they are unable to find frequent patterns (supersets). As mentioned before, not only frequency but also periodicity is dealt with. For that Rashidi and Cook (2009) record for every pattern a tentative periodicity, which is put on a tentative list, and check for each additionally instance of an activity found inside the event data whether or not the tentative periodicity / periodicities apply or not. If not, they add another tentative periodicity to the list. In the end, only tentative periodicities that fulfil a certain threshold are moved to a definite list and all other periodicities are dismissed. While not mentioned in the paper (Rashidi and Cook, 2009), in the eyes of the researcher the most probably interpretation is that patterns with one or more periodicities on the definite list are considered as periodic while all other patterns are dismissed (unless, of course, they are frequent).

Besides the recognition of patterns in the data, the identification of triggers is an important aspect discussed by Rashidi and Cook (2009). Triggers are special activities that are triggering a pattern: in the scenario described at the beginning of this paper, the pattern ("Turning on the light in the bed room -> Opening the venetian blinds in the bed room -> Turning on the light in the bath room") is triggered by the "waking up of the inhabitant". It is noteworthy that Rashidi and Cook not only use power line adapters to record appliances that are switched on or off but rely on additional sensors as well (e.g.

motion sensor placed all over the apartment that would be able to recognize that an inhabitant has woken up).

This requirement for additional sensors (besides the power line adapters that record the switching on and off of appliances), however, is not desirable because it requires the smart home inhabitants to invest additional resources in the acquisition and maintenance of those sensors. It would be better to rely only on a) the power consumption data or b) the events reported by the power line adapters placed in the power line before each appliance in a smart home. This is the approach chosen for the research project at hand.

Regarding the topic of pattern recognition in smart homes it is worth mentioning that there exists additional literature, besides the papers mentioned so far, which deals with pattern recognition in the smart home area. For example Furth (2005) reported that the TAHI project in the UK successfully used "simple ambient lighting that glowed red if it detected an unusual pattern of activity in the home of an elderly [person]". DeMaria (2002) mentions in his article "Home smart home" that it is possible to "run scripts on […] timers [controlling different appliances] with randomizer functions to simulate a person moving around the house to different rooms, turning lights and appliances on and off […]" and that "[…] some intelligent timers can learn […] usage patterns and simulate typical usage […]". Unfortunately, both authors withhold information about how such pattern recognition and rule execution is done. This is also true for Dounis and Caraiscos (2009) who write that it is possible to "learn the comfort zone from the user's preference" but do not elaborate on how this is done. However, all these sources indicate that in theory it should be possible to learn usage patterns from smart home inhabitants, which is what this research project is focusing on.

2.5 LITERATURE REGARDING GENERIC POSSIBILITIES FOR PATTERN RECOGNITION

Because Apriori does not consider sequence in its original form and as an alternative to adapting Apriori, generic and well-known algorithms for finding frequent sequential patterns were looked at. Algorithms which are considering the sequence of patterns are known in literature as (frequent) sequential pattern mining algorithms. A wide range of such sequential pattern mining algorithm is implemented in the open-source data mining library SPMF by Fournier-Viger, et al. (2013). The implemented generic algorithms include BIDE+ (Wang & Han, 2004) and PrefixSpan (Pei, et al., 2004). Since these two algorithms are inspired by and build open the ideas of the Apriori algorithm (Agrawal & Srikant, 1994), which was described in detail earlier, they are not discussed in-depth here. What is important to know about these algorithms is that they both feature a minimum support parameter / requirement, as does Apriori, but, unlike Apriori, are able to mine sequential patterns. They are in general considered very fast and efficient sequential pattern mining algorithms (Fournier-Viger, et al., 2013), properties they achieve by using pruning technics. The details about the algorithms can be found in the original papers by Wang & Han (2004) and Pei, et al. (2004).

Another way to recognize patterns is with genetic algorithms (GA), a certain type of an evolutionary algorithm. A genetic algorithm mimics natural evolution and Darwin's paradigm of "survival of the fittest" by representing possible solutions to an optimization problem typically as an array of bits called chromosome (Holland, 1975). In a GA the possible solutions are called individuals. Via an objective function, in GA terms called fitness function, the algorithms evaluate the degree to which each individual fulfills the qualities that are important regarding the problem that has to be solved. Over

multiple generations GAs try to find the best solution by crossing parts of the chromosomes from fit individuals (that means individuals that reaching high results in the fitness function) with each other and by mutating fit individuals' chromosome in the hope that even fitter individuals result from such crossover and mutation. Normally the algorithm terminates after it has found an individual, meaning a solution, which fulfils a predefined level of fitness or after a maximum number of generations is reached, whichever happens first (Obitko, 1998).

Each genetic algorithm has to be adapted to the concrete problem it wants to solve. This is the hardest part of designing a GA and can be, depending on the problem, very difficult as Obitko (1998) mentions. The general steps involved when working with GAs are the following (Holland, 1975):

1. Setting up the GA by defining the problem representation and the fitness function.
2. Initialize the initial population via random generation.
3. Calculate the fitness of each individual and select a predefined number of pairs of individuals in a generation (different approaches exist, but in general fitter individuals have a higher chance to be selected (Obitko, 1998)). The same individual can be selected multiple times.
4. Reproduce individuals for a new generation (by crossing chromosomes of some but not all of the selected individuals and by mutating some but not all of the selected individuals). The so-called elitism GA variation specifies that the fittest solution of a generation always has to be selected and passed on to the next generation in an unchanged form (Obitko, 1998).
5. Iterate until termination (either because the maximum number of iterations is reached or a good enough solution / individual was found).

For a genetic algorithm to work, some parameters have to be predefined. Since different values for these parameters can lead to different results when running the GA, also the parameter set has to be optimized for and adapted to the concrete problem. This is normally done by running a GA multiple times and with multiple different values for the parameters. According to Obitko (1998) the parameters that have to be predefined include:

1. Size of the population / generation = number of individuals. 20-30 or 50-100, depending on the problem, have proven to be good values.
2. Probability of a mutation taking place. Should neither be 0% nor 100% but something in between, often 0.5% - 1% have proven to be good values.
3. Type of the mutation taking place.
4. Probability of crossover taking place. Should neither be 0% nor 100% but something in between, often 80%-95% or about 60%, depending on the problem, have proven to be good values.
5. Type / position of crossover taking place.
6. Selection scheme for selecting n individuals of a generation.
7. Number of generations that terminates the GA.
8. Number of generations where no further optimization takes places that terminates the GA.
9. Minimum fitness / solution quality to be attained for the GA to terminate.
10. Elitism: either yes or no or the number of fittest n individuals that are taken to the next generation in an unchanged form. It is strongly suggested to use elitism.

While genetic algorithms are normally used in optimization problems, e.g. Ghosh, et al. (2010) used them for pattern recognition, too. In their paper called "Mining frequent itemsets using genetic algorithm" they proved that finding patterns in big datasets (what they and others call frequent pattern

mining) is possible with GA and claim that their GA even outperforms the Apriori algorithm when it comes to computational complexity (Ghosh, et al., 2010, p. 140). However, while they provide a theoretical base for the assumption that the GA is faster than the Apriori, they do not include any proof in their paper that their GA implemented in Matlab really is faster than the Apriori in finding frequent patterns.

The designed GA which is able to find frequent patterns as described by Ghosh, et al. (2010) has the following characteristics and predefined parameters:

- The problem representation is done in a binary way.
- While not mentioned explicitly in the paper, the fitness function most probably is the support.
- The initial population contains 20 individuals.
- Crossover takes place randomly inside the chromosomes.
- The probability that mutation takes place is 5%.
- Neither the number of individuals selected from each generation for crossover / mutation (which most probably means, that the population size stays the same as the initial population), the method for this selection, the percentage of crossover taking place, the termination criterion nor the choice about elitism is given in the paper.

A similar approach was chosen by Keshavamurthy, et al. (2013), who describe their GA in more detail:

- The problem representation is done in a binary way.
- The fitness function is the support.
- The size of the initial population is not mentioned.
- The selection of the initial population is done by roulette wheel selection (one kind of the different available selection options).
- Crossover takes randomly place inside the chromosomes (uniform multipoint crossover): the two offsprings' chromosomes are made by copying the first parent's first part of the chromosome to the first offspring, the second parent's second part of the chromosome to the first offspring and vice-versa for the second offspring.
- The probability that crossover takes place is 75%.
- The probability that multi-point non-uniform mutation (one kind of the different available mutation options) takes place is 2.5% or 9.5% respectively, depending on whether the chromosome's fitness value is above or below the minimum support.
- The termination criterion is a not specifically defined limit of stall generations.
- Neither the number of individuals selected from each generation for crossover / mutation (which most probably means, that the population size stays the same as the initial population) nor the choice about elitism is given in the paper

The high-level overview of the GA proposed by Keshavamurthy, et al. (2013) looks as follows:

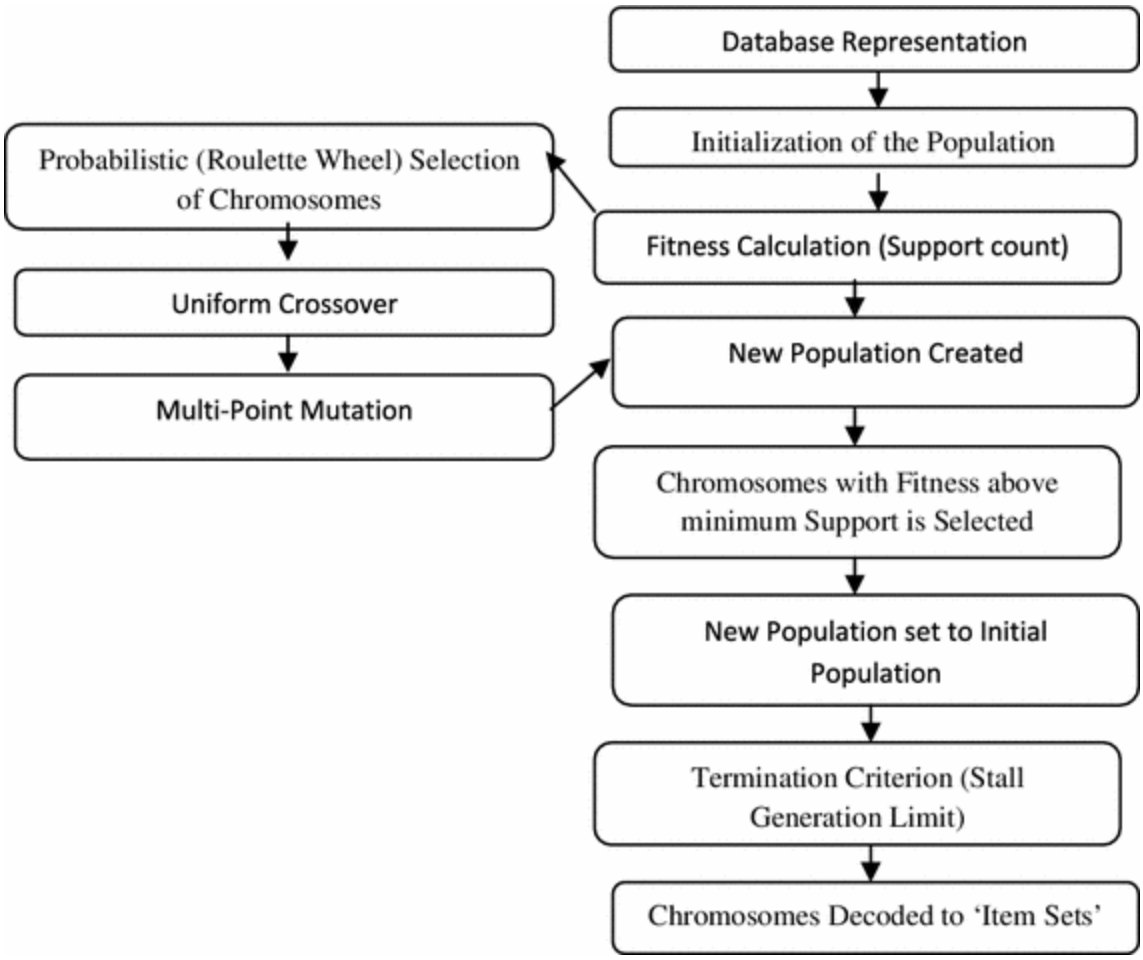


FIGURE 5: HIGH-LEVEL OVERVIEW OF KESHAVAMURTHY, ET AL.'S GA FOR FREQUENT PATTERN MINING

Unfortunately, the paper by Keshavamurthy, et al. (2013) mixes up the two terms "minimum support (minSup)" and "support count", which are two different things: for example it is written that they "[…] compared the genetic algorithm with Apriori algorithm using different support counts" and it is claimed that they used 0.3 and 0.35 respectively. However, these cannot be figures for support count because support count can only be an unsigned integer. Additionally it is not elaborated what the step "Chromosomes with Fitness above minimum Support is Selected" means (which is curious because this step seems to be a repetition of the step "Probabilistic (Roulette Wheel) Selection of Chromosomes" done early in the algorithm) and there is also no hard evidence provided for how a heuristic algorithm like GA can find all the possible frequent patterns (which in theory is a typical quality only deterministic algorithms like Apriori possess). Because of that, there is some doubt about the credibility of the work as such and especially about the claim that Keshavamurthy, et al.'s GA is working for frequent pattern recognition.

Additionally, it is worth mentioning that the GA of Keshavamurthy, et al. (2013) was not applied to data sets from smart home appliances but e.g. shopping baskets and other large data sets. This is also true for the GA of Ghosh, et al. (2010). However, despite the reservation regarding Keshavamurthy, et al.'s paper, it can be said that in theory GA should by a suitable candidate for frequent sequential pattern mining in the smart home environment.

In addition to the two algorithms discussed in depth up until now, Apriori and GA, other methods and algorithms, e.g. modern neural networks, can be used to find patterns in data. However, for this research project it was decided to only focus on Apriori-inspired algorithms and GA because of the limited time horizon available for the master thesis.

2.6 LITERATURE REGARDING OVERLAPPING / NON-OVERLAPPING PATTERNS

Regardless of whether a deterministic or a heurist algorithm is used for mining sequential frequent patterns, an important consideration is the choice of the type of sequence. The problem can be well illustrated for the frequent sequential pattern mining algorithms BIDE+ (Wang & Han, 2004) and PrefixSpan (Pei, et al., 2004):

- The power consumption as well as the event data in a smart home is available as a single continuous sequence which starts with the very first and ends with the last event. There are no logical units which separate the sequence.
- However, BIDE+ and PrefixSpan as well as all the other well-known frequent sequential pattern mining algorithms expect their input in the form of a so-called sequence database (Fournier-Viger, et al., 2013). In a sequence database, the events are grouped as logical units (e.g. each shopping basket would be such a logical unit).

Such a sequence database as "[…] a set of sequences where each sequence [… can be] a list of itemsets" (Fournier-Viger, et al., 2013) could look for example like this:

ID	Sequences
S1	(1), (1 2 3), (1 3), (4), (3 6)
S2	(1 4), (3), (2 3), (1 5)
S3	(5 6), (1 2), (4 6), (3), (2)
S4	(5), (7), (1 6), (3), (2), (3)

FIGURE 6: EXAMPLE OF A SEQUENCE DATABASE, SOURCE: FOURNIER-VIGER, ET AL. (2013)

Here S1, S2, S3 and S4 are the four different sequences, consisting of four (S2), five (S1 and S3) and six (S4) itemsets. Each itemset is delimited by parentheses () and a comma. For example the itemset (1 2 3) consists of the three individual items 1, 2 and 3.

To get from a continuous sequence of data, like it is available for this research project, to a sequence database, Chikhaoui, et al. (2010) propose to generate either overlapping or non-overlapping subsequences. The generation of non-overlapping subsequences can be illustrated with the following example of 20 events:

(100:200) -> (300:400) -> (500:600) -> (700:800) -> (1100:1200) -> (100:200) -> (500:600) -> (300:400) -> (900:1000) -> (500:600) -> (1100:1200) -> (100:200) -> (700:800) -> (500:600) -> (300:400) -> (1100:1200) -> (1300:1400) -> (100:200) -> (500:600) -> (300:400)

These 20 events would result in four non-overlapped sequences of five events (itemsets) each:

ID	Sequences
S1	(100:200), (300:400), (500:600), (700:800), (1100:1200)
S2	(100:200), (500:600), (300:400), (900:1000), (500:600)
S3	(1100:1200), (100:200), (700:800), (500:600), (300:400)
S4	(1100:1200), (1300:1400), (100:200), (500:600), (300:400)

FIGURE 7: A SEQUENCE DATABASE WITH NON-OVERLAPPED SMART HOME EVENTS

The specialty of the Aizo smart home data, that individual events can always be ordered chronologically, results in lists of itemsets which consist always of itemsets that contain only one item (= event).

As mentioned, Chikhaoui, et al (2010) distinguish two different ways to build a sequence database out of the one continuous sequence of smart home data:

- Overlapping patterns / sequences.
- Non-Overlapping patterns / sequences.

The differences between these two kinds can be illustrated with the following figure of some sample 5-events patterns:

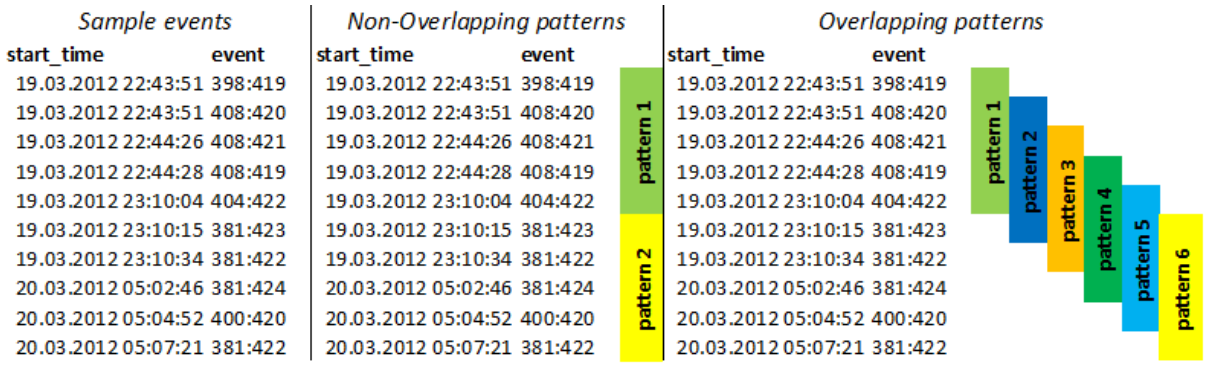
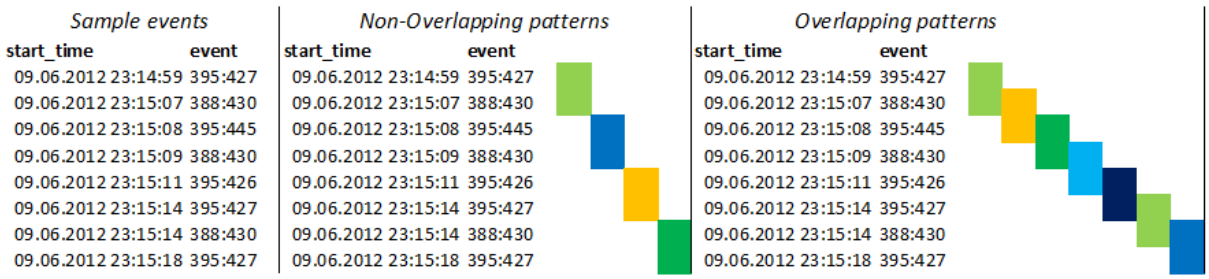


FIGURE 8: THE DIFFERENCE BETWEEN OVERLAPPING AND NON-OVERLAPPING PATTERNS

As can be seen from this example, the same events result in a different number of subsequences / patterns, depending on whether they are preprocessed as overlapping or as non-overlapping patterns. While non-overlapping patterns have the advantage that fewer of them exist, which makes the algorithm work faster because it has to process less input, they have the disadvantage that certain patterns are missing. In the above examples these are the overlapping patterns 2 to 5:

- 408:420, 408:421, 408:419, 404:422, 381:423
- 408:421, 408:419, 404:422, 381:423, 381:422
- 408:419, 404:422, 381:423, 381:422, 381:424
- 404:422, 381:423, 381:422, 381:424, 400:420

For the mining process this means that if non-overlapping patterns are produced, the calculated support (count) could be wrong because certain patterns are missed. This can be illustrated with the following example:



Sample events

start_time	event
09.06.2012 23:14:59	395:427
09.06.2012 23:15:07	388:430
09.06.2012 23:15:08	395:445
09.06.2012 23:15:09	388:430
09.06.2012 23:15:11	395:426
09.06.2012 23:15:14	395:427
09.06.2012 23:15:14	388:430
09.06.2012 23:15:18	395:427

Non-Overlapping patterns

start_time	event
09.06.2012 23:14:59	395:427
09.06.2012 23:15:07	388:430
09.06.2012 23:15:08	395:445
09.06.2012 23:15:09	388:430
09.06.2012 23:15:11	395:426
09.06.2012 23:15:14	395:427
09.06.2012 23:15:14	388:430
09.06.2012 23:15:18	395:427

Overlapping patterns

start_time	event
09.06.2012 23:14:59	395:427
09.06.2012 23:15:07	388:430
09.06.2012 23:15:08	395:445
09.06.2012 23:15:09	388:430
09.06.2012 23:15:11	395:426
09.06.2012 23:15:14	395:427
09.06.2012 23:15:14	388:430
09.06.2012 23:15:18	395:427

FIGURE 9: EXAMPLE OF A MISSED PATTERN BY NON-OVERLAPPING PATTERNS

As can be seen in the figure above, the first pattern (395:427, 388:430) occurs only once as non-overlapping pattern but twice as overlapping pattern. For that reason, if the algorithm is able to deal with the higher amount of patterns, for the area of smart homes overlapping patterns should be preferred over non-overlapping patterns. Only if overlapping patterns are produced from the smart home event data it is assured that all frequent events are found and the support (count) is calculated correctly.

2.7 LITERATURE REGARDING CLOSED / OPEN PATTERNS

Another difference, which has to be considered when dealing with the different frequent sequential pattern mining algorithms, is the one of closed versus open patterns. A good explanation what closed sequential patterns are is provided by (Fournier-Viger, et al., 2013): "A closed sequential pattern is a sequential pattern […] that […] is not strictly included in another pattern having the same support [count]".

For example, there are three sequential patterns SP1 (4 events), SP2 (3 events) and SP3 (3 events) with the following support counts found by a certain sequential pattern mining algorithm:

ID	Event 1	Event 2	Event 3	Event 4	Support count	Closed?
SP1	100:200	300:400	500:600	700:800	13	Yes
SP2	300:400	500:600	700:800	---	13	No, see SP1
SP3	100:200	300:400	500:600	---	15	Yes

TABLE 1: COMPARISON OF CLOSED AND OPEN PATTERNS

From those three open patterns, only SP1 and SP3 are also closed patterns because SP2 is strictly included in SP1 *and* has the same support as SP1. In other words, an open sequential mining algorithm like PrefixSpan would return all three patterns while a closed sequential mining algorithm like BIDE+ would only return SP1 and SP3.

As can be seen from those elaborations, closed sequential pattern mining algorithms normally need less memory because they return less patterns. Additionally, Fournier-Viger, et al. (2013) state that learning closed sequential patterns is often more efficient, meaning less CPU-intensive and therefore faster than discovering all, meaning also the open, patterns. However, from the benchmarks done with the smart home data for this project (see chapter 6) it can be seen that this is not always the case.

2.8 LITERATURE REGARDING WILDCARDED PATTERNS

The idea behind wildcarding patterns is that there could exist interesting patterns which are very similar but vary in a few events (e.g. one or two) and are not considered frequent because of this variation. If a wildcard would replace the event, which is varying, the frequency (the support count) would increase

and the wildcarded pattern could become frequent. This fact can be illustrated with the following example:

Pattern	Event 1	Event 2	Event 3	Event 4	Event 5	Event 6	Support Count
1	100:200	300:400	500:600	700:800	900:1000	100:201	10
2	100:200	300:400	500:600	700:801	900:1000	100:201	10
3	100:200	300:400	500:600	700:802	900:1000	100:201	10
4	100:200	300:400	500:600	*	900:1000	100:201	30

TABLE 2: THE INCREASE OF THE SUPPORT COUNT FOR WILDCARDED PATTERNS

As can be seen, the 6-events patterns 1 – 3 vary only in event 4 but are otherwise identical. If this fourth event would be replaced by a wildcard, in the example the asterisk *, the new wildcarded pattern 4 is created. The support count of pattern 4 is three times as high as the support count for the original patterns 1 - 3. This increase could make a wildcarded pattern a frequent pattern while the original, non-wildcarded patterns themselves are not frequent.

There exist different sequential pattern mining algorithms which offer such wildcarding capabilities, e.g. MAIL by Xie, et al. (2010) and PMBC by Wu, et al. (2013), which both adhere to the one-off condition, or GapBIDE by Li, et al. (2012). The one-off condition as described by Wu, et al. (2013) is not interesting for mining smart home event data. Therefore out of the three wildcarding algorithms mentioned before, GapBIDE, an adaption and enhancement of the previously mentioned BIDE+ algorithm, is the most interesting one for smart home event mining.

2.9 LITERATURE REGARDING MULTI-AGENT SMART HOMES

Since it is not enough to only learn frequent and periodic sequential patterns from the smart home inhabitants to achieve the aforementioned positive effects, the mining algorithm has to be implemented in what Wang, et al. (2010) and (2012) call an "intelligent multi-agent control system". As the name suggests, such a system is composed of multiple agents that each have a distinctive function. The switch agent for example decides when to get the power from the electrical grid ("grid-connected mode") and when to disconnect from it to receive the power from a power producer attached directly to the smart home like solar panels ("islanded mode"). The intelligence of such a multi-agent system lies in a central coordinator agent which can process outdoor information and inhabitants' preferences to keep the power consumption low and the comfort level high.

While Wang, et al. (2012) suggest that manually specified preferences are used, this project wants to analyze if and how such preferences in the form of frequent and periodic patterns can also be gained autonomously through frequent and periodic sequential pattern mining.

2.10 THEORY ABOUT DATA ANALYSIS AND VISUALISATION

One aspect, which is not addressed by any of the papers mentioned so far, has been brought up by Prof. Dr. Ott (2013) in a discussion about the suitability of algorithms for pattern detection in the smart home environment. He mentioned that the choice of the algorithm should not only be made according to the general needs regarding an algorithm (e.g. pattern recognition) or the field of application (e.g. smart homes) but also based on what the data available for a research project looks like and what information it contains.

For this research project it has been decided that a detailed analysis of the data (including a visualization) shall be done before choosing an algorithm, ensuring that only suitable algorithms will be chosen.

2.11 CONCLUSION OF CHAPTER 2

In this chapter the available literature regarding smart homes and their potential to learn what constitutes as normal behavior of smart home inhabitants was looked at. To achieve that, todays smart homes have to become even smarter and recognize their inhabitants' usage patterns of the various appliances. It was recognized that specifying preferences and usage patterns manually is not a good approach and that machine learning in the form of pattern recognition and sequential and periodic pattern mining is needed.

While the literature review showed that a lot of research was and still is being done in the field of smart homes and pattern recognition, there are mainly two aspects that make this research project unique:

1. The projects mentioned in the literature review analyzed only few data for a small period of time or from laboratory installations or data not gathered in smart homes. This means that the efficiency of those algorithms is unknown for mining real-life smart home data. This project on the other hand is analyzing, per smart home, a big amount of real-life data collected over a period of eight months (see also chapter 3.2.5.1). Therefore the algorithms chosen for finding patterns in this research project need to be able to deal with a lot of smart home data in an efficient way. This means that this project has a higher significance regarding real-world applicability of pattern detection in smart homes.
2. Unlike the other projects mentioned in the literature review, this project is finding patterns in data that was collected solely from the usage of appliances. Other projects were enriching the data with additional sensors (e.g. motion sensors) used in the smart home. The installation of such additional sensors is costly and therefore this project wants also to find out, whether or not it is possible to find useful patterns only inside the usage data of appliances and the events like switching on and off that are recorded in a smart home.

These two aspects mentioned above differentiate this project from other work done in the smart home area. Other aspects of the project, however, are influenced by the propositions of the various papers discussed in this chapters. These are mainly the following aspects:

- The idea of a multi-agent system is borrowed from Wang, et al. (2012). One possible scenario is that the multi-agent system is enhanced by an additional agent that learns the patterns and preferences of the smart home inhabitants.
- Different researchers like Ghosh, et al., (2010) or Rashidi and Cook (2009) showed that in principal different algorithms are suitable for finding frequent patterns: while Ghosh, et al., (2010) used a GA, Rashidi and Cook (2009) applied the principles of the Apriori algorithm to the respective data available for their projects.

In summary it can be said that what Jahn, et al. (2010) suggest as a next step, that "as smart homes become even more smart, systems could learn over the time and calculate the most efficient ways to configure the home appliance", is the goal of this research project: to make smart homes smarter by letting them learn the usage patterns of their inhabitants over time.

The choice of the algorithms used for this project shall be influenced by the following criteria:

- The algorithm has to be able to find patterns in data. To be more precise it has to find
  - frequent sequential patterns and
  - periodic (sequential) patterns.
- It has to be able to deal with wildcards, which should include the possibility
  - to find wildcarded patterns and
  - to output where the wildcard is positioned in the pattern.
- The algorithm should not adhere to the one-off condition.
- The algorithm needs to be able to process the single continuous sequence of data available in a smart home.
- It must be able to deal with a big amount of data, since this project is dealing with a big amount of real-life data.

These criteria and the suitability of the different algorithms can be summarized as follows:

	Apriori	PrefixSpan	BIDE+	GapBIDE	MAIL	PMBC	GA
Frequent sequential pattern mining	✗	✓	✓	✓	✓	✓	✓
Periodic pattern mining	✗	✗	✗	✗	✗	✗	✗
Wildcards (1)	✗	!	!	✓	✓	✓	✓
Output wildcard position (2)	✗	✗	✗	✗	!	!	✓
No adherence to one-off condition	✓	✓	✓	✓	✗	✗	✓
Single sequence as input (3)	!	!	!	!	✓	!	!
Efficiency for smart home events (4)	!	!	!	!	!	!	!

FIGURE 10: THE EVALUATION CRITERIA FOR THE ALGORITHMS

[1] PrefixSpan and BIDE+ only support pseudo-wildcarding, which can neither be deactivated nor controlled regarding the length of the wildcards.

[2] The ability of MAIL and PMBC to output the positions of wildcards is unknown.

[3] All listed algorithms (except MAIL, which accepts a single sequence as input) can deal with the single continuous input sequence of smart home data only after it was preprocessed.

[4] For all algorithms their efficiency in real-life smart home data is unknown. Additionally, for genetic algorithms it is questionable if they can really find all the patterns in an efficient way.

In theory, these criteria can be fulfilled (to different degrees) by both, deterministic as well as heuristic algorithms. If deterministic algorithms like the Apriori-inspired algorithms prove to be too slow to mine a lot of power consumption or event data, a heuristic algorithm could be a viable option to speed up the mining process. This because a deterministic algorithm like Apriori works fundamentally different than a heuristic / stochastic algorithm like GA: the Apriori is creating a complete tree containing all the possible frequent patterns and checks for all the nodes (except for the ones that can be pruned, in Figure 11 this would be the pattern ABCD) of this tree, this means for all potential frequent patterns, whether they in fact are frequent. This is done in a predefined (deterministic) way, which means that the second node is checked after the first node, the third after the second node and so on. The GA on the other hand

is roaming around the space of all possible solutions in a random (stochastic) way and checks these random possible solutions for their support (for an illustration, see the sequence no. of the frequent pattern search in Figure 11). As a heuristic algorithm, the GA does not check all the possible frequent patterns.

This means that it is not sure that a GA finds every frequent pattern as opposed to the Apriori, which guarantees to find all frequent patterns, which is a disadvantage of GA / an advantage of Apriori. However, Ghosh, et al. (2010) hinted that it is sufficient to use GA for frequent pattern recognition. On the other hand, GA needs less memory for storing the different individuals than Apriori needs for its tree and it is possible that it consumes less CPU time, which are advantages of GA / disadvantages of Apriori.

The difference between a deterministic / the Apriori algorithm and a heuristic / a genetic algorithm when dealing with the problem of finding patterns can also be illustrated as follows:

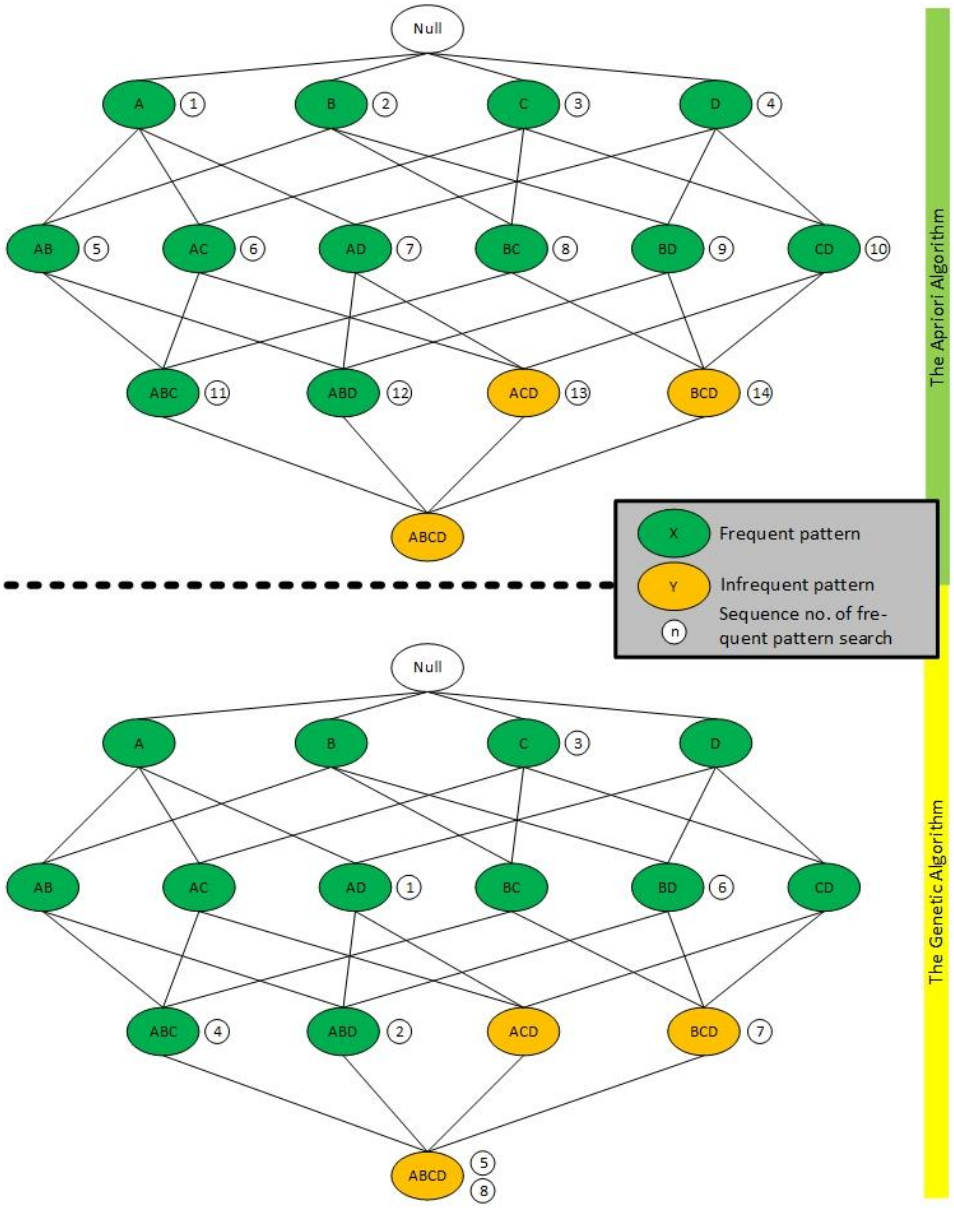


FIGURE 11: THE DIFFERENCE BETWEEN APRIORI AND GA REGARDING FINDING PATTERNS

As can be seen from the literature review done for this research project, in general both Apriori-inspired algorithms employing pruning technics (deterministic) as well as genetic algorithms (heuristic) are suitable for finding frequent patterns in large datasets. However, to look at heuristic algorithms makes only sense if the deterministic algorithms are too slow to find sequential patterns in the power consumption or event data because otherwise the aforementioned disadvantages of heuristics (mainly that the algorithm may get stuck at a local maximum and therefore not find the global maximum, which means that it does not find all patterns) outweigh their advantages. Since this project was successful at mining sequential patterns with deterministic algorithms (see chapters 5 and 6), heuristic algorithms were not looked at during the implementation.

3 RESEARCH DESIGN

3.1 INTRODUCTION TO CHAPTER 3

This third chapter is dealing with aspects of the different research designs that can be chosen for a master thesis in general and the one that was chosen for this particular thesis. Each different aspect of the research design is addressed in a separate sub-chapter.

The choice of the correct research design is important for the success of a research project and has therefore to be considered carefully.

3.2 THE RESEARCH DESIGN OF THIS MASTER THESIS

When planning and designing a research project like for example a master thesis, various aspects have to be considered. Saunders, et al. (2009, pp. 108) have grouped those aspects in their research onion:

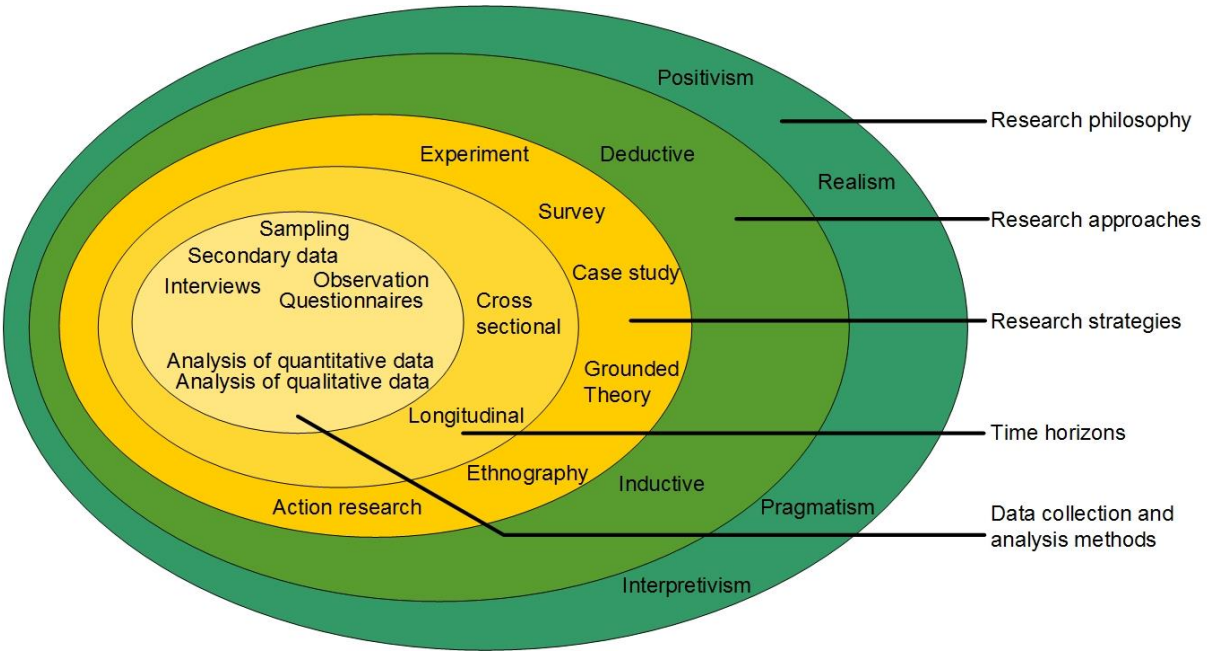


FIGURE 12: THE RESEARCH ONION, ADAPTED FROM SAUNDERS, ET AL (2009).

Each of the five dimensions of the research onion is going to be described in the following sub-chapters.

3.2.1 RESEARCH PHILOSOPHIES

As explained by Saunders, et al. (2009) the research philosophies deal with the stance a researcher can take. Classically, four different kinds of researchers are distinguished:

- Positivists
- Interpretive researchers
- Direct realists and critical realists (realism)
- Pragmatists

While a positivist is answering questions along the lines of "Are you currently unemployed?", an interpretive researcher tries to answer questions like "How do you feel about being unemployed?". In other words, a positivist believes an objective viewpoint can be taken and that only observable phenomena provide credible data while an interpretivist is more of the opinion that it is necessary to understand and pay attention to social factors, too, which results in more subjective research (Saunders, et al., 2009, pp. 113 - 119).

Realism is similar to the stance a positivist takes and "assumes a scientific approach to the development of knowledge" (Saunders, et al., 2009, pp. 114). However, while the positivist undertakes research in a value-free way, direct as well as critical realists' research is value laden and biased by the researcher's world view (Saunders, et al., 2009, pp. 119).

A pragmatist is convinced that it is possible and useful to adopt different stances in one and the same research project if different research questions ask for different philosophies (Saunders, et al., 2009, p. 109).

In addition to these four philosophies mentioned in the research onion above, there is a fifth called design science research, which was introduced by Vaishnavi and Kuechler (2004). They characterize it as „yet another 'lens' or set of analytical techniques and perspectives (complementing the Positivist and Interpretive perspectives) for performing research in information systems". Design science research is concerned with the design of artifacts, e.g. a new algorithm.

3.2.1.1 THE RESEARCH PHILOSOPHY OF THIS PROJECT

Since the goal of this research project is to design and implement prototypes that are able to detect frequent and periodic sequential usage patterns of smart home inhabitants, this research's underlying philosophy is design science research. The design-centric idea behind this philosophy is the most fitting for a research project such as this one, where the design (and development) of an artefact is the targeted end result.

3.2.2 RESEARCH APPROACHES

There are two different and opposite approaches that can be taken in research: one is the so-called deductive approach, the other is the inductive approach.

Saunders, et al. (2009, pp. 124) write that deduction "is the dominant research approach in the natural sciences". According to them deductive research normally progresses in a five step approach:

1. Deduce a hypothesis from a theory.

2. Express the hypothesis in operational terms by indicating exactly how variables are measured.
3. Test this operational hypothesis.
4. Examine the outcome.
5. If necessary, modify the theory / hypothesis in the light of the findings and repeat the whole cycle.

Induction is the opposite of deduction and can be seen as bottom-up approach (Trochim, 2006):

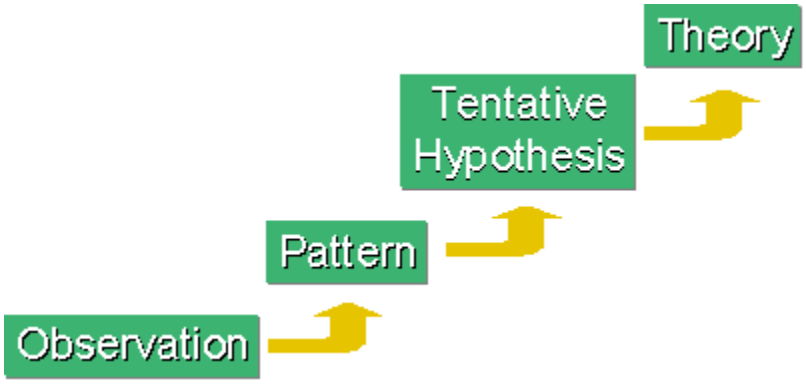


FIGURE 13: INDUCTION AS A BOTTOM-UP APPROACH ACCORDING TO TROCHIM (2006)

Induction is mainly concerned with building a theory. While deduction is a highly structured approach and deals with collecting quantitative data, induction normally collects qualitative data and offers more flexible structures in a research project (Saunders, et al., 2009, pp. 127). One of the goals of the inductive approach is to "get a feel of what is going on, so as to understand better the nature of the problem" (Saunders, et al., 2009, pp. 126).

Trochim (2006) adds that "inductive reasoning […] is more open-ended and exploratory" and notes that often research projects are neither 100% deductive nor 100% inductive but mix the two approaches: "Even though a particular study may look like it's purely deductive […,] most […] research involves both inductive and deductive reasoning processes at some time in the project".

3.2.2.1 THE RESEARCH APPROACH OF THIS PROJECT

At the beginning, this research project is following a bottom-up approach. This means that first of all the researcher needs to get a feeling for and understanding of the problem at hand. Via the detection of patterns (in the power consumption and event data), the theory, meaning working software prototypes, is derived. Since not all the important criteria can be known beforehand, the best approach is to test it out. Therefore the research approach chosen for this research project is an inductive one for the first part of the project.

However, as mentioned by Trochim (2006), also this research project is not following the inductive approach exclusively: the derived theory in form of different software prototypes is tested and benchmarked in a deductive approach in the second part of the project.

To summarize things up, it can be said that detecting the usage patterns and preferences is an inductive approach while the benchmarking of the different prototypes follows more the deductive approach.

3.2.3 RESEARCH STRATEGIES

According to the research onion, the first possible strategy are experiments. They often involve an experimental and a control group and the differences of variables are analyzed as shown in the following Figure (Saunders, et al., 2009, pp. 142):

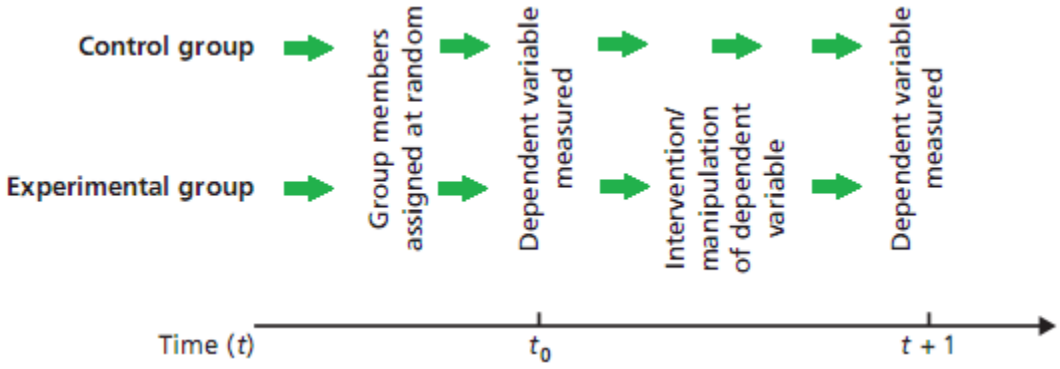


FIGURE 14: A CLASSICAL EXPERIMENT STRATEGY ACCORDING TO SAUNDERS, ET AL. (2009)

In the field of information systems one form of experiments are benchmarks to compare different solutions. Such benchmarks can either be used to measure the performance of a newly designed solution against a standard or against other variations of the new solution (e.g. designing different prototypes that implement different algorithms and then benchmark them against each other).

Another strategy are surveys where the researcher often uses questionnaires to gather data from large groups in a standardized and systematic way. This gathered data is evaluated to find patterns and relationships between variables, which then in turn have to be interpreted to e.g. build models of these relationships (Saunders, et al., 2009, pp. 144-145).

The third strategy are case studies, which are of particular interest to researchers that wish to gain a rich understanding of the context of the research and often "generate[…] answers to the question 'why?'" (Saunders, et al., 2009, p. 146). Additionally, it is likely that researches employing the case study strategy need multiple sources of data for finding answers.

Action research is yet another strategy that can be chosen. Saunders, et al. (2009, p. 147) state that "the implications of change together with those who experience the issues directly" is often studied. In the field of business information systems this could be done e.g. by acceptance tests of a newly developed software or GUI. Another very typical characteristic of action research is the collaboration of researchers with practitioners (Saunders, et al., 2009, p. 147).

Grounded theory, as described by Saunders, et al. (2009, pp. 148-140), has no initial framework regarding a theory. Rather the theory is developed from an initial data collection and then, in a next step, tested with further observations through data gathering. Grounded theory therefore can be looked at as a combination of the deductive and inductive approach.

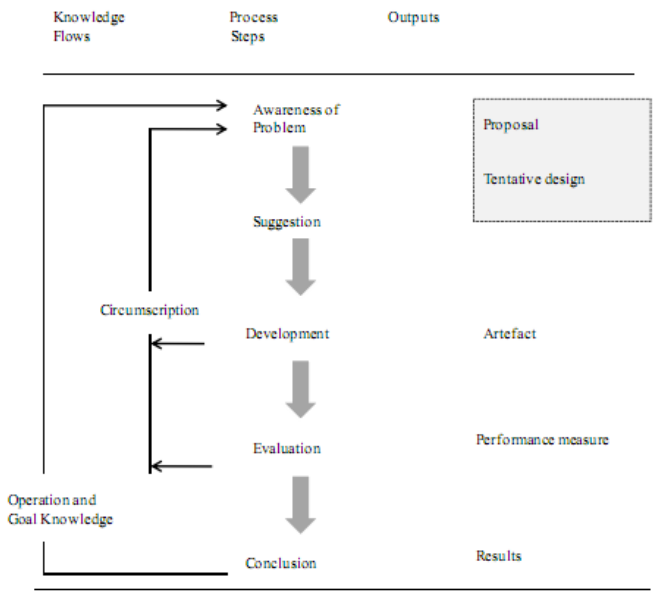
In Ethnography, which is not a very dominant research strategy according to Saunders, et al. (2009, p. 150), a researcher is trying to gain insights into the social world of the people which are involved in his or her particular research subject. Additionally, the researcher wants to understand the context of the research from the perspective of the involved people.

The final strategy mentioned in the research onion is called archival research. It is the analysis of secondary data (data originally collected for a different purpose than the archival research project) and aims at answering questions of changes over time. Saunders, et al. (2009, pp. 150) argue that missing or inaccessible data is a big problem when it comes to archival research.

In addition to the research strategies mentioned in the research onion, there is also design research. Researchers that follow the design research approach by Vaishnavi and Kuechler (2004) do the following: after becoming aware of a problem they suggest and develop a solution that is then evaluated and a conclusion is drawn.

3.2.3.1 THE RESEARCH STRATEGY OF THIS PROJECT

The research strategy of this project looks as follows: after identifying the problem (that it is necessary to have an algorithm, which is able to find frequent and periodic sequential patterns inside the big amount of power consumption and event data collected in a smart home) a tentative design is suggested and a prototype for pattern detection is developed. After that, the prototype's performance is measured and evaluated. The cycle of tentative design, development and evaluation is repeated until a satisfactory conclusion can be drawn:



Patterns in smart home data reflect the inhabitants' behavior and implicitly (to a certain degree) their preferences. To find these patterns an algorithm for sequential pattern mining is needed.

Suggest a possible tentative design, e.g. propose to use a modified version of Apriori or BIDE+ to find sequential patterns.

Build a prototype (= artefact) that is applied to the available data.

Evaluate the prototype's performance (and if necessary repeat the cycle).

Finally come to a conclusion (and if necessary repeat the cycle).

FIGURE 15: THE TYPICAL CYCLE OF DESIGN RESEARCH ACCORDING TO VAISHNAVI AND KUECHLER (2004)

As can be seen from the above elaborations, this research project follows the ideas of design research.

In addition to that, for the evaluation of the different prototypes, the benchmarking approach suggested by the research strategy "experiments" is used, which results in the combination of design research and experiments for this research project.

3.2.4 TIME HORIZONS

Regarding the time horizon, researchers can choose between two alternatives: cross-sectional or longitudinal studies. While cross-sectional studies have to be understood as a snapshot taken at a certain point in time, longitudinal studies can be compared to a diary: they record data over a longer period of time. A big benefit of longitudinal studies over cross-sectional studies is that they allow the study of changes and developments over time (Saunders, et al., 2009, p. 155).

However, the choice of a longitudinal study has the implication that it has to be started with plenty of time still available on the timeline of the research project.

3.2.4.1 THE TIME HORIZON OF THIS PROJECT

Since this project wants to find a baseline of smart home inhabitants' usage patterns and preferences it would not be appropriate to mine patterns in a dataset which spans just a few hours or days, which would be the case if a cross-sectional time horizon was chosen. The much better approach is to mine for the patterns in a data set which spans several months and therefore allows to make a meaningful statement regarding the possibility of finding the usage patterns in an autonomous way.

Additionally, the aspect of mining for periodic sequential patterns is only possible if a longitudinal horizon is chosen because certain periodic patterns, which occur seldom, could not be detected in a dataset consisting of only a short time of power consumption and event data.

Finally it can be said that the often mentioned problem of too little time for a longitudinal horizon does not apply to this project because the data already exists (for details, see the chapter 3.2.5.1).

3.2.5 DATA COLLECTION METHODS

The innermost layer of the research onion is dealing with the different data collection methods. There are mainly five different methods to consider:

- Secondary data
- Observation
- Interviews
- Questionnaires
- Sampling

When thinking about data collection, often the first thing that comes to mind is collecting new data (so-called primary data) that fits exactly to the specific needs of a certain research project. However, often secondary data, that is data already collected for some other purpose, can be very useful for researchers (Saunders, et al., 2009, p. 256). Examples of such secondary data include sales figures, minutes of meetings or usage data and can either be available publicly or only inside the boundaries of the company that collected the data. In the latter case, access to the data has to be negotiated as part of the research project (Saunders, et al., 2009, p. 257).

Another method to collect data is through so-called observation. Saunders, et al. (2009, p. 288) characterize observation as "the systematic […] recording, description, analysis and interpretation of people's behavior". This could be for example achieved via attending a meeting and observing how the participants behave.

In a research project, interviews are often used for data collection. Interviews are dialogues between people to obtain specific information. For Saunders, et al. (2009, pp. 318-342) it is advisable to prepare an interview in detail and plan things like how to record the data and information shared by the interviewee. Additionally one should be sure to ask the right kind of questions (e.g. open versus closed questions) and verify that one understands what the interviewee said (e.g. by asking probing questions or by rephrasing his / her responses). Furthermore the interview should be scheduled and, if beneficial, the questions handed out in advance. Another important aspect is the preparation of the researcher for difficult situations during the interview like participants that are unwilling to disclose information, begin to cry because they feel threatened or criticize the researcher to show off their knowledge.

A questionnaire is defined by DeVaus (2002) as "all techniques of data collection in which each person is asked to respond to the same set of questions in a predetermined order".

Since it often is not possible or practical to collect the data of an entire population (census), sampling comes into play. It allows to get a representative amount of data for a population without the need to gather all the data of the whole population. Sampling can be combined with any of the aforementioned data collection methods like interviews or questionnaires (Saunders, et al., 2009, pp. 210-213).

For much more details about the available data collection methods, see the very informative chapters 7 through 11 of Saunders, et al.'s Research Methods for Business Students (2009).

3.2.5.1 THE DATA COLLECTION METHOD OF THIS PROJECT

The data necessary for this research project was collected by Aizo, the Swiss company that produces the so-called digitalSTROM components for home automation. It is available as secondary data to this project via the signing of a non-disclosure agreement between the researcher and Aizo.

Secondary data as a data collection method and the concrete data of Aizo was chosen because it offers hard evidence collected in real-life smart homes in a large enough amount that allows to find patterns in it. This was a crucial characteristic for the data needed for this project. While interviews and especially questionnaires would have allowed to gather some of the data available as secondary data, they would have been most probably incomplete because the smart home inhabitant as a human could never have been as precise and complete as the data gathered by a machine (in this case the digitalSTROM components).

Another argument for secondary data as the collection method was the time horizon (see the chapter 3.2.4.1): while observations would have been a possibility to gain access to the needed data (e.g. by observing the day-to-day lives of smart home inhabitants) the time horizon of the master thesis of about six months would not have permitted it to gather primary data from a long enough period to be able to observe periodic patterns which have a big time span between the occurrences.

The available secondary data contains all the usage patterns and detailed electricity consumption data of about 20 private households over a period of eight months.

In addition to negotiating access to the aforementioned secondary data, the researcher wrote down what typical patterns he expected to find in the data based on his own experience and life style. This can be regarded as a special kind of interview where the interviewer answers the questions him- / herself. The answers were used in the initial manual data analysis to have a starting point for looking for patterns.

3.2.6 DATA EVALUATION METHODS

The last aspect addressed by the research onion is the data evaluation. After having defined how the data is collected with one of the aforementioned data collection methods, it is important for a researcher to think about how he / she can analyze and evaluate the available data. For that analysis two opposite strategies exist:

- Analyzing quantitative data
- Analyzing qualitative data

Saunders, et al (2009, p. 482) are of the opinion that the following three criteria are the most distinguishing features of quantitative and qualitative data:

Quantitative data	Qualitative data
Based on meanings derived from numbers	Based on meanings expressed through words
Collection results in numerical and standardized data	Collection results in non-standardized data requiring classification into categories
Analysis conducted through the use of diagrams and statistics	Analysis conducted through the use of conceptualization

TABLE 3: DIFFERENCE BETWEEN QUANTITATIVE AND QUALITATIVE DATA (SAUNDERS, ET AL., 2009)

As the name suggests, analyzing quantitative data is needed to "make [quantitative data] useful, that is, to turn them into information" (Saunders, et al., 2009, p. 414). Quantitative data is represented as numerical data and aims at generalizing empirical facts (Boutellier & Gassmann, 2010, p. 2). This is done by using tables, diagrams or statistical techniques and supported by software like Excel, R or SPSS (Saunders, et al., 2009, p. 415).

The opposite of quantitative data is qualitative data. Qualitative data refers to all non-numerical data and data that cannot be quantified, e.g. meeting notes, a list of responses or entire policy documents (Saunders, et al., 2009, p. 480) . The analysis of qualitative data can be assisted by so-called computer aided qualitative data analysis software (CAQDAS). However, Saunders, et al., (2009, p. 481) mention that such software is not available for all different research scenarios and researchers and that the analysis of qualitative data is, also nowadays, often still done without the aid of computers.

3.2.6.1 THE DATA EVALUATION METHOD OF THIS PROJECT

The secondary data from Aizo that is available for this project is clearly quantitative data. It consists of hundreds of megabytes of numerical data stored in different tables in a relational database.

However, the first part of this research project is to get acquainted with and a feeling for the problem at hand, its characteristics and what the numerical representations in the data mean in terms of usage and preferences of the Smart Home users. For that it is also necessary to have a look at only a few very specific situations and usage patterns that can be found in all the quantitative data. This few but typical patterns form a small subset of the available data are going to be analyzed thoroughly: not on their numeric representation inside the whole data collection but on what their meaning is regarding the behavior of the Smart Home inhabitants. The design and development of the different prototypes depends heavily on the insights gained from this initial analysis of qualitative data.

Additionally, after the prototypes are developed, quantitative data analysis will be done in the form of experiments and the goal is to assess the performance of the different prototypes when mining usage patterns.

Therefore this research project has both aspects, the ones from analyzing quantitative data as well as the ones from analyzing qualitative data.

3.2.7 THE RESEARCH PLAN

When summarizing all the chosen aspects, the research onion of this project looks as follows:

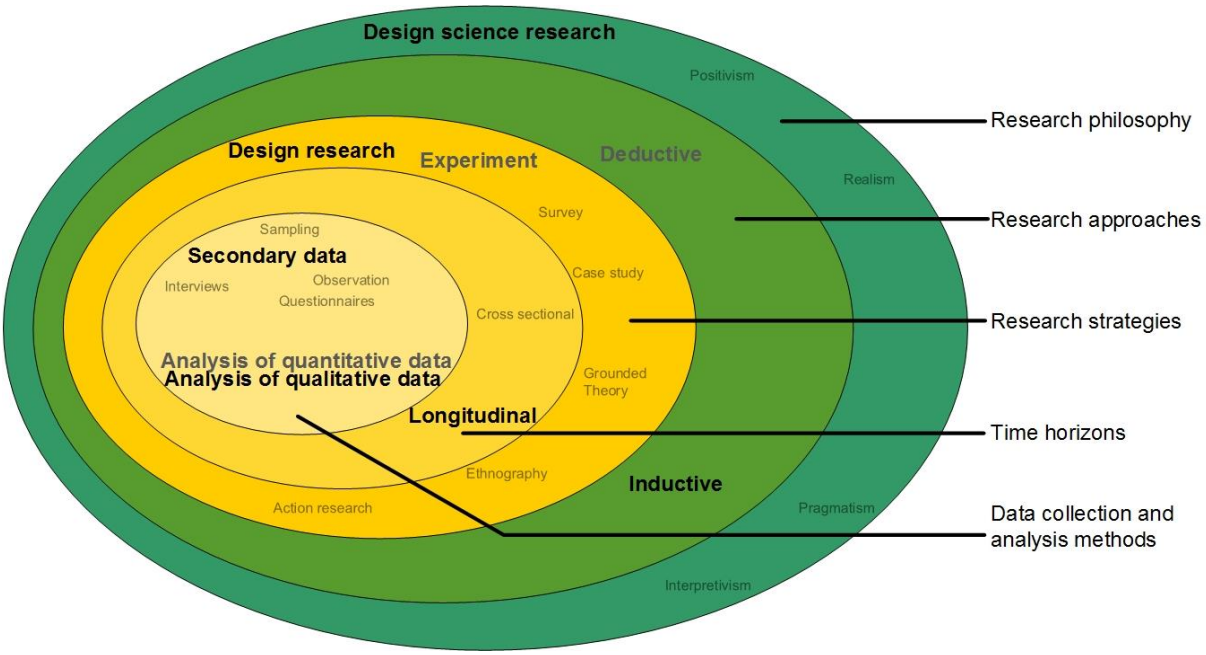


FIGURE 16: THE RESEARCH ONION OF THIS RESEARCH PROJECT

This in turn leads to the following research plan:

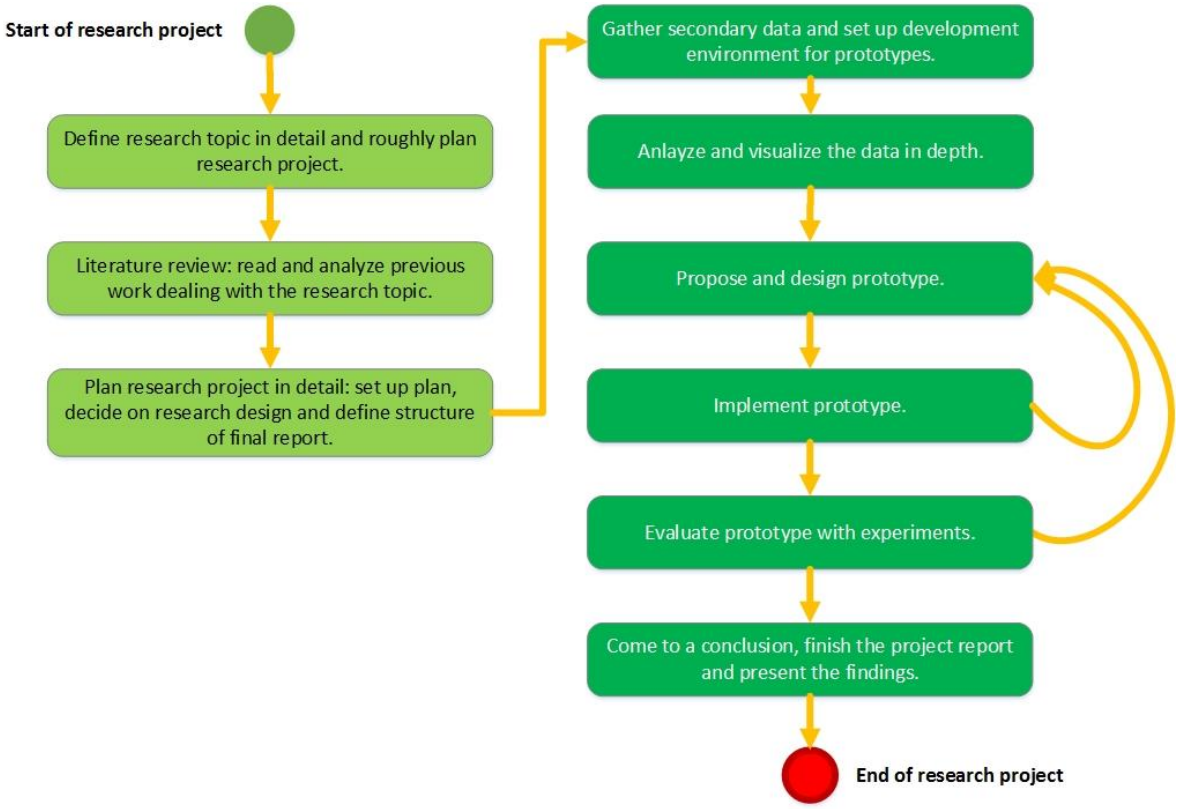


FIGURE 17: THE RESEARCH PLAN OF THIS RESEARCH PROJECT

The plan for this research project depicted above is based on the research process as proposed by Saunders, et al. (2009, pp. 11) as well as the design research approach by Vaishnavi and Kuechler (2004):

- The first step is the rough planning of the whole research project and to define the research topic via the thesis statement, the scope and limitation, etc. Its goal is to narrow down the rough boundaries and goals of the research project.
- This is followed by a thorough literature review to analyze work that was done previously in the field of (sequential) pattern recognition in the smart home environment. The goal of this step is to get familiarized with the topic and its problems and to already gain some insights and ideas how to tackle the concrete problem at hand.
- The third step deals with the detailed planning, mainly the research design and the structure of the report but also with a more detailed definition of the research questions derived from the thesis statement. It shall ensure that the research project is well planned and not done in an entirely ad-hoc fashion. This is the third and last step of the preparations for this research project.
- After that the first part of the actual research is the gathering and preparing of the secondary data as well as the setting up of the development environment for the prototypes. The goal is to have a suitable and working environment for the remainder of the master thesis and to lay the groundwork for a good design, implementation and evaluation of the prototypes.
- Next the secondary data has to be analyzed in depth and visualized. That the researcher gets familiar with the data and its details is the main goal of this step and an additional benefit is the visualization of the data that helps presenting the data in the final report. Additionally, it is planned to test various theories formed independently of the data.

- After getting familiar with the data, the sixth step is to propose and design a prototype. The insights gained from the previous parts of the research is crucial here and algorithms best fit to the specific data available for this project will be proposed. This shall ensure that not just a randomly chosen algorithm, possibly one that does not fit the data, is used in this research project.
- The seventh step called "implement prototype" is concerned with the actual realization of the prototype according to the design done previously. The goal of this is self-explanatory: build a running prototype.
- After building the prototype it is evaluated in the eighth step. For that, experiments with different parameter settings will be used. Doing experiments is an essential contribution towards the overall goal of finding the best algorithm for this research project, meaning the algorithm that performance the best with the specific data available to this project.
  Steps six, seven and eight can be repeated multiple times for the different algorithms which are compared regarding their performance as part of this research project.
- The final part of the research project is dealing with coming to the conclusion of the master thesis and finishing up the project report before it can be handed in to the supervisors as well as presented and defended at the master thesis defense. The goal of this step is to ensure that there is still time left after the implementation and evaluation to finish the research project in a proper way and write a high quality paper / final report.

3.3 CONCLUSION OF CHAPTER 3

In this chapter the research methodology chosen for this project as well as the theoretical background for the choice is provided. The goal of this chapter is to elaborate what is done when and to provide a rational for the proposed approach. This goal is achieved via the research plan and its description in the previous subchapter.

4 ANALYSIS OF THE DATA

4.1 INTRODUCTION TO CHAPTER 4

The following fourth chapter is the first one dealing with the actual research and addresses the need for an in-depth data analysis. The goal of this analysis is for the researcher to get to know the available secondary data for this research project and to answer the first two research questions:

1. What exactly is a pattern and what kind of patterns exist inside the data?
2. What regularities do the patterns have?

This helps to prove (or disprove) the thesis statement. The following sub-chapters are presenting the results of the data analysis.

4.2 THE ERD OF THE POWER CONSUMPTION & EVENTS DATABASE

The ERD of the database storing the power measurements and events looks as follows:

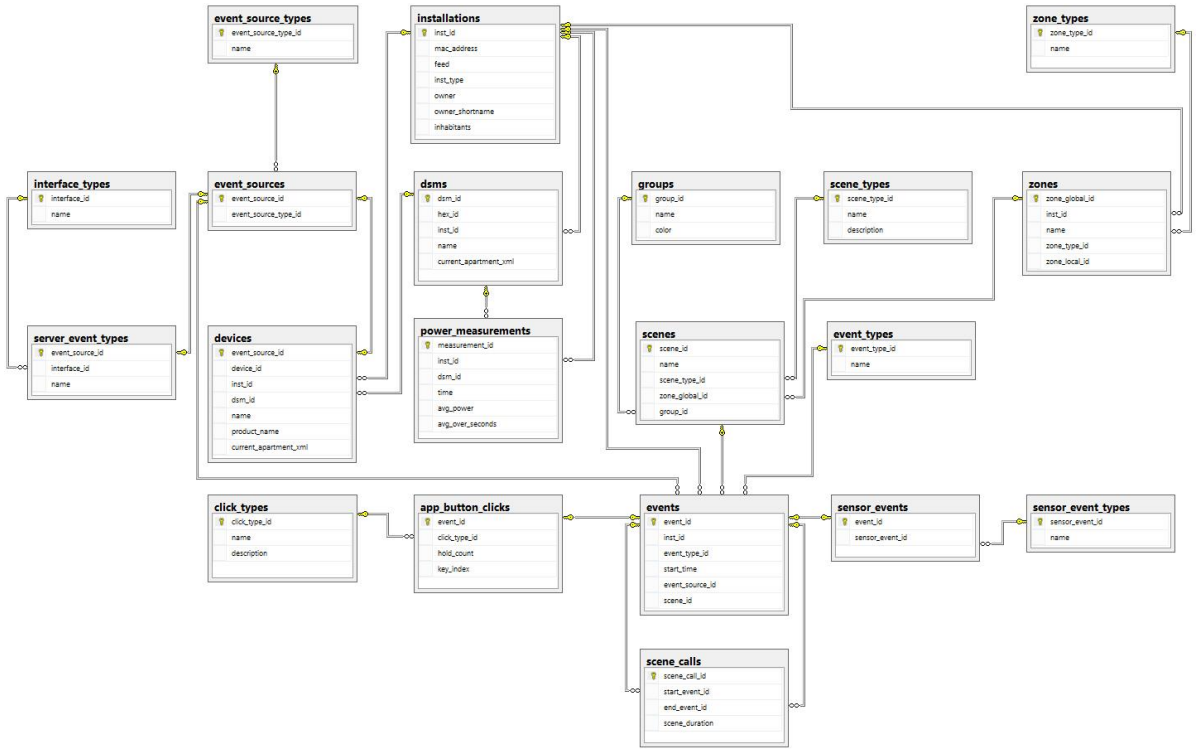


FIGURE 18: ERD OF THE AVAILABLE POWER CONSUMPTION AND EVENTS DATABASE

For this research project, the tables power_measurements and events are the most important ones:

- To find power consumption patterns, the data inside the power_measurements table have to be analyzed with the following SQL query, which aggregates the attribute avg_power:

```
SELECT pm.time, SUM(pm.avg_power)
FROM power_measurements AS pm
WHERE inst_id = [smart_home_id]
AND time BETWEEN [start] AND [end]
GROUP BY time
ORDER BY time
```

The results of such an analysis for one week and one smart home looks something like this:

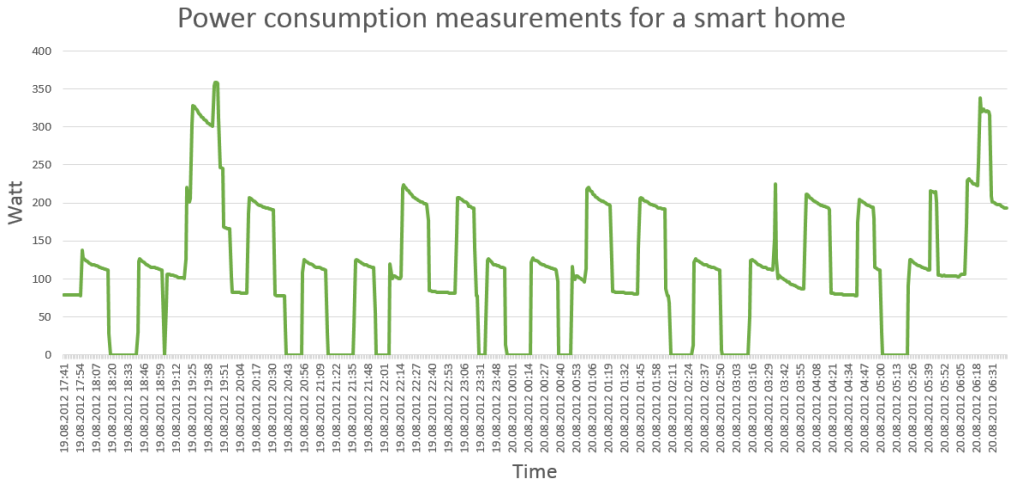


FIGURE 19: EXAMPLE OF A PATTERN INSIDE THE POWER_MEASUREMENTS TABLE

- The event patterns can be found inside the events table by analyzing the attributes event_source_id (= where the activity was triggered, e.g. the light switch in the bed room) and scene_id (what was triggered, e.g. on or off) ordered chronologically and by event_id. In addition to the table events, which holds the vast majority of all event information, there are special cases where some event information is recorded in the tables app_botton_clicks and sensor_events. To be able to analyze these events as well, a UNION is used. And finally it is necessary to excluded non-user-events, which are necessary for the Aizo digitalSTROM components to work correctly, but are not interesting for this research project. To analyze such patterns the following SQL query can be used:

```
SELECT e.start_time, e.event_source_id, e.scene_id AS 'scene-/click-/sensor-id',
e.event_id
FROM events AS e, event_sources AS es, event_source_types AS est
WHERE inst_id = [smart_home_id]
AND e.event_source_id = es.event_source_id
AND es.event_source_type_id = est.event_source_type_id
AND est.name != 'SERVER_EVENT' AND e.scene_id IS NOT NULL

UNION

SELECT e.start_time, e.event_source_id, e.scene_id AS 'scene-/click-/sensor-id',
e.event_id
FROM events AS e, event_sources AS es, server_event_types AS [set],
interface_types AS it
WHERE inst_id = [smart_home_id]
AND e.event_source_id = es.event_source_id
AND es.event_source_id = [set].event_source_id
AND [set].interface_id = it.interface_id
AND it.name = 'JSON'

UNION

SELECT e.start_time, e.event_source_id, abc.click_type_id AS 'scene-/click-
/sensor-id', e.event_id
FROM events AS e INNER JOIN app_button_clicks AS abc ON e.event_id =
abc.event_id
WHERE abc.click_type_id != '0' --exclusion of strange outliers
```

```
AND inst_id = [smart_home_id]

UNION

SELECT e.start_time, e.event_source_id, se.sensor_event_id AS 'scene-/click-
/sensor-id', e.event_id
FROM events AS e INNER JOIN sensor_events AS se ON e.event_id = se.event_id
WHERE inst_id = [smart_home_id]

ORDER BY e.start_time, event_id  --event_id needed for events at the same sec
```

One example for a six-events pattern in smart home no. 6, which is found by executing the above query, looks as follows:

start_time	event_source_id	scene_id
28.04.2012 13:26:38	377	434
28.04.2012 13:30:39	381	424
28.04.2012 13:30:49	381	422
28.04.2012 13:41:50	377	436
28.04.2012 13:42:10	381	424
28.04.2012 13:42:10	381	422

TABLE 4: EXAMPLE OF A 6-EVENTS PATTERN FOUND IN THE EVENTS TABLE

4.3 THE DEFINITION OF A PATTERN FOR THIS RESEARCH PROJECT

Based on the above analysis of the ERD, a pattern in the context of this research project is:

A combination of the same single events in the same order.

For example, the following six events make up one pattern: Switch 377: Mood1 (scene_id 434) -> Switch 381: Mood2 (scene_id 424) -> Switch 381: Off (scene_id 422) -> Switch 377:Off (scene_id 436) -> Switch 381: Mood2 (scene_id 424) -> 381:Off (scene_id 422)

As mentioned in the chapter 4.2 the information about this kind of pattern can be found inside the events, app_botton_clicks and sensor_events tables.

Also as mentioned above, the other kind of pattern is found in the power_measurements table. These patterns are the ones of the power consumption per smart meter in Watt. What is considered a pattern in this context is highlighted in red below. The particular pattern in the example is repeated two times:

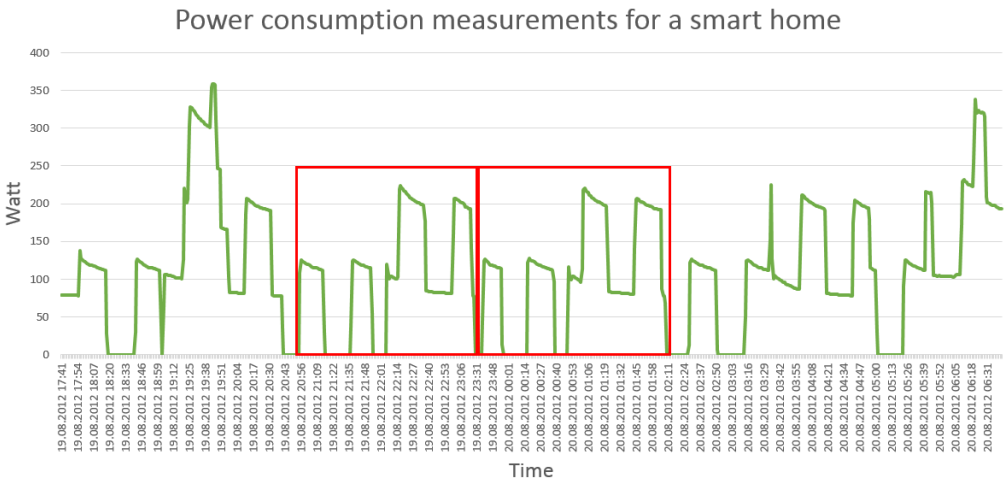


FIGURE 20: EXAMPLE OF A PATTERN INSIDE THE POWER_MEASUREMENTS TABLE (HIGHLIGHTED)

For both, event as well as power measurement patterns, two different regularities could be found in the data from Aizo and can be distinguished for the remainder of this research project:

- Frequent patterns, i.e. patterns that occur significantly more often than other patterns or single events.
- Periodic patterns, i.e. patterns that are not frequent but that reoccur at always the same intervals.

This finding regarding regularities matches with what Rashidi and Cook (2009) found in their smart home data.

4.4 INITIAL ANALYSIS AND VISUALIZATION OF THE DATA

To get an impression of relationships and connections in the available data, an initial analysis of them was needed. This initial analysis wanted to answer the following questions, which are thought interesting by the researcher:

- What is the total number of power measurements for each smart home?
- What is the total number of events for each smart home?
- How many events are there per event type (this is a specialty of the Aizo components, which distinguish so-called event types. Different event types can e.g. be lighting, shading or climate)?
- Which is the most promising smart home regarding patterns?

4.4.1 NUMBER OF EVENTS PER SMART HOME

To analyze the available events, the following SQL query can be used:

```
SELECT e.inst_id AS "Smart home no.", COUNT(event_id) AS "Event count"
FROM events AS e
GROUP BY e.inst_id,
ORDER BY e.inst_id
```

The results of this query look as follows:

Smart home	Event count	Smart home	Event count	Smart home	Event count
2	48'097	17	36'686	34	1'268
3	8'685	18	72'585	39	28'649
5	126'424	20	36'991	40	26'206
6	13'883	23	126'713	50	25'408
7	1'173	24	21'084	54	72'304
9	15'289	26	13'821	55	43'835
10	81'731	28	70'873	56	14'241
11	399'340	30	43'631	94	5'000

TABLE 5: NO. OF EVENTS PER SMART HOME

The above results presented as a column chart look as follows:

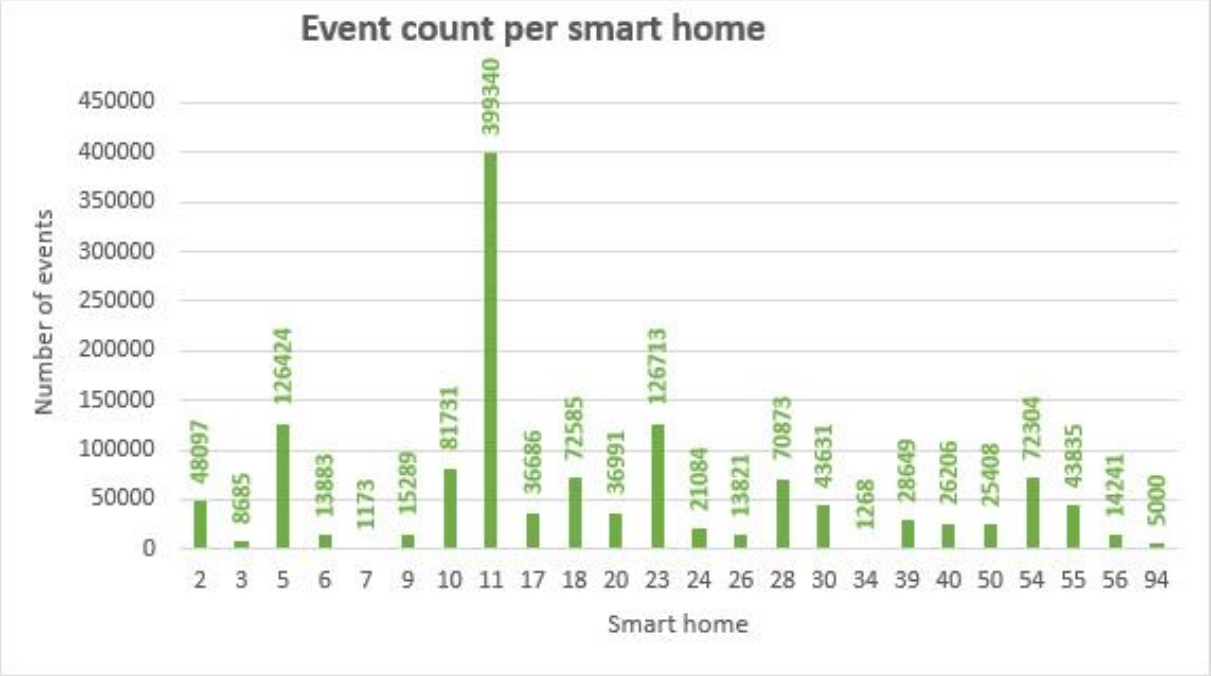


FIGURE 21: NO. OF EVENTS PER SMART HOME

The number of recorded events in the database varies greatly. The implication here is that the time for mining might vary greatly depending on which smart home is mined since the run time is expected to depend on the number of events that have to be processed.

4.4.2 NUMBER OF EVENTS PER EVENT TYPE

Since the event data inside the database is categorized according to different event types, it can be analyzed per type. For this purpose, the following SQL query was used:

```
SELECT g.name AS "Event type", COUNT(event_id) AS "Event count"
FROM events AS e
INNER JOIN scenes AS s ON e.scene_id = s.scene_id
INNER JOIN groups AS g ON s.group_id = g.group_id
GROUP BY g.name, g.color
```

If above query is executed, the results look as follows:

Event type	Event count	Event type	Event count	Event type	Event count
Audio	5'369	Climate	5'428	Security	81
Broadcast	167'253	Lighting	625'366	Video	5'524
Joker	12	Shading	59'242	Access	21

TABLE 6: NO. OF EVENTS PER EVENT TYPE

The resulting column chart looks as follows:

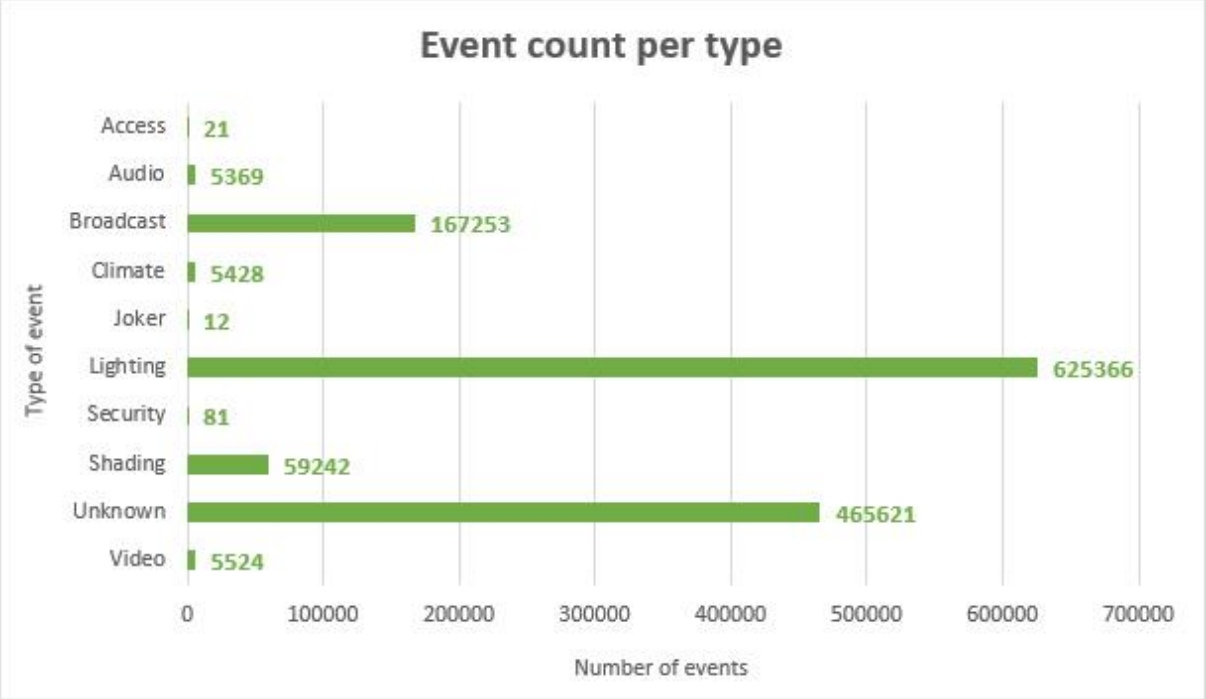


FIGURE 22: NO. OF EVENTS PER EVENT TYPE

As can be seen, lighting events are by far the ones that happen most often followed by events for which the type is unknown. For this research project this is of little consequence, however, since the goal is to find patterns and such patterns exist also inside the big amount of lighting events and the events for which the type is unknown.

Additionally the distribution of the different event types at the different smart homes was analyzed. For that, the events were grouped by type and smart home with the following SQL query, which only includes manual events:

```
DECLARE @group_id int
DECLARE smart_homes_cursor CURSOR FOR
       SELECT DISTINCT group_id
       FROM groups
       ORDER BY group_id
OPEN smart_homes_cursor
FETCH NEXT FROM smart_homes_cursor
INTO @group_id
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
	SELECT e.inst_id AS "Smart home", COUNT(event_id) AS "Event count"
	FROM groups AS g
	FULL OUTER JOIN scenes AS s ON s.group_id= g.group_id
	FULL OUTER JOIN events AS e ON e.scene_id= s.scene_id
	FULL OUTER JOIN event_sources AS es ON e.event_source_id = es.event_source_id

	WHERE g.group_id = @group_id
	AND es.event_source_type_id != '0' -- EXCLUSION OF SERVER_EVENT
	GROUP BY e.inst_id
	UNION
	SELECT e.inst_id, 0
	FROM events e
	WHERE e.inst_id NOT IN
	(
		SELECT DISTINCT e.inst_id
		FROM groups AS g
		FULL OUTER JOIN scenes AS s ON s.group_id= g.group_id
		FULL OUTER JOIN events AS e ON e.scene_id= s.scene_id
		FULL OUTER JOIN event_sources AS es ON e.event_source_id =
es.event_source_id
		WHERE g.group_id = @group_id
		AND es.event_source_type_id != '0' -- EXCLUSION OF SERVER_EVENT
		GROUP BY e.inst_id
	)
	ORDER BY "Smart home"
	FETCH NEXT FROM smart_homes_cursor
	INTO @group_id
END
CLOSE smart_homes_cursor
DEALLOCATE smart_homes_cursor

--- AND ---

SELECT inst_id AS "Smart home", COUNT(event_id) AS "Event count"
FROM events AS e
INNER JOIN event_sources AS es ON e.event_source_id = es.event_source_id
WHERE scene_id IS NULL
AND es.event_source_type_id != '0' -- EXCLUSION OF SERVER_EVENT
GROUP BY e.inst_id
UNION
SELECT DISTINCT e.inst_id, 0
FROM events AS e
WHERE e.inst_id NOT IN
(
		SELECT e.inst_id
		FROM events AS e
		INNER JOIN event_sources AS es ON e.event_source_id = es.event_source_id
		WHERE e.scene_id IS NULL
		AND e.scene_id IS NULL
		AND es.event_source_type_id != '0' -- EXCLUSION OF SERVER_EVENT
		GROUP BY inst_id
)
ORDER BY "Smart home"
```

The following table represents the consolidation of the different result sets:

Smart Home	Broadcast	Lighting	Shading	Climate	Audio	Video	Security	Access	Joker	Unknown	TOTAL
2	6'574	30'501	4'523	0	38	0	2	0	0	115	41'753
3	882	4'840	87	0	0	0	0	0	0	962	6'771
5	33'807	11'497	2'639	12	985	290	0	0	0	19'061	68'291
6	494	8'856	202	126	0	0	0	0	0	6	9'684
7	273	793	0	0	0	56	0	0	0	0	1'122
9	2'510	4'394	0	318	10	0	3	0	0	3'905	11'140
10	9'274	70'720	0	0	0	0	0	0	0	0	79'994
11	3'578	50'127	9'523	51	0	10	0	0	0	318'563	381'852
17	12'992	13'658	7'296	0	286	0	0	0	0	116	34'348
18	3'219	31'622	0	0	0	15	0	0	0	29'066	63'922
20	495	13'113	765	221	0	0	0	0	0	19'314	33'908
23	11'460	40'016	831	6	1'304	696	0	0	0	4'929	59'242
24	4'735	15'483	0	0	0	0	0	0	0	0	20'218
26	2'627	7'234	0	22	31	0	0	0	0	0	9'914
28	3'986	20'663	0	0	802	2'172	0	11	0	38'039	65'673
30	6'885	25'655	3'763	2	0	0	0	0	0	2'044	38'349
34	67	745	0	0	0	0	0	0	0	313	1'125
39	2'654	21'606	0	0	0	0	0	0	0	71	24'331
40	8'887	9'409	0	0	410	0	0	0	0	4'856	23'562
50	2'538	17'751	392	32	0	0	0	0	0	7	20'720
54	9'829	36'266	2'332	2	0	247	0	0	0	13'176	61'852
55	8'155	19'193	1'921	0	0	0	0	0	0	10'869	40'138
56	1'254	12'300	0	0	0	0	0	0	0	0	13'554
94	242	3'919	27	1	0	0	0	0	0	209	4'398
ALL	137'417	470'361	34'301	793	3'866	3'486	5	11	0	465'621	**1'115'861**

TABLE 7: NO. OF MANUAL EVENTS PER EVENT TYPE AND SMART HOME

This table can be normalized, and a stacked column chart of these normalized values looks as follows:

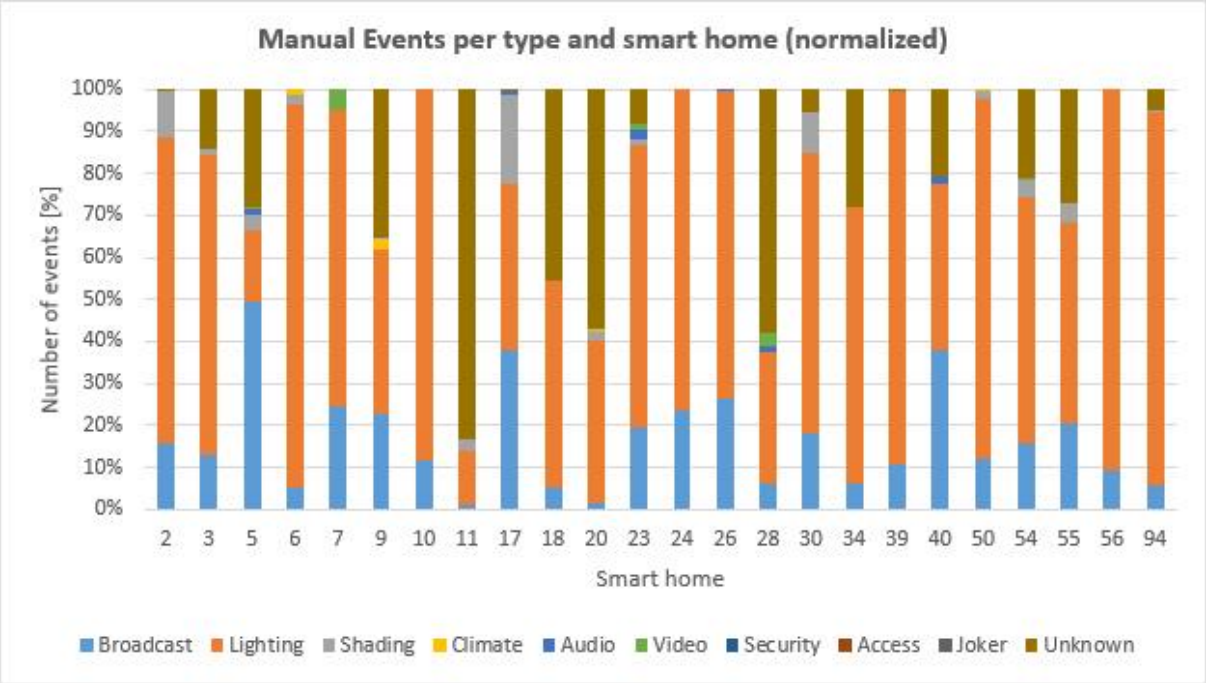


FIGURE 23: MANUAL EVENTS PER TYPE AND SMART HOME (NORMALIZED)

As can be seen, the two major event types, lighting and unknown, make up the main part of events in every smart home except for no. 5 and 17. These two exceptions have to be regarded as coincidental if one keeps the overall result from all smart homes in mind.

The previous statement therefore holds still true: the patterns have to be found mainly in the lighting event group or in the group of unknown events for this research project.

4.4.3 THE MOST PROMISING SMART HOME REGARDING PATTERNS

Based on above analysis and additional criteria, the most promising smart home for initial pattern detection was looked for. The criteria for the choice of such a model smart home included:

- A big range of power measurements since this would indicate a high variance of use of appliances.
- An above-average value for the average power consumption to be sure to exclude smart homes with only a few appliances connected to the installed smart meters.

Additional criteria for finding a short range of time, e.g. one day, which could be analyzed in a first manual data mining session, had to be specified to cope with the high amount of available data per smart home since it is not feasible to analyze thousands of events or power measurements by hand. These criteria included:

- The chosen day should not fall into the first few days of data collection since it is conceivable that the smart home inhabitants would be trying out special things during the first few days / weeks after the data collection is activated resulting in outliers and irregularities in the data.
- The chosen day should be a week day since on weekends inhabitants sometimes are away from home which results in less patterns in the data.

- The chosen day should have more than a hand full of events to be sure that there was enough user activity that day. Additionally, this criterion excludes days with some data collection problems e.g. with the components, the network or the database, which would result in a non-average number of measurements, which in turn would make the analysis worthless.

A lot of searching and analyzing (including dealing with outliers and irregularity in the data as well as rejecting smart homes and dates that seemed fitting only at a first glance) provided smart home no. 54 on 3 September 2012 as a suitable model smart home and day.

The data for the first manual analysis, which was the power consumption per hour, was selected with the following query:

```
SELECT LEFT(time,13)+' : xx' AS time, SUM(avg_power)/60 AS "average combined power
consumption per hour of all dsm", COUNT(dsm_id)/60 AS "number of dsms reporting
consumption"
FROM power_measurements
WHERE inst_id = 54
AND time BETWEEN '2012-09-03' AND '2012-09-04'
GROUP BY LEFT(time,13)
ORDER BY LEFT(time,13) ASC
```

The result of this query looks as follows:

Time	Average combined power consumption per hour of all DSMs	Number of DSMs reporting consumption
2012-09-03 00 : xx	343.2066778	9
2012-09-03 01 : xx	292.4585687	9
2012-09-03 02 : xx	313.6624265	9
2012-09-03 03 : xx	290.8616391	9
2012-09-03 04 : xx	288.4580851	9
2012-09-03 05 : xx	329.6928258	9
2012-09-03 06 : xx	318.5355649	9
2012-09-03 07 : xx	806.6884355	9
2012-09-03 08 : xx	280.5363538	7
2012-09-03 09 : xx	538.4421606	4
2012-09-03 10 : xx	595.7873942	4
2012-09-03 11 : xx	127.8930234	4
2012-09-03 12 : xx	102.1212632	4
2012-09-03 13 : xx	204.5029643	4
2012-09-03 14 : xx	200.7446348	4
2012-09-03 15 : xx	179.3874242	4
2012-09-03 16 : xx	114.4605915	4
2012-09-03 17 : xx	360.8442751	4
2012-09-03 18 : xx	803.1161284	4
2012-09-03 19 : xx	199.9710942	4
2012-09-03 20 : xx	557.1621603	6
2012-09-03 21 : xx	1'095.2617050	10

2012-09-03 22 : xx	786.7482717	10
2012-09-03 23 : xx	604.4874980	10

TABLE 8: POWER CONSUMPTION IN THE MOST PROMISING SMART HOME 54 ON 3 SEPT. 2012

As can be seen from the significantly higher power usage, the smart home 54 inhabitant(s) was/were awake after 07:00. The lower consumption between 08:00 and 17:59 indicates that he/she/they were away from home during this time. Therefore two interesting time periods to look for patterns are between 07:00 and 08:00 (getting up) and between 18:00 and 19:00 (coming home).

For the purpose of analyzing the events happening in the same smart home at the same day, the following SQL query was used:

```
SELECT e.*, et.name, s.name
FROM events AS e, event_types AS et, scenes AS s
WHERE inst_id = 54
AND start_time BETWEEN '2012-09-03 00:00' AND '2012-09-03 23:59:59'
AND e.event_type_id = et.event_type_id
AND e.scene_id = s.scene_id
ORDER BY start_time
```

The result of that query looks as follows:

event_id	inst_id	event_type_id	start_time	event_source_id	scene_id	name	name
619664	54	1	03.09.2012 06:42:37	3455	3118	SCENE_CALL	Stimmung2
619665	54	1	03.09.2012 06:57:56	3455	3119	SCENE_CALL	Off
619666	54	1	03.09.2012 07:00:06	3506	3139	SCENE_CALL	Stimmung1
619667	54	1	03.09.2012 07:00:22	3488	3155	SCENE_CALL	Off
619668	54	1	03.09.2012 07:02:25	3495	3114	SCENE_CALL	Off
619669	54	1	03.09.2012 07:02:29	3495	3112	SCENE_CALL	Stimmung1
619670	54	1	03.09.2012 07:02:51	3495	3114	SCENE_CALL	Off
619671	54	1	03.09.2012 07:02:57	3495	3112	SCENE_CALL	Stimmung1
619672	54	1	03.09.2012 07:03:39	3495	3114	SCENE_CALL	Off
619673	54	1	03.09.2012 07:03:52	3455	3141	SCENE_CALL	Stimmung1
619674	54	1	03.09.2012 07:08:03	3453	3124	SCENE_CALL	Off
619675	54	1	03.09.2012 07:08:05	3453	3123	SCENE_CALL	Stimmung1
619676	54	1	03.09.2012 07:10:00	3727	3156	SCENE_CALL	Stimmung1
619677	54	1	03.09.2012 07:11:41	3453	3124	SCENE_CALL	Off
619678	54	1	03.09.2012 07:12:09	3453	3123	SCENE_CALL	Stimmung1
619679	54	1	03.09.2012 07:12:28	3496	3113	SCENE_CALL	Stimmung1
619680	54	1	03.09.2012 07:16:10	3479	3151	SCENE_CALL	Off
619681	54	1	03.09.2012 07:16:11	3479	3152	SCENE_CALL	Stop
619682	54	1	03.09.2012 07:16:13	3479	3157	SCENE_CALL	Stimmung1
619683	54	1	03.09.2012 07:22:23	3453	3124	SCENE_CALL	Off
619684	54	1	03.09.2012 07:24:53	3455	3119	SCENE_CALL	Off
619685	54	1	03.09.2012 07:25:00	3455	3118	SCENE_CALL	Stimmung2
619686	54	1	03.09.2012 07:26:30	3469	3121	SCENE_CALL	Stimmung1
619687	54	1	03.09.2012 07:29:55	3469	3122	SCENE_CALL	Off
619688	54	1	03.09.2012 07:40:13	3455	3119	SCENE_CALL	Off
619689	54	2	03.09.2012 07:45:52	3497	3129	LOCAL_DEVICE_SCENE	On-Device
619690	54	2	03.09.2012 07:50:25	3475	3129	LOCAL_DEVICE_SCENE	On-Device
619691	54	2	03.09.2012 07:51:28	3483	3129	LOCAL_DEVICE_SCENE	On-Device
619692	54	2	03.09.2012 07:54:44	3483	3132	LOCAL_DEVICE_SCENE	Off-Device
619693	54	1	03.09.2012 08:08:26	3496	3115	SCENE_CALL	Off

619694	54	1	03.09.2012 08:23:00	3727	3162	SCENE_CALL	Off
619695	54	1	03.09.2012 09:30:00	3727	3139	SCENE_CALL	Stimmung1
619696	54	1	03.09.2012 09:30:01	3727	3157	SCENE_CALL	Stimmung1
619697	54	1	03.09.2012 09:30:02	3727	3140	SCENE_CALL	Stimmung1
619698	54	1	03.09.2012 09:40:47	3726	3116	SCENE_CALL	Bell
619699	54	1	03.09.2012 15:47:44	3488	3264	SCENE_CALL	Stimmung2
619700	54	1	03.09.2012 15:47:45	3488	3171	SCENE_CALL	Stop
619701	54	1	03.09.2012 15:47:49	3488	3155	SCENE_CALL	Off
619702	54	1	03.09.2012 15:49:46	3455	3118	SCENE_CALL	Stimmung2
619703	54	1	03.09.2012 15:49:59	3455	3119	SCENE_CALL	Off
619704	54	1	03.09.2012 16:01:04	3726	3116	SCENE_CALL	Bell
619705	54	1	03.09.2012 17:44:04	3496	3160	SCENE_CALL	Stimmung2
619706	54	2	03.09.2012 17:47:12	3475	3132	LOCAL_DEVICE_SCENE	Off-Device
619707	54	1	03.09.2012 18:20:44	3496	3115	SCENE_CALL	Off
619709	54	1	03.09.2012 19:29:51	3496	3113	SCENE_CALL	Stimmung1
619710	54	1	03.09.2012 19:33:31	3484	3117	SCENE_CALL	Stimmung1
619711	54	1	03.09.2012 20:00:00	3727	3138	SCENE_CALL	Stimmung4
619712	54	1	03.09.2012 20:00:01	3727	3155	SCENE_CALL	Off
619713	54	1	03.09.2012 20:00:02	3727	3159	SCENE_CALL	Off
619715	54	1	03.09.2012 20:14:26	3727	3128	SCENE_CALL	Off
619716	54	2	03.09.2012 20:15:11	3449	3129	LOCAL_DEVICE_SCENE	On-Device
619717	54	1	03.09.2012 20:24:06	3469	3121	SCENE_CALL	Stimmung1
619718	54	1	03.09.2012 20:25:17	3469	3122	SCENE_CALL	Off
619719	54	1	03.09.2012 20:25:52	3496	3115	SCENE_CALL	Off
619720	54	1	03.09.2012 20:34:26	3727	3135	SCENE_CALL	Stimmung1
619721	54	1	03.09.2012 20:36:02	3455	3141	SCENE_CALL	Stimmung1
619722	54	1	03.09.2012 20:38:58	3455	3119	SCENE_CALL	Off
619723	54	1	03.09.2012 20:40:32	3495	3226	SCENE_CALL	Stimmung2
619724	54	1	03.09.2012 20:40:33	3727	3127	SCENE_CALL	Stimmung3
619725	54	2	03.09.2012 20:49:55	3507	3132	LOCAL_DEVICE_SCENE	Off-Device
619726	54	1	03.09.2012 20:50:03	3455	3118	SCENE_CALL	Stimmung2
619727	54	2	03.09.2012 20:51:02	3507	3129	LOCAL_DEVICE_SCENE	On-Device
619728	54	2	03.09.2012 20:51:33	3507	3132	LOCAL_DEVICE_SCENE	Off-Device
619729	54	1	03.09.2012 20:52:18	3455	3119	SCENE_CALL	Off
619730	54	1	03.09.2012 20:52:23	3467	3130	SCENE_CALL	On-Area3
619731	54	1	03.09.2012 20:52:46	3488	3140	SCENE_CALL	Stimmung1
619732	54	1	03.09.2012 20:52:48	3453	3123	SCENE_CALL	Stimmung1
619733	54	1	03.09.2012 20:52:54	3453	3143	SCENE_CALL	Stimmung2
619734	54	2	03.09.2012 20:58:02	3448	3132	LOCAL_DEVICE_SCENE	Off-Device
619735	54	1	03.09.2012 21:14:12	3491	3136	SCENE_CALL	Stimmung1
619736	54	2	03.09.2012 21:14:15	3497	3132	LOCAL_DEVICE_SCENE	Off-Device
619737	54	1	03.09.2012 21:15:16	3496	3160	SCENE_CALL	Stimmung2
619738	54	1	03.09.2012 21:18:57	3455	3118	SCENE_CALL	Stimmung2
619739	54	1	03.09.2012 21:23:21	3496	3115	SCENE_CALL	Off
619740	54	1	03.09.2012 21:25:09	3491	3137	SCENE_CALL	Off
619741	54	1	03.09.2012 21:25:51	3455	3119	SCENE_CALL	Off
619742	54	1	03.09.2012 22:45:17	3727	3148	SCENE_CALL	Off
619743	54	1	03.09.2012 22:56:02	3473	3125	SCENE_CALL	On-Area1
619744	54	1	03.09.2012 22:56:26	3455	3141	SCENE_CALL	Stimmung1
619745	54	1	03.09.2012 22:56:51	3461	3142	SCENE_CALL	Off
619746	54	1	03.09.2012 22:56:52	3485	3163	SCENE_CALL	On-Area1
619747	54	1	03.09.2012 22:56:54	3485	3146	SCENE_CALL	Off-Area1
619748	54	1	03.09.2012 22:56:57	3473	3126	SCENE_CALL	Off-Area1
619749	54	1	03.09.2012 23:03:11	3455	3119	SCENE_CALL	Off
619750	54	1	03.09.2012 23:03:15	3501	3149	SCENE_CALL	Off-Area2
619751	54	1	03.09.2012 23:03:18	3501	3150	SCENE_CALL	On-Area2
619753	54	2	03.09.2012 23:04:47	3537	3132	LOCAL_DEVICE_SCENE	Off-Device

619754	54	1	03.09.2012 23:19:09	3501	3149	SCENE_CALL	Off-Area2
619755	54	1	03.09.2012 23:20:07	3455	3141	SCENE_CALL	Stimmung1
619756	54	1	03.09.2012 23:23:49	3455	3119	SCENE_CALL	Off
619757	54	1	03.09.2012 23:23:52	3467	3131	SCENE_CALL	Off-Area3
619758	54	2	03.09.2012 23:25:44	3503	3129	LOCAL_DEVICE_SCENE	On-Device
619759	54	2	03.09.2012 23:34:05	3503	3132	LOCAL_DEVICE_SCENE	Off-Device

TABLE 9: EVENTS IN THE MOST PROMISING SMART HOME 54 ON 3 SEPT. 2012

As can be seen when looking at all the events of 3 September 2012 listed in the table above, they also suggest that the periods between 07:00 and 08:00 and between 18:00 and 19:00 could be interesting for pattern mining since the first event happened shortly before 7:00 (at 6:42) and the main body of events after 17:44 (which corresponds with the higher consumption starting after 7:00 and 18:00, respectively).

4.4.4 THE PROBLEM OF FUZZINESS OF THE POWER CONSUMPTION DATA

From the analysis of the power consumption data and the events of the model smart home no. 54 it became apparent that the aggregated power consumption data contains a certain fuzziness and is less sharp than events are. That problem can be illustrated well with the following figure:

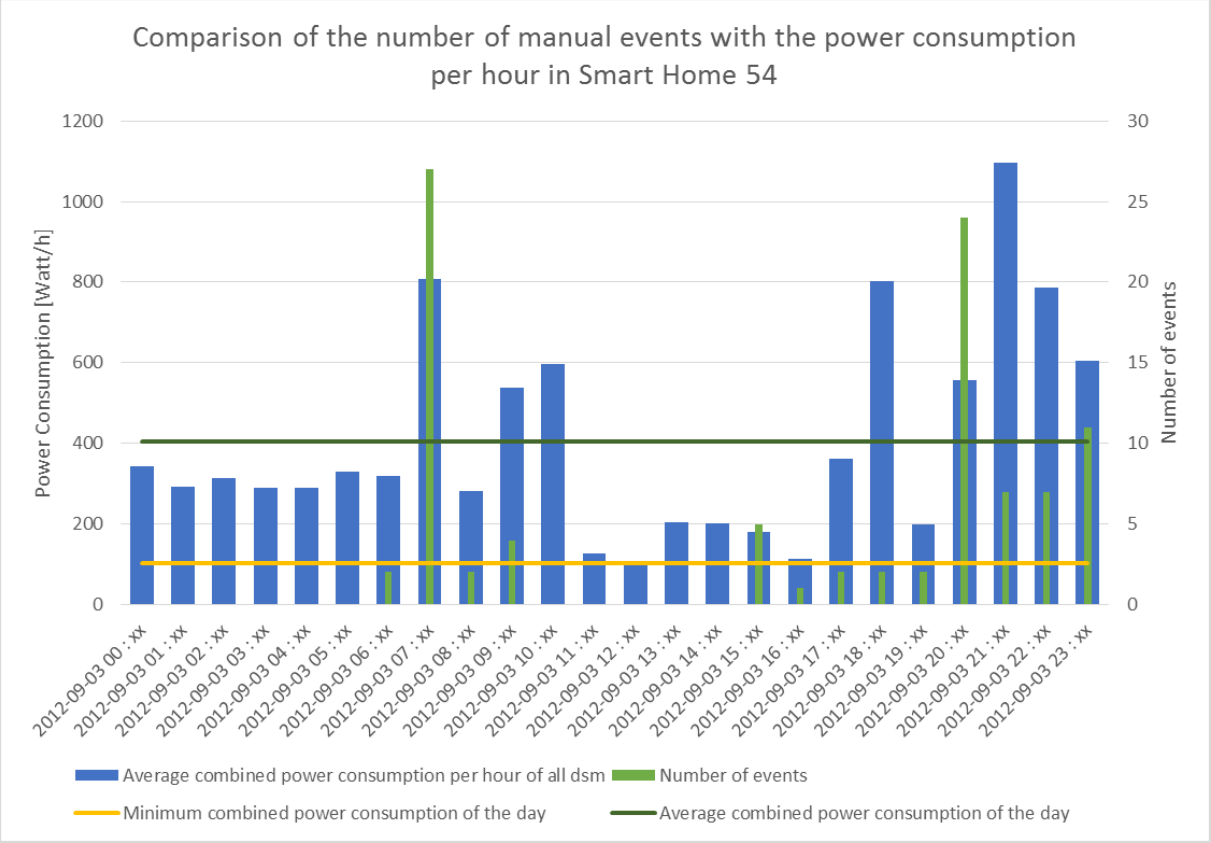


FIGURE 24: COMPARISON OF EVENTS AND POWER CONSUMPTION PER HOUR IN SMART HOME 54

When looking at Figure 24 above, it becomes clear that certain natural expectations are met, e.g. that during the night (00:xx – 05:xx) the power consumption stays more or less constant and no manual events occur because the inhabitants are most probably asleep. Additionally, when getting up at around 7:00 the events as well as the power consumption skyrocket, which is also what can be expected.

However, what is unexpected is that the decrease of power usage between 8:00 and 8:59 does not signify that the inhabitants left the house as can be seen from the manual events that still occur between 8:00 and 9:59. In addition, the time period between 15:00 and 16:59 with manual events but exceptionally low power consumption and the time period between 10:00 and 10:59 without manual events but exceptionally high power consumption were also surprising.

Because this fuzziness of the aggregated power consumption data was detected and since the mining of the events seemed better suited for pattern detection that can be used for automation, it was decided to set the focus solely on events and disregard the power consumption for this research project.

4.5 IN-DEPTH ANALYSIS OF THE EVENT DATA

Although the initial analysis and familiarization with the ERD took a lot of time and yielded little presentable results, it was non-the-less invaluable for the progress of this research project and to get to know the secondary data and its specialties. An additional important insight was that the focus should be laid on the event and not the power consumption data.

The evaluation of a suitable model smart home, the decision to only analyze events, and the finding of two interesting periods of time (7:00 – 8:00 and 18:00 – 19:00) were a prerequisite for the in-depth analysis regarding pattern detection.

The first in-depth analysis looked only at the events in smart home 54 over several days. A first hypothesis was that the inhabitants of smart home 54 normally get up on a work day at around 7 o'clock and that's why the timeframe from 00:00 till 09:00 was examined closer for a whole week:

```
SELECT e.*, et.name AS "Name of event type", s.name AS "Name of scene", est.name AS
"Name of event source type"
FROM events AS e, event_types AS et, scenes AS s, event_sources AS es,
event_source_types AS est
WHERE inst_id = 54
AND
(
  start_time BETWEEN '2012-09-03 00:00' AND '2012-09-03 09:00' OR
  start_time BETWEEN '2012-09-04 00:00' AND '2012-09-04 09:00' OR
  start_time BETWEEN '2012-09-05 00:00' AND '2012-09-05 09:00' OR
  start_time BETWEEN '2012-09-06 00:00' AND '2012-09-06 09:00' OR
  start_time BETWEEN '2012-09-07 00:00' AND '2012-09-07 09:00'
)
AND e.event_type_id = et.event_type_id
AND e.scene_id = s.scene_id
AND e.event_source_id = es.event_source_id
AND es.event_source_type_id = est.event_source_type_id
ORDER BY start_time
```

The results were analyzed side-by-side and per day. While a very general and broad pattern could be discerned (at least one inhabitant was getting up at around 07:00) most of the events reported by smart home 54 were random and no useful pattern could be detected by hand.

Therefore still another smart home, no. 6, was looked at (with the SQL query provided above). The results were again analyzed side-by-side and per day. This time much more homogenous events were visible. The longest pattern, which repeated itself, consisted of 6 activities and looked as follows:

Event I	Event II	Event III	Event IV	Event V	Event VI
410: 438	410: 440	381: 423	381: 422	395: 427	388: 430

TABLE 10: MANUALLY FOUND 6-EVENTS PATTERN FOR SMART HOME NO. 6

The first ID in the pattern above is the device where the event was triggered while the second ID is the triggered state. The event ID can be translated to the following clear-text descriptions of the events:

Event I	Event II	Event III	Event IV	Event V	Event VI
410: Mood1	410: Off	381: Mood1	381:Off	395: Mood1	388:Off

TABLE 11: MANUALLY FOUND 6-EVENTS PATTERN FOR SMART HOME NO. 6 (DESCRIPTIVE NAMES)

To mine for the 6-events pattern mentioned above, all manual events (`est.name != 'SERVER_EVENT'`) from the database for smart home 6 over the whole available period of eight months can be selected with the following query:

```
SELECT e.*, et.name AS "Name of event type", s.name AS "Name of scene", est.name AS
"Name of event source type", g.name AS "Name of scene group"
FROM events AS e, event_types AS et, scenes AS s, event_sources AS es,
event_source_types AS est, groups AS g
WHERE inst_id = 6
AND e.event_type_id = et.event_type_id
AND e.scene_id = s.scene_id
AND e.event_source_id = es.event_source_id
AND es.event_source_type_id = est.event_source_type_id
AND s.group_id = g.group_id
AND est.name != 'SERVER_EVENT'
ORDER BY start_time, event_id  -- event_id is needed for events which happen at the
same second
```

In an Excel spreadsheet the result set was mined with a simple IF-formula and found only 2 times:

IF(AND(E9678=”388”;H9678=”430”;E9677=”395”;H9677=”427”;E9676=”381”;H9676=”422”;E9675=”381”;H9675=”423”;E9674=”410”;H9674=”440”;E9673=”410”;H9673=”438”);true;false)

Because there are a total of 9’678 manual events stored in the database for smart home 6, the support count of this pattern is 2 and the support equals $2 : 9678 = 2.066542674106220293449059723083 3e - 4$.

Additionally the following shorter part of the pattern, which occurs 77 times, was found:

Event I	Event II
410: 438	410: 440

TABLE 12: MANUALLY FOUND MORE FREQUENT 2-EVENTS PATTERN FOR SMART HOME NO. 6

This pattern has a support count of 77 or a support of 0.00796.

Originally it was anticipated that additional analysis is needed to be able to design and implement an algorithm. This analysis would have been very explorative and broad in nature and done with the statistical open-source software R which allows to do things like:

- Time series analysis
- Cross-correlation
- Auto-correlation
- Cluster analysis

However, because the manual mining process described above showed that there are patterns in the event data, these analyses were skipped.

4.6 CONCLUSION OF CHAPTER 4

In this chapter the available data for this research project was analyzed: both in a superficial way as well as in-depth. The required tables of the database and the query to select the needed data have been identified. As another important insight it became clear that the power consumption data is not (as) suitable (as is the event data) to detect patterns that can be used to establish the baseline on which automation can rely later on.

Additionally it can be seen from the previous elaborations in this fourth chapter that the smart home event data contains patterns. It therefore seemed appropriate to continue this project by designing prototypes capable of mining frequent sequential patterns in the event data.

5 ALGORITHMS FOR DATA MINING

5.1 INTRODUCTION TO CHAPTER 5

This fifth chapter of the master thesis is going to introduce the different algorithms used for mining sequential patterns in the events of the Aizo smart home data.

To compare and benchmark different algorithms regarding their performance when mining the available smart home event data, it was necessary to design or adapted them, to write interfaces to existing frameworks, and to pre-process the input data as well as post-process the results. All this is described in the following subchapters 5.2 to 5.5.

5.2 BRUTE-FORCE APPROACH FOR FREQUENT SEQUENTIAL PATTERN MINING

Since a manual analysis and mining process as described in chapter 4.5 is a lengthy and inefficient process and the goal of this research project is to present prototypes able to find frequent or periodic patterns in the Aizo data set, the steps described before (finding a possible frequent pattern by hand and letting Excel count the occurrences inside the data) have to be automated in such a way that an algorithm is able to mine the patterns automatically.

The algorithm needs to be able to mine the frequent sequential patterns for all smart homes in all the event data. A straight forward solution is an algorithm following a brute-force approach. The activity diagram of such an algorithm looks as follows:

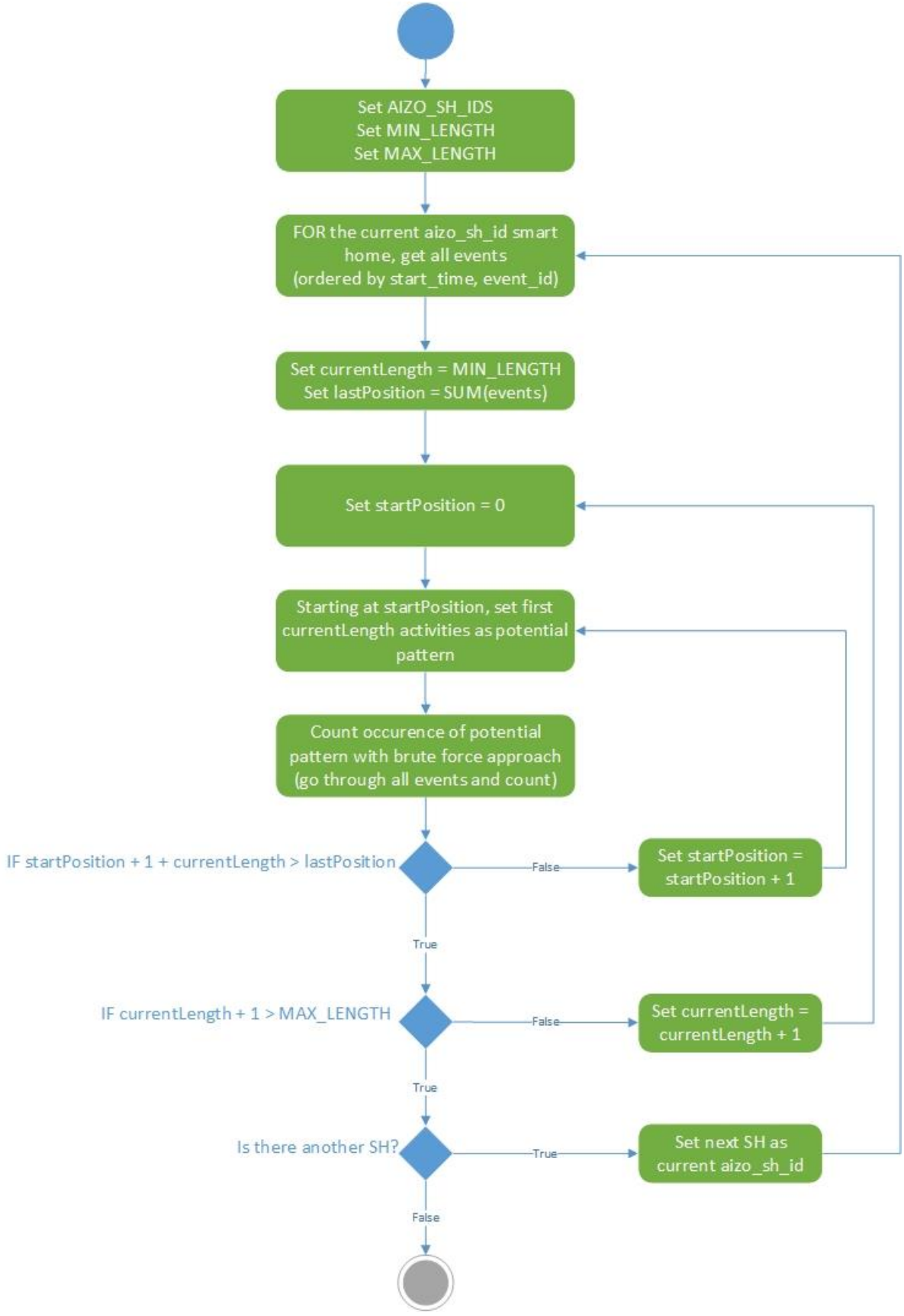


FIGURE 25: ACTIVITY DIAGRAM FOR BRUTE-FORCE SEQUENTIAL PATTERN MINER FOR THE AIZO DATA SET

The algorithm depicted above does the following:

1. All the IDs of the smart homes which shall be mined (at least one) as well as a minimum length and a maximum length for the patterns have to be set by the user.
2. The first of the specified IDs is taken (stored in the variable Aizo_sh_id) and all the events selected from the database for this smart home. These events are order first by their start time and then by their event_id. This second argument, event_id, in the ORDER BY clause ensures that the events are always ordered the same.
3. The parameter specified as the minimum length for patterns is then used as the current length, while the variable lastPosition is set to the number of events that were found in the database for the current smart home. lastPosition is needed later on to decide when the last event has been reached and the algorithm has to continue with the next pattern length.
4. The variable startPosition is initialized with 0. startPosition is needed in the next step to decide which position in the array containing the events shall be selected.
5. Starting with the event at startPosition (i.e. initial at 0, which is the first event), a pattern of length currentLength is selected as a potential frequent pattern. For example, if the startPosition = 0, currentLength = 2 and the first three activities in the events array are 100:200, 300:400 and 500:600, then the initial potential pattern is "100:200, 300:400".
6. The whole events array is scanned for this potential pattern and the count (which is the support count of this pattern) is remembered.
7. After the support count is determined, a gateway checks IF startPosition + 1 + currentLength is greater than lastPosition:
    a. if this is not the case, startPosition is increased by 1 and the algorithm continues with step 5 (in the example above, startPosition would now be 1 and step 5 would therefore set "300:400, 500:600" as the second potential pattern).
    b. if it is the case, startPosition is not changed and the algorithm continues with step 8.
8. A gateway checks IF currentLength + 1 is greater than MAX_LENGTH:
    a. if this is not the case, currentLength is increased by 1 and the algorithm continues with step 4. In the previous example this means that currentLenght = 3 and after resetting the startPosition to 0 in step 4, the potential pattern set by step 5 would now be "100:200, 300:400, 500:600".
    b. if it is not the case, currentLength is not changed and the algorithm continues with step 9.
9. The algorithm checks now if there was another smart home specified by the user:
    a. if this is the case, this next smart home is stored in the variable Aizo_sh_id and the algorithm repeats the whole procedure, beginning at step 2.
    b. if it is not the case, the algorithm is finished and all the support counts of all the potential frequent patterns of length MIN_LENGTH to MAX_LENGTH are known.

For this research project the above activity diagram was implemented in Java knowing that such a brute-force approach most probably would have problems regarding the run time when a lot of events have to be mined. However, the reason for the implementation of the brute-force approach was not to achieve good run times but to find out whether or not other patterns with a higher support count than the previously found 77 (see Table 12) or longer patterns with a comparable support exist in the available data for smart home no. 6.

The prototype of about 280 lines of Java code, which produces so-called overlapping patterns (Chikhaoui, et al., 2010), was able to confirm the support counts found by hand as described in chapter 4.5. In addition it found much more frequent patterns, e.g. the following 2-events pattern:

Event I	Event II
410:457	410:458

TABLE 13: THE MOST FREQUENT 2-EVENTS PATTERN FOR SMART HOME NO. 6

It was found 935 times inside the available data, which equals a support count of 935 and a support of 0.0966.

The longest six-events pattern, which consists of six times the same event, was found 82 times (support count = 82, support = 0.0085):

Event I	Event II	Event III	Event IV	Event V	Event VI
5542:3719	5542:3719	5542:3719	5542:3719	5542:3719	5542:3719

TABLE 14: THE MOST FREQUENT 6-EVENTS PATTERN FOR SMART HOME NO. 6

While these results were encouraging and supported the assumption of this research project that patterns are deducible in event data of smart homes, the run times were, as can be expected from a brute-force approach, quite disappointing. As can be seen in the following figure, the prototype is not very efficient and the measured run times were therefore not good. For example it took the algorithm 47'359 seconds (= 789 minutes = over 13 hours) to mine the 79'994 events (the tables app_botton_clicks and sensor_events were excluded) of smart home no. 10:

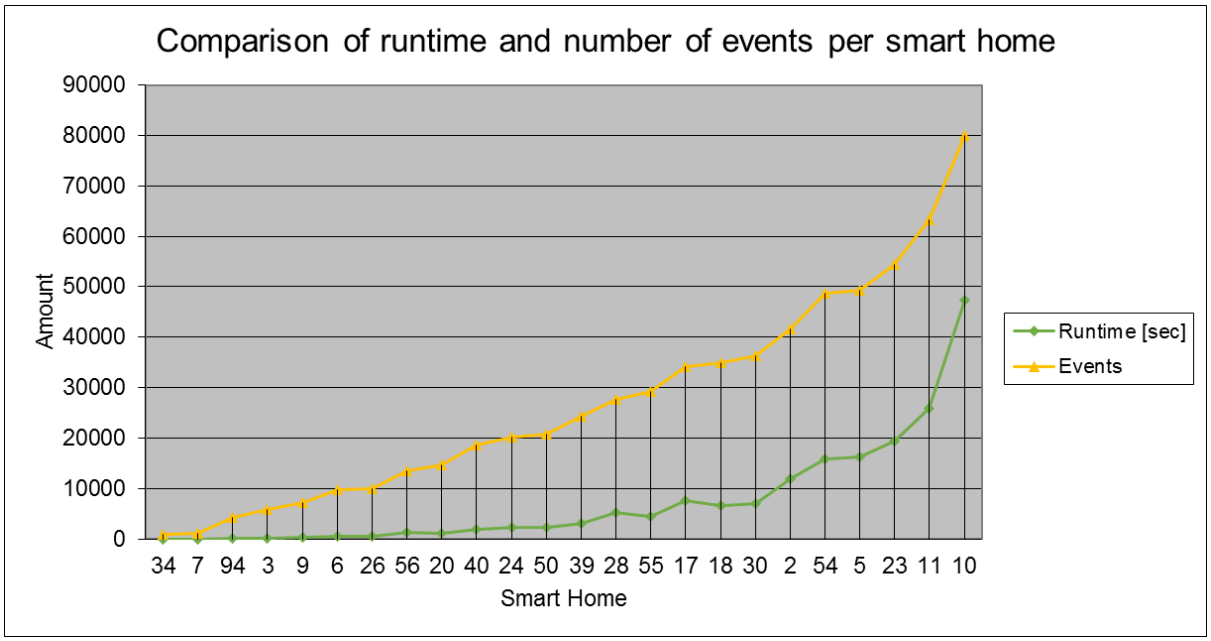


FIGURE 26: COMPARISON OF RUN TIME AND NUMBER OF EVENTS PER SMART HOME

While the algorithm develops more or less linear to the number of events, which is not be a bad thing as such, the throughput is much too low to be of any use in a real-world scenario. This is illustrated in Figure 27, where the number of events processed per second is compared to the overall number of events per smart home:

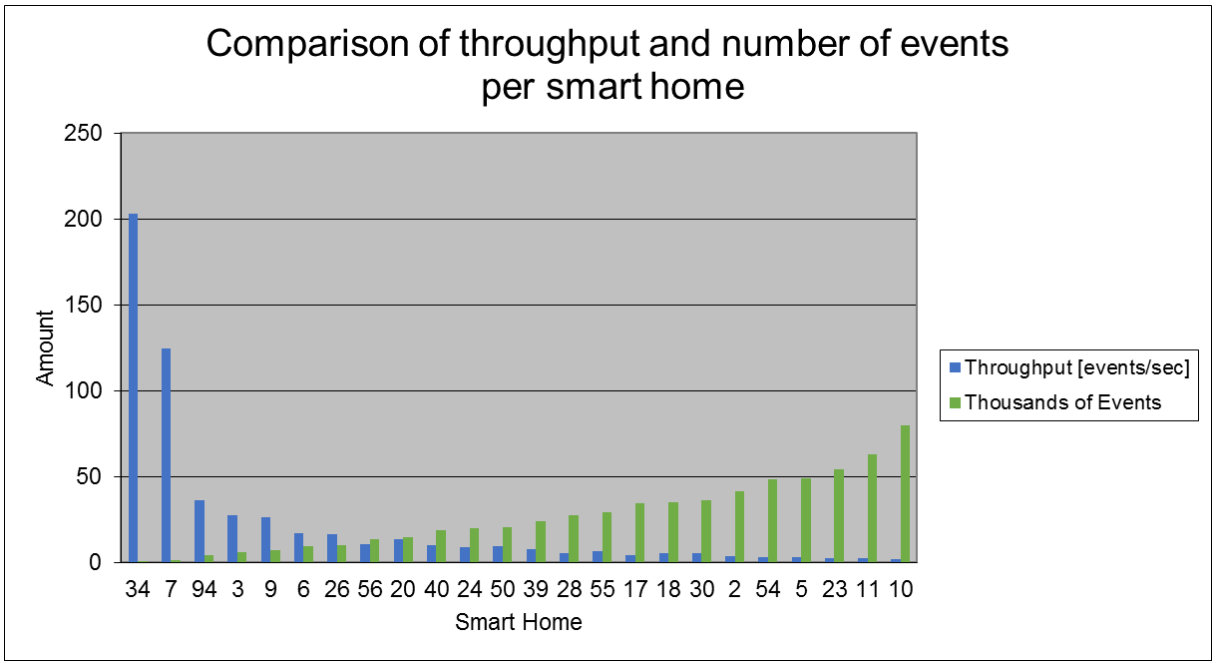


FIGURE 27: COMPARISON OF THROUGHPUT AND NUMBER OF EVENTS PER SMART HOME

A throughput of clearly less than 50 events per second (this is the case for all smart homes except no. 34 and no. 7, which have only very few events compared to the other smart homes) is not acceptable except for trial runs to gain a first insight into the available patterns. For example an average smart home like no. 55 has about 30‘000 events in the database available for this project (excluding the tables app_botton_clicks and sensor_events) and it took the algorithm over an hour to mine all the patterns with a length between two and six events. Since this is much too slow, different alternatives to speed up the algorithm were sought and evaluated.

The alternative that was finally chosen as the best solution includes two improvements:

1. It uses pattern de-duplication to minimize the patterns which are mined. Since the brute-force window sliding algorithm presented so far (called WSBF, Windows Sliding with Brute-Force, for the rest of the paper) does not do de-duplication, every pattern that occurs n times is mined n times, meaning that frequent patterns are mined over and over again, slowing down the whole process. The enhanced algorithm mines every possible pattern only once, which also gives the algorithm its name: WSDD, Window Sliding with De-Duplication.
2. Instead of building possible patterns in a first loop and then mine for these found patterns in a second loop, the counting is done in the same loop. This eliminates the need for looping over all the available data again which in turn speeds up the algorithm.

To achieve those two improvements a simple yet highly efficient data structure was used: the possible patterns were stored in a Java HashMap. A HashMap, sometimes also called associative array or just hash in other programming languages, is similar to an array but offers the ability to name the key individually while an array uses by definition unsigned integers as keys to address its content.

The HashMap used for the WSDD algorithm uses the key for storing the pattern itself and the support count of this pattern is stored as the value. This means that if a pattern is produced for the first time by sliding a window of a given size over the data, it is stored as a new key in the HashMap and the counter

(i.e. the key's value) is set to 1. If the algorithm is producing a pattern that already exists as a key in the HashMap, the key's value is incremented by one. This means that when WSDD has finished mining, the support of a certain pattern is stored as the value, while the pattern itself is contained as the key.

As an additional improvement, in analogy to the Apriori algorithm, a minimum support parameter was introduced. This was done because very infrequent patterns are not interesting, e.g. all the patterns which are executed only once can never be considered "normal behavior" of the smart home inhabitants. The minimum support parameter is therefore used to post-process the found patterns and WSDD is only returning patterns which make the cut, i.e. have a support equal to or higher than the specified minimum support.

When implemented in Java, it became clear that using associative memory in the form of HashMaps is an exceptionally fast way of generating possibly frequent sequential patterns and counting their support inside the data. When compared to the WSBF algorithm presented thus far, the improved WSDD algorithm, which does de-duplication, showed run times which are 80 times to 10'000 times faster, depending on the number of events: the more events there are to mine, the better the factor. This can be clearly seen in the following two figures which compare run times and throughput of the two prototypes:

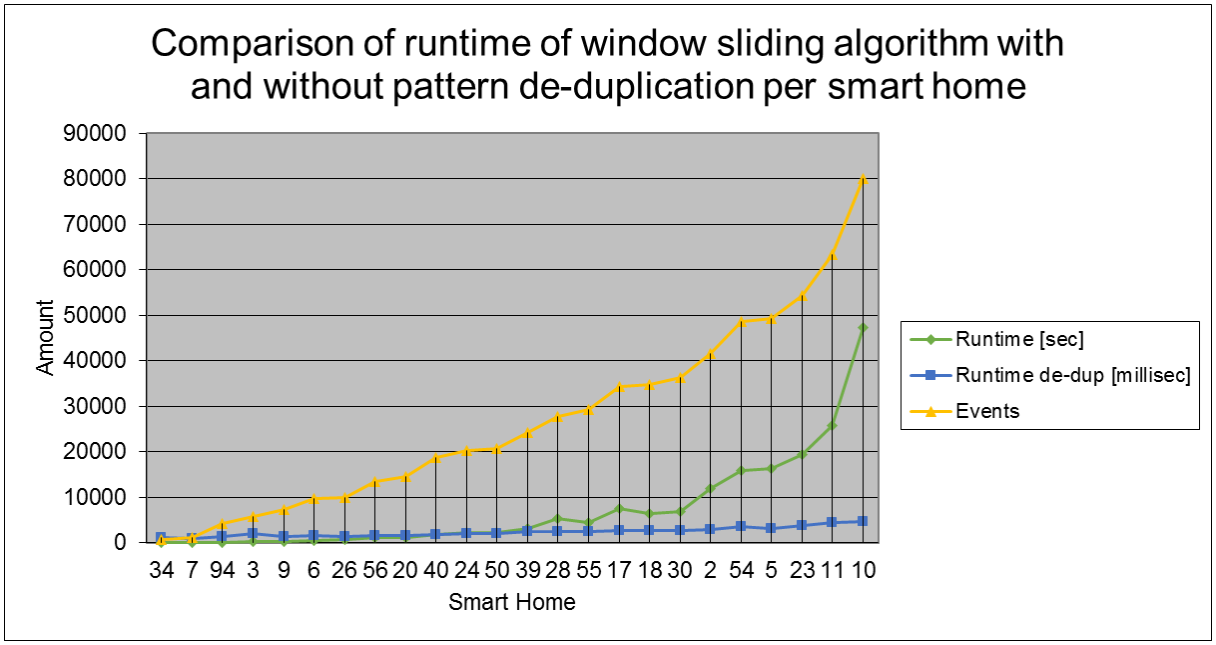


FIGURE 28: COMPARISON OF RUN TIMES BETWEEN THE TWO PROTOTYPES WSBF AND WSDD

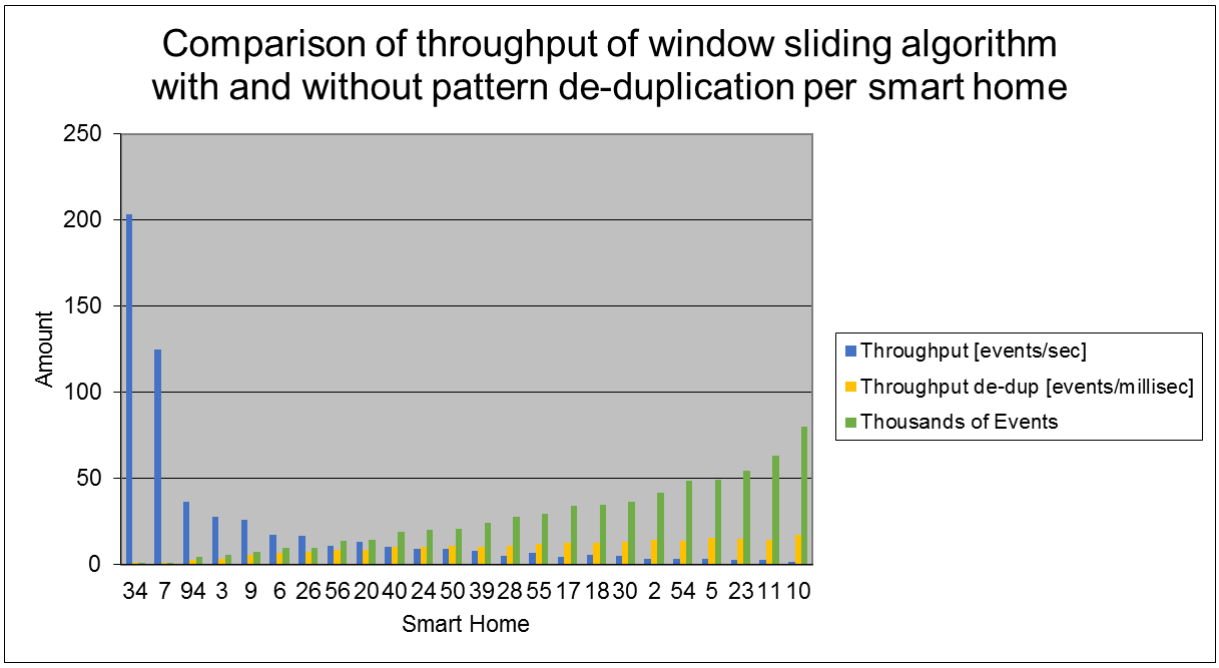


FIGURE 29: COMPARISON OF THROUGHPUT OF THE TWO PROTOTYPES WSBF AND WSDD

When comparing the throughput it has to be kept in mind that the depicted run times and throughputs of the WSBF algorithm are measured in seconds and events per seconds respectively, while the ones from the improved WSDD algorithm are measured in milliseconds and events per milliseconds respectively.

Besides the sheer improvement regarding the absolute run times and the throughput, the fact that the throughput is increasing with an increasing number of events is a good thing too, because it means that, while the overall run time increases with higher volumes of events, the relative time spent to mine a single pattern is decreasing and therefore the algorithm running faster.

5.3 TWO WELL-KNOWN SEQUENTIAL PATTERN MINING ALGORITHMS

Since a comparison of two self-written algorithms is not a scientific approach, the performance of the newly developed WSDD algorithm was compared with two well-known sequential pattern mining algorithms: BIDE+ and PrefixSpan. For that purpose an interface to the open-source Java framework SPMF, the Sequential Pattern Mining Framework by Philippe Fournier-Viger, et al. (2013), was implemented in the prototype. Besides this interface, some modifications had to be made to the prototype, which included a modularization of the WSDD prototype into the following five parts:

1. TestrobotSmartHomeMiner (includes the main method and calls certain combination of the four other parts)
2. DB connector
3. Pattern generator
4. Occurrence counter
5. Output writer

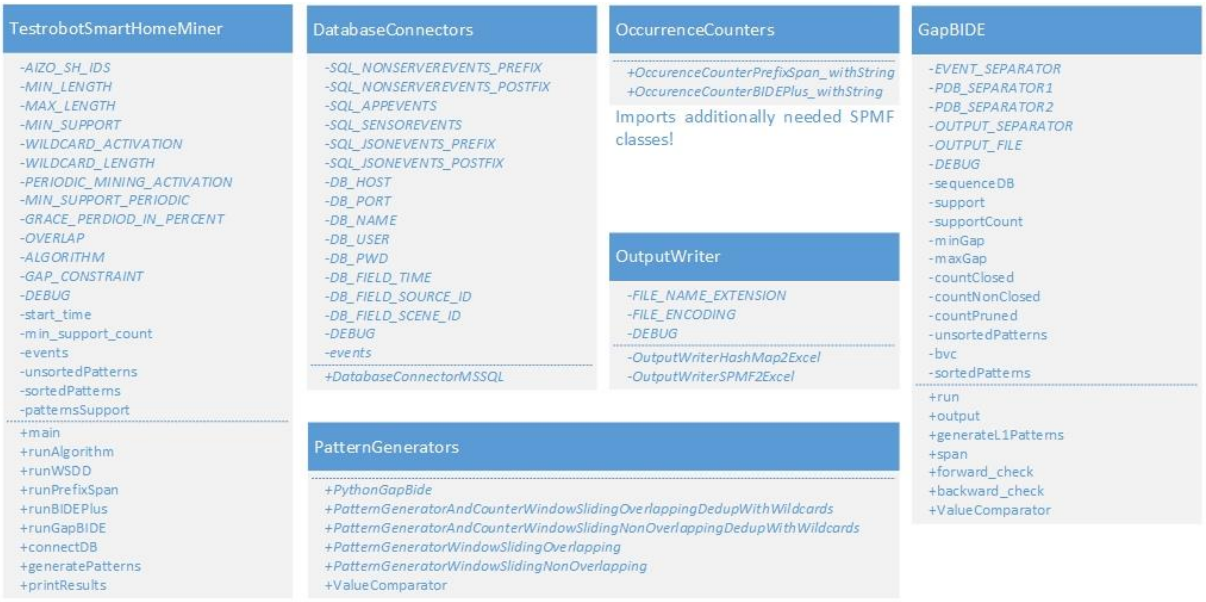


FIGURE 30: CLASS DIAGRAM OF THE MODULARIZED PROTOTYPE

These different classes have the following functionalities:

- TestrobotSmartHomeMiner is the class which is called when doing a test run with one of the algorithms. Different parameters like the smart home IDs, the minimum / maximum pattern length or the minimum support have to be specified. The main class is then invoking the runAlgorithm method which in turn calls the needed combination of:
  - DB connector (connectDB)
  - Pattern generator (generatePatterns)
  - Occurrence counter (done either in runWSDD, runPrefixSpan, runBIDEPlus or runGapBIDE)
  - Output writer (printResults)
- DatabaseConnectors provides a JDBC interface (via sqljdbc4.jar provided by Microsoft) to the MSSQL database in which the events form the Aizo smart homes are stored for this project. With minimal effort it would be possible to add interfaces to other DBMS like MySQL or Oracle.
- PatternGenerators offers mainly two static methods: one to generate overlapping patterns and one to generate non-overlapping patterns according to what Chikhaoui, et al. (2010) suggest. Additionally there are the special static methods PatternGeneratorAndCounterWindowSlidingOverlappingDedupWithWildcards and PatternGeneratorAndCounterWindowSlidingNonOverlappingDedupWithWildcards respectively. These are the WSDD implementation for overlapping and non-overlapping patterns. As described before, these methods do not only generate the patterns but also calculate the support count in one step. A fifth method is called PythonGapBIDE, which is needed to be able to call an algorithm written in Python and described later on. Finally there is a ValueComparator implemented which is needed to sort the patterns found by WSDD according to their support count, which makes it easier to analyze the patterns.
- OccurrenceCounters is the interface to SPMF and its BIDE+ and PrefixSpan implementation. Contrary to WSDD, which does the pattern generation and support counting in one step

(implemented as PatternGenerators.PatternGeneratorAndCounterWindowSlidingOverlappingDedupWithWildcards and PatternGenerators.PatternGeneratorAndCounterWindowSlidingNonOverlappingDedupWithWildcards), BIDE+ and PrefixSpan need two steps to achieve the same result. First the PatternGenerators (either for overlapping or non-overlapping patterns) generate multiple sequences / patterns out of the one continuous sequence of events as available in smart homes. These (non-)overlapping patterns are then transformed to a sequence database (as needed by SPMF) and this sequence database is finally mined for frequent sequential patterns. To be able to do this mining, additional classes from the SPMF framework are imported by the OccurrenceCounters class.

- Finally the two static methods of OutputWriters, OutputWriterHashMap2Excel and OutputWriterSPMF2Excel, are needed to output the mined results to a CSV file, which can be easily analyzed for example in Excel or any another spread sheet application.

The Class GapBIDE as well as the static variables WILDCARD_ACTIVATION, WILDCARD_LENGTH, PERIODIC_MINING_ACTIVATION, MIN_SUPPORT_PERIODIC, GRACE_PERIOD_IN_PERCENT and GAP_CONSTRAINT depicted in Figure 30 are described later on in this paper and are needed for wildcarding and periodic pattern mining (see the following chapters).

For the complete Java source code of all those classes, see the CD accompanying this paper.

To run tests with the open sequential pattern miner PrefixSpan and the closed sequential pattern miner BIDE+, the modularization mentioned above was implemented. While different settings for the pattern length, the amount of events and the minimum support produce different results, it became clear that regarding throughput and run times the developed WSDD algorithm is very competitive for the Aizo data set. Additionally it could be confirmed that the results produced by the two windows sliding algorithms, WSBF and WSDD, were correct.

Where the new algorithm is clearly outperforming both BIDE+ as well as PrefixSpan is when a low minimum support is needed. While all the algorithms like Apriori, BIDE+ or PrefixSpan, which are using pruning technics to keep the computational complexity at bay, are faster if higher minimum support values are specified, WSDD is classifying all patterns, regardless of their frequency, in a very short time. For example smart home no. 10's nearly 80'000 events (= tables app_button_clicks and sensor_events are excluded) are mined and classified in about 5 seconds while PrefixSpan needs 124 seconds and BIDE+ even 13'432 seconds to classify all the overlapping potential patterns of up to six events with a minimum support count of 2.

Even if the minimum support value is increased dramatically, e.g. to 0.01 which equals a support count of about 800 for smart home no. 10, the run times of WSDD are better (measurements in seconds):

Smart Home	10
WSDD	1
PrefixSpan	3
BIDE+	25

TABLE 15: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-6

More detailed benchmarks can be found in chapter 6.

The reason for the good run times of the WSDD algorithm compared to PrefixSpan and BIDE+ can be found in the fact that:

1. The algorithms uses a very efficient data structure and therefore is very fast.
2. The number of the different patterns is relatively small which allows the algorithm to store all the patterns without producing very big HashMaps that would slow down the algorithm.

An additional, domain specific advantage of WSDD is the possibility to specify the minimum (MIN_LENGHT) as well as the maximum length (MAX_LENGTH) of the patterns to be mined. This is not possible in traditional sequential pattern mining algorithms like PrefixSpan or BIDE+ and can be simulated only by employing the method described by Chikhaoui, et al. (2010) of producing overlapping or non-overlapping sequences of MAX_LENGTH out of the whole events available as one continuous sequence in smart home power consumption and event data. This producing of overlapping or non-overlapping sequences, however, has the disadvantage that the support of patterns with less than the maximum length is not correct. This can be illustrated by the following example where the first twelve events of a certain smart home are the following ones:

Event no.	Event
1	3726:498
2	3726:499
3	3726:460
4	3726:461
5	3726:456
6	**3726:455**
7	3726:458
8	3726:459
9	3726:473
10	3726:474
11	3726:467
12	3726:458

TABLE 16: SAMPLE DATA OF EVENTS FOR ILLUSTRATION OF SUPPORT MISCALCULATION

As can be seen, the single event no. 6 (3726:455) occurs exactly one time, which means that the 1-event pattern 3726:455 also occurs once. Its support count should therefore be one, which is what WSDD returns if it is mining the twelve events from Table 16 for 1- to 6-event(s) patterns.

However, if PrefixSpan or BIDE+ (or any other frequent pattern mining algorithm, which uses a sequence database as its input) mines the same smart home events for 1- to 6-event(s) patterns, the support count returned for the 1-event pattern 3726:455 is six. The cause of this miscalculation is the fact that PrefixSpan and BIDE+ need a sequence database as input. Such a sequence database can be generated out of the smart home data, which is one continuous sequence of events per smart home, by using the method of generating overlapping or non-overlapping sequences as described by Chikhaoui (2010).

If the overlapping method is applied to the events from Table 16, the first seven patterns created are:

Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	…
3726:498	3726:499	3726:460	3726:461	3726:456	**3726:455**	3726:458	…
3726:499	3726:460	3726:461	3726:456	**3726:455**	3726:458	3726:459	…
3726:460	3726:461	3726:456	**3726:455**	3726:458	3726:459	3726:473	…
3726:461	3726:456	**3726:455**	3726:458	3726:459	3726:473	3726:474	…
3726:456	**3726:455**	3726:458	3726:459	3726:473	3726:474	3726:467	…
3726:455	3726:458	3726:459	3726:473	3726:474	3726:467	3726:458	…

TABLE 17: OVERLAPPING PATTERNS FOR ILLUSTRATION OF SUPPORT MISCALCULATION

As a result of generating overlapping patterns, event no. 1 (3726:455) is now part of six patterns (patterns no. 1 – 6). PrefixSpan as well as BIDE+ count this event six times and return a support count of six instead of one. That a wrong support is returned by PrefixSpan and BIDE+ is true for all patterns which are not of maximum length, i.e. in the example above, patterns with a length of 1 to 5 events would have a wrong support and only 6-events patterns would have the correct support.

This problem could be lessened to a certain degree by using non-overlapping patterns, as did Chikhaoui (2010), where the event no. 6 is only part of pattern 1 and the support count is calculated and reported correctly by PrefixSpan and BIDE+ as 1. However, this method has a shortcoming of its own: it does not find all patterns in the smart home data. This can be seen when comparing Table 17 to the following listing of non-overlapped patterns:

Pattern 1	Pattern 2	…
3726:498	3726:458	…
3726:499	3726:459	…
3726:460	3726:473	…
3726:461	3726:474	…
3726:456	3726:467	…
3726:455	3726:458	…

TABLE 18: MISSING MAXIMUM-LENGTH PATTERNS WHEN USING NON-OVERLAPPING PATTERNS

As can be seen from such a comparison, the maximum length and overlapping patterns 2 to 6 (see Table 17) are missing from the list of non-overlapping patterns presented in Table 18 and only patterns 1 and 7 are produced by both methods, the generation of overlapping as well as non-overlapping patterns. However, the overlapping patterns 2 to 6 could be as interesting as pattern 1 and 7 and therefore the generation of overlapping patterns is to be preferred over the generation of non-overlapping when mining smart home data.

To illustrate the problem of missing patterns in detail, the following table is presenting the input and output for WSDD if non-overlapping patterns from another smart home are used:

	Input for WSDD		Output of WSDD			
Event	2-events pattern groups	3-events pattern groups	2-events patterns	Support	3-events patterns	Support
1	525:506	525:506	525:506, 521:507	1	525:506, 521:507, 520:508	1
2	521:507	521:507				
3	520:508	520:508	520:508, 520:509	1		
4	520:509	520:509			520:509, 3726:510, 520:508	1
5	3726:510	3726:510	3726:510, 520:508	1		
6	520:508	520:508				
7	525:506	525:506	**525:506, 521:511**	**1**	525:506, 521:511, 521:512	1
8	521:511	521:511				
9	521:512	521:512	521:512, 521:513	1		
10	525:506	525:506			521:513, 521:511, 521:513	1
11	521:511	521:511	521:511, 521:513	1		
12	521:513	521:513				

TABLE 19: MISSING NON-MAXIMAL PATTERNS WHEN USING NON-OVERLAPPING PATTERNS (WSDD)

Besides others, the overlapping pattern of length 2 consisting of events 2 and 3 (521:507, 520:508) or the overlapping pattern of length 3 consisting of events 2, 3, 4 (521:507, 520:508, 520:509) are missing from the output generated by WSDD.

While the statement, that certain patterns are missing when using non-overlapping patterns, is true for all maximal and for all non-maximal length patterns if WSDD is used, it is only true for all maximal length and for certain non-maximal length patterns if PrefixSpan and BIDE+ are used. This can be illustrated with the following table:

	Input for PF / B+	Output of PrefixSpan / BIDE+						
Event	3-events sequence database	2-events patterns	3-events patterns	2-events patterns	Support	3-events patterns	Support
1	525:506	1	3	1	525:506, 521:507	1	525:506, 521:507, 520:508	1
2	521:507	2			521:507, 520:508	1		
3	520:508		3		525:506, 520:508	1		
4	520:509	4	6	2	520:509, 3726:510	1	520:509, 3726:510, 520:508	1
5	3726:510	5			3726:510, 520:508	1		
6	520:508		6		520:509, 520:508	1		
7	525:506	7	9	3	**(525:506, 521:511)**	**(1)**	525:506, 521:511, 521:512	1
8	521:511	8			521:511, 521:512	1		
9	521:512		9		525:506, 521:512	1		
10	525:506	10(7)	12	4	**525:506, 521:511**	**2**	521:513, 521:511, 521:513	1
11	521:511	11			521:511, 521:513	1		
12	521:513		12		525:506, 521:513	1		

TABLE 20: MISSING NON-MAXIMAL PATTERNS WHEN USING NON-OVERLAPPIG PATTERNS (PS/B+)

As can be seen, the pattern consisting of events 2 and 3 (521:507, 520:508), which is missing from WSDD, is found by PrefixSpan and BIDE+. However, the pattern of length 2 consisting of events 3 and 4 (520:508, 520:509) as well as the overlapping pattern of length 3 consisting of events 2, 3, 4 (521:507, 520:508, 520:509) is missing here, too.

Looking at the support in Table 19 and Table 20 an additional issue appears: the pattern **525:506, 521:511** is reported to have a support count of 1 by WSDD, while PrefixSpan and BIDE+ report a correct support count of 2.

All these issues make a comparison of the results from WSDD and PrefixSpan and BIDE+ harder. To counteract this problem and to make the found patterns more comparable as well as to lessen the problem of the wrongly reported support for overlapping frequent patterns that are not of maximum length, a post-processing step was added to PrefixSpan and BIDE+, which is omitting all the found frequent patterns shorter than specified by minimum_pattern_length. However, only if the parameters minimum_pattern_length and maximum_pattern_length are identical, the results from all three algorithms are absolutely identical.

The remaining problem, i.e. to be able to specify different values for minimum_pattern_length and maximum_pattern_length, can be solved by running PrefixSpan and BIDE+ n times, where n = maximum_pattern_length – minimum_pattern_lenght + 1. For example if minimum_pattern_length = 2 and maximum_pattern_length = 7, it would be possible to run PrefixSpan or BIDE+ a first time with minimum_pattern_length=2 and maximum_pattern_length=2, then minimum_pattern_length=3 and maximum_pattern_length=3, …, and finally minimum_pattern_length=7 and maximum_pattern_length=7. This makes the results from all three algorithms absolutely identical and correct, also if different values for minimum_pattern_length=7 and maximum_pattern_length are specified. However, this also increases the run time for the mining process of PrefixSpand and BIDE+.

Since PrefixSpan and BIDE+ are already slower as it is (without calling those algorithms with different values for minimum_pattern_length / maximum_pattern_length to mine the same smart home), from a run time point of view it seems that what this project wants to achieve is best realized by using WSDD when mining overlapping patterns.

5.4 FREQUENT SEQUENTIAL PATTERN MINING WITH WILDCARDS

As explained in chapter 2.8, the idea behind wildcarding patterns is that there could exist interesting patterns which are very similar but vary in a few events (e.g. one or two) and because of this variation are not considered frequent. If a wildcard would now replace the event, which is varying, the frequency (the support count) would increase and the wildcarded pattern could become frequent.

Regarding wildcarding it has to be said that both PrefixSpan as well as BIDE+ support some kind of wildcarding. This can be seen if the previously presented example of 3-events patterns as input is looked at again. Here it becomes obvious that the third 2-events pattern (525:506, 520:508) was created by wildcarding the second event (512:507):

Input for PF / B+		Output of PrefixSpan / BIDE+					
Event	3-events sequence database	2-events patterns	3-events pattern	2-events patterns	Support	3-events pattern	Support
1	525:506	1 3	1	525:506, 521:507	1	525:506, 521:507, 520:508	1
2	521:507	2		521:507, 520:508	1		
3	520:508	3		525:506, 520:508	1		

TABLE 21: THE PSEUDO-WILDCARDING OF PREFIXSPAN AND BIDE+

However, unfortunately PrefixSpan and BIDE+ do not indicate whether a pattern is absolutely sequential and has no events, which were ignored (= wildcarded), in-between (patterns 1 and 2) or if there were events ignored (pattern 3). Additionally, neither PrefixSpan nor BIDE+ offer the possibility to disable the wildcarding.

With this in mind, a solution which is able to show if a pattern has wildcarded events in it was designed. The activity diagram of the extension for the WSDD algorithm to also mine frequent sequential patterns with wildcards looks like this:

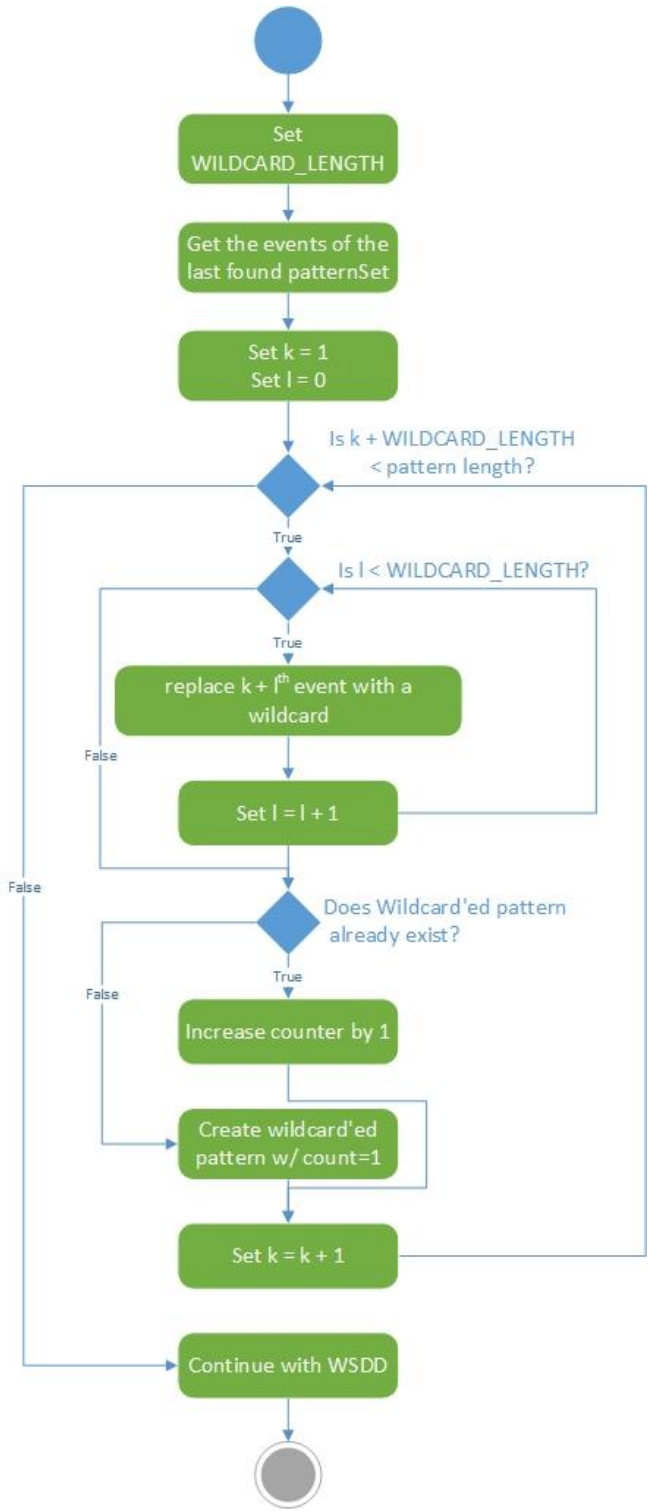


FIGURE 31: ACTIVITY DIAGRAM FOR THE EXTENSION OF WSDD WITH WILDCARDS

The above depicted extension of the WSDD algorithm, which takes place after every pattern mined by WSDD, does the following:

1. The user has to specify an additional parameter called WILDCARD_LENGTH. This parameter defines the length of the inserted wildcard.
2. Get all the events that make up the last found pattern and store them in an array.

3. Set a counter variable named k = 1, and another one named l = 0. Later on those counters are needed for looping through the patterns' events and replace the events with a wildcard character.
4. Check if k + WILDCARD_LENGTH is less than the pattern's length:
    a. if this is the case, the algorithm continues with step 5. For example if the last found pattern consists of three events, WILDCARD_LENGTH = 1, and k = 1 (which is the case when this gateway is reached for the first time for a certain pattern), then this condition evaluates as true and the algorithm continues with step 5.
    b. if it is not the case, the algorithm continues with step 10.
5. Check if l is less than WILDCARD_LENGTH:
    a. if this is the case, the algorithm continues with step 6. In the previous example l is 0 and WILDCARD_LENGTH is 1, therefore this would be true.
    b. if it is not the case, the algorithm continues with step 8.
6. Now the $(k + l)^{th}$ event is replaced with a wildcard. In the previous example, because of k + l = 1 + 0 = 1, the event at position 1 of the array with the events (which is the second event of a pattern) would be replaced with a wildcard. Because in the example the pattern consists of three events, the wildcarded pattern would now look like this: [event 1], *, [event 3].
7. The counter l is incremented by 1 and the algorithm continues with step 5.
8. The algorithm checks if the wildcarded patterns exists in the HashMap:
    a. if this is the case, the counter (which is the support count of a pattern) is increased by one. The algorithm would then continue with step 9.
    b. if it is not the case, the pattern is inserted as a new key into the HashMap and its counter initialized with 1. In the previous example, this would be the case for the pattern "[event 1], *, [event 3]". Therefore the key of the HashMap would be "[event 1], *, [event 3]" and the value 1. The algorithm would then continue with step 9.
9. The counter k is increased by 1 and the algorithm continues with step 4.
10. This step is only reached if step 4 evaluates as false. This is the case if k was increased by step 9 and is now equal to or bigger than pattern length - WILDCARD_LENGTH (or in other words k + WILDCARD_LENGTH is no longer smaller than pattern length). Now the rest of the WSDD algorithm continues with the rest of its execution, i.e. it mines the next pattern in the event data. After WSDD found another pattern, this triggers again the wildcarding steps 1 – 10.

As can be seen from the above explanations, the de-duplication and counting of wild-carded patterns is again realized with a HashMap (as is used for counting non-wildcarded frequent sequential patterns with WSDD), which keeps the algorithm fast. Since the implementation was done as an enhancement to the WSDD algorithm, the user can choose whether he / she wants to mine wildcarded patterns or not and if wildcarding is activated, the length of the wildcard needs to be specified.

If wildcarding is activated, WSDD needed to be compared to another algorithm able to mine sequential patterns with wildcards. There exist different such algorithms, e.g. MAIL by Xie, et al. (2010) or PMBC by Wu, et al. (2013), which both adhere to the one-off condition, or GapBIDE by Li, et al. (2012). Since the one-off condition as described by Wu, et al. (2013) is not interesting for mining smart home event data, it was decided to compare the WSDD when wildcarding is activated with GapBIDE. Since GapBIDE is an extension of BIDE+, which was already implemented for the mining of frequent sequential patterns in this project, it was expected that the run-times and other aspects of the two algorithms would be quite comparable.

Unfortunately, only after implementing GapBIDE in Java, it came to light that GapBIDE only offers the possibility to specify the wildcard length (similar to WILDCARD_LENGTH of WSDD) but that it does not report where a wildcard was found (which is a big disadvantage of PrefixSpan and BIDE+). This can be seen if the same input used in Table 20 for PrefixSpan and BIDE+ is used for GapBIDE and enhanced by the wildcard information:

Input for GapBIDE		Output of GapBIDE					Enhanced output
Event	3-events sequence database	2-events patterns		3-events patterns	2-events patterns	Support	2-events patterns with wildcards
1	525:506	1	3	1	525:506, 521:507	1	
2	521:507	2			521:507, 520:508	1	
3	520:508		3		525:506, 520:508	1	525:506, *, 520:508
4	520:509	4	6	2	520:509, 3726:510	1	
5	3726:510	5			3726:510, 520:508	1	
6	520:508		6		520:509, 520:508	1	520:509, *, 520:508
7	525:506	7	9	3	**(525:506, 521:511)**	**(1)**	
8	521:511	8			521:511, 521:512	1	
9	521:512		9		525:506, 521:512	1	525:506, *, 521:512
10	525:506	10(7)	12	4	**525:506, 521:511**	**2**	
11	521:511	11			521:511, 521:513	1	
12	521:513		12		525:506, 521:513	1	525:506, *, 521:513

TABLE 22: THE WILDCARD CAPABILITIES OF GAPBIDE

As can be seen, the 2-events patterns no. 3, 6, 9 and 12 are wildcarded patterns. As is the case with PrefixSpan and BIDE+, GapBIDE is not reporting where the wildcard was inserted but reports the patters as if they were a non-wildcarded pattern. To make clear what was expected from GapBIDE because it is needed to decide what is normal behavior in a smart home, the column "Enhanced output" was inserted in Table 22. It shows where the wildcard was detected.

One advantage that GapBIDE has over BIDE+ and PrefixSpan, is the possibility to disable wildcarding, as is the case with WSDD. This feature is missing from both PrefixSpan and BIDE+: they are always reporting wildcarded patterns and there is no way to deactivate this behavior besides using the aforementioned workaround of specifying an identical value for the parameters minimum_pattern_length and maximum_pattern_length in combination with the post-processing step, which omits all found patterns shorter than minimum_pattern_length. In other words: PrefixSpan and BIDE+ are always behaving as if a hypothetical minimum wildcard length of 1 and a maximum wildcard length = maximum_pattern_length would be specified. For GapBIDE the maximum wildcard length can be set to 0, which disables the wildcarding and lets GapBIDE return more similar results compared to WSDD.

Even though GapBIDE fails to fulfill the expectations regarding the information where wildcards were found in the patterns, it was decided to do experiments to compare WSDD and GapBIDE. Besides its better abilities to deal with the presentation of wildcards, WSDD showed promising results in a few initial benchmarks, for example when mining smart home no. 6 (measured run times in seconds):

Smart Home	6
WSDD	2
GapBIDE	10

TABLE 23: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-6

For more detailed information and additional benchmarks, see chapter 6.

5.5 PERIODIC SEQUENTIAL PATTERN MINING

The criteria for selecting patterns mentioned so far was only their frequency. However, it is probable that there exist not-so-frequent pattern which are still interesting when finding out what constitutes as normal behavior in a smart home because they are executed at a regular interval. Additionally, in the initial manual analysis of the event data, some periodic patterns were found.

Therefore the WSDD algorithm was extended to deal with periodic patterns in addition to frequent patterns. What is meant by a periodic pattern can be illustrated with the following table:

Event	100:200	…	…	…	100:200	…	…	…	100:200	…	…	…
Time	1.1. @ 8:05	…	…	…	2.1. @ 8:00	…	…	…	3.1. @ 8:05	…	…	…

TABLE 24: EXAMPLE OF A PERIODIC PATTERN

In the above example, the 1-event pattern 100:200 would be considered a periodic pattern because it always occurs at around 8 o'clock in the morning. Similar to frequent patterns, which are considered when establishing what constitutes as normal behavior of smart home users, also periodic patters could be considered. For example it would be interesting to know that the lighting is always turned on around 8:00 in the bedroom because this is when a smart home inhabitant is getting up. This knowledge could be used either for automation (increase of comfort), to notify someone if the pattern did not occur (e.g. in an assisted living home) or to save electricity (e.g. at 6:00 the smart home would normally begin with heating the building, but it is known that the inhabitant is only getting up at 8:00 while in the current season the sun rises at 7:00 already and is heating up the building enough during the one hour between 7:00 and 8:00 so that no extra heating is needed at 6:00).

For periodic patterns not only the pattern and its support count should be reported, but also what its interval is and when this interval starts. The interval is calculated as follows:

$$\frac{last\ timestamp\ the\ pattern\ occurred - first\ timestamp\ the\ pattern\ occurred}{number\ of\ occurrences - 1}$$

In the example above this would be: (3.1. @ 8:05 – 1.1. @ 8:05) / (3-1) = 2880 minutes / 2 = 1440 minutes = 24 hours.

The algorithm would next compare how many of the patterns occur at this interval. Since minor differences can occur between two occurrences, a grace period is also taken into account. For example the above three occurrences of the pattern 100:200 can be considered periodic even though a minor difference of five minutes in the occurrences can be observed. If a grace period of 10% of the calculated interval (in the example this would equal 1440 * 0.1 = 144 minutes) would be specified, the patterns would still be counted as periodic.

Finally to prevent a bias towards high frequency patterns, not the absolute count of periodic patterns but the fraction of the periodic patterns divided through the total number of the pattern occurrences (periodic as well as non-periodic) would be returned by WSDD.

For a better comprehension of how the detection of periodic patterns is realized, the used HashMap structure is explained in Table 25 and Table 26. The HashMap dedupedPatterns<String, Integer>, which is used by WSDD for storing frequent sequential patterns, contains only the pattern (as the key of the HashMap) and the count of occurrences of this pattern (as the value):

Pattern	Counter
Key	*Value*
100:200, 300:400, 500:600	27
…	…

TABLE 25: STRUCTURE OF HASHMAP DEDUPEDPATTERNS FOR PERIODIC PATTERN MINING

In the above example, the sequential pattern consists of the three events 100:200, 300:400 and 500:600 and occurs a total of 27 times. These information are sufficient when mining frequent sequential patterns: every time the same pattern is found in the event data, the counter is incremented by one, as explained before, and no further data needs to be tracked because in the end the keys and the values of the HashMap contain all the needed information (which are pattern and support count).

However, for periodic sequential pattern mining additional information is needed. This information need can be satisfied by the HashMap periodicPatterns<String, String[6]>, which is depicted next:

Pattern	Counter	Interval[min]	startOfInterval	allStart Timestamps	firstStart Timestamp	lastStart Timestamp
Key	*Value[0]*	*Value[1]*	*Value[2]*	*Value[3]*	*Value[4]*	*Value[5]*
100:200	20	1440	08:00	1.1.13 8:05, 2.1.13 8:00, 3.1.13 8:05, …	01.01.2013 08:00	03.01.2013 07:00
…	…	…	…	…	…	…

TABLE 26: STRUCTURE OF HASHMAP PERIODICPATTERNS FOR PERIODIC PATTERN MINING

- The Key is the same as for frequent pattern mining: the pattern itself.
- The first value is also familiar from frequent pattern mining: it is the support count (this absolute support count is later on changed to relative figure of occurrence of periodic patterns divided by the overall occurrence, i.e. periodic plus non-periodic occurrences, of this pattern to avoid a bias towards high-frequency patterns).
- The second value contains the calculated interval at which the pattern should occur if it is a periodic pattern.
- The third value stores the start of the interval as the time of day. This could be easily adapted to store a concrete date / time value if this should prove more suitable later on when automation of periodic patterns is planned and implemented.
- The fourth value contains all the timestamps at which the pattern occurs. This String could get quite large and it would perhaps be more suitable to use a list or some other data structure for storing these values. However, the benchmarks described later in this paper proved that Java could handle the long strings without problems.

- The fifth value is the timestamp when the pattern first occurred.
- The sixth and last value stores the timestamp when the pattern occurred last.

The HashMap periodicPatterns is used for periodic sequential pattern mining. As with wildcarding, WSDD was extended to be able to do periodic pattern mining. However, unlike wildcarding, the activities for periodic pattern mining is not executed after every frequent pattern found but only once after all the frequent patterns are found. The final version of WSDD is therefore looking as follows (Attention: the details about wildcarding are omitted here to keep the activity diagram short(er). For those details see either Figure 31 or the Java source code on the CD accompanying this paper):

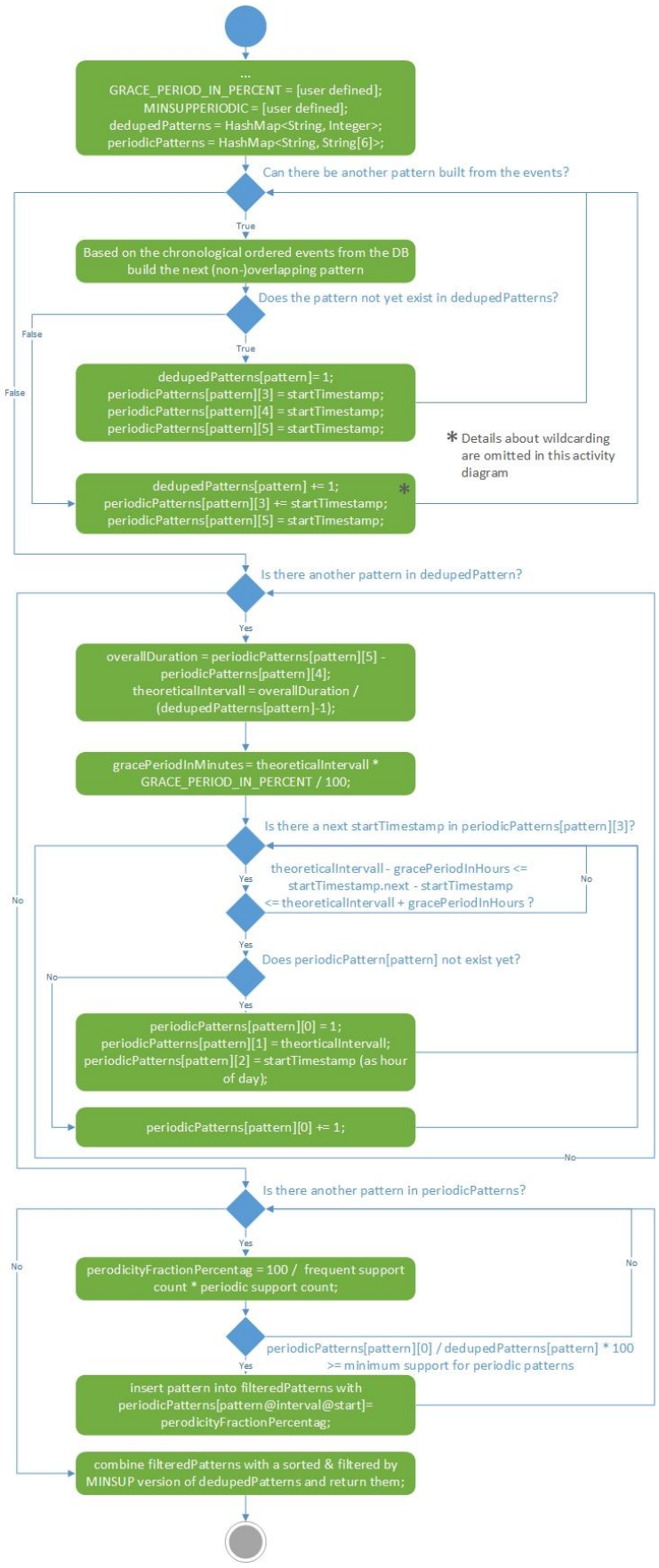


FIGURE 32: ACTIVITY DIAGRAM FOR THE EXTENSION OF WSDD FOR PERIODIC PATTERN MINING

1. The user has to specify two additional parameters called GRACE_PERIOD_IN_PERCENT and MINSUPPERIODIC (besides the previously mentioned parameters needed for frequent pattern mining like MIN_LENGTH, MAX_LENGTH or MINSUPCOUNT). Additionally, the two HashMaps dedupedPatterns (containing all the frequent patterns and their counts) as well as periodicPatterns (containing the additional data needed for periodic mining) are needed. Those HashMaps are filled later on during the frequent sequential pattern mining by WSDD.
2. First the algorithm checks if further patterns can be built from events retrieved from the database (this depends on the specified MIN_LENGTH and MAX_LENGTH of the patterns and from where in the continuous sequence the last pattern was built):
    a. if this is the case, this next pattern (either an overlapping or a non-overlapping one, depending on what the user wants) is being built. Afterwards the algorithm continues with step 3.
    b. if it is not the case, the algorithm continues with step 4.
3. Next, it is checked if the pattern does not yet exist in the HashMap dedupedPatterns:
    a. if this is the case, the pattern is inserted as a new key and the support count initialized with 1. Additionally, in the HashMap periodicPatterns the pattern is also inserted as a new key and the start timestamp of the pattern is recorded as the first start timestamp in the list of all timestamps (value[3]), as the first timestamp (value[4]) and also as the last timestamp (value[5]). The algorithm continues with step 2.
    b. if it is not the case, i.e. if the pattern was previously found and already exists as key in dedupedPatterns, the support count of that HashMap is increased by 1. Additionally, in periodicPatterns the value[3] and value[5] are updated with this pattern's timestamp. The algorithm continues with step 2.
4. This step is only reached after all the possible patterns were built. The algorithm checks now if there is another pattern in the previously filled HashMap dedupedPatterns, which contains all the found patterns and their support count:
    a. if this is the case, the algorithm now calculates the overall duration (overallDuration) of a pattern. This is the time elapsed between the first time the pattern occurred and the last time it occurred. Additionally, the theoretical interval (theoreticalInterval) is calculated by dividing the theoretical interval through the number of times the pattern was found minus 1. The algorithm then continues with step 5.
    b. if it is not the case, the algorithm continues with step 7.
5. Based on the theoretical interval and the user-specified GRACE_PERIOD_IN_PERCENT, the grace period in minutes is calculated.
6. Next, all the time stamps in value[3] are split. For each consecutive pair of timestamps n and n+1, n being a timestamp at which the pattern occurs, it is checked if the time difference between n+1 and n falls within the theoretical interval +/- the grace periods:
    a. if this is the case, a further check is executed to see if the pattern does not yet exist in periodicPatterns. If this is the case, the algorithm inserts the pattern as a new key into periodicPatterns, initializes the support counter with 1 and sets the start of the interval (value[2]) to the time-of-day of pattern n+1. If it is not the case (i.e. the pattern already exists as a key in periodicPatterns), the periodic support count is increased by 1. The algorithm then continues and repeats step 6 again to evaluate the next pair of timestamps (or with step 7 if the last pair was reached).

   b. if it is not the case, the algorithm continues directly with step 6 again by taking the next pair of timestamps to be evaluated (or with step 7 if the last pair was reached).
7. This step is only reached after every pair of timestamps (at which a pattern occurs) of every pattern is compared regarding periodicity. The HashMap periodicPatterns now holds as the keys every pattern and as the values the following information: value[0]=periodic support count, value[1]=the theoretical interval, value[2]=the time-of-day when the interval starts, value[3]=all the timestamps the pattern occurs, value[4]=the first time the pattern occurs and value[5]=the last time the pattern occurs. Now the algorithm checks if there is another pattern in periodicPatterns which is not yet processed:
   a. if this is the case, to avoid a bias towards high frequency patterns, the absolute periodic support count of the current pattern is divided by the support count of the pattern, which results in the fraction of patterns that occur at the same interval out of all the pattern occurrences (stored as perodicityFractionPercentage). The algorithm then continues with step 8.
   b. if it is not the case, the algorithm continues with step 9.
8. Now the algorithm checks if the perodicityFractionPercentage is equal to or greater than the user-specified MINSUPPERIODIC:
   a. if this is the case, the pattern along with the interval (value[1] of periodicPatterns) and the interval start (value[2] of periodicPatterns) is stored as the key of the HashMap filteredPatterns, while the value is the fraction of the periodic patterns (perodicityFractionPercentage). The algorithm continues with step 7.
   b. if it is not the case, the algorithm continues with step 7.
9. This step is only reached after it was checked for every pattern in periodicPatterns if it meets the minimum support for periodic patterns. filteredPatterns now contains all those periodic patterns that made the cut while dedupedPatterns contains all the frequent patterns. Therefore dedupedPatterns is now filtered by MINSUPCOUNT and combined with filterPatterns before both, periodic as well as frequent patterns are returned by WSDD.

To the best of the researcher's knowledge, there exist no well-known periodic sequential mining algorithms in the field of smart homes (i.e. none was found when reviewing literature for this project). While some sources mention that periodic patterns could be interesting in the field of smart homes, e.g. (Rashidi, et al., 2011), there seems to be no published algorithm's source code. Since the time frame of this research project did not permit to adapt and implement another generic algorithm for periodic sequential mining (as was done with PrefixSpan, BIDE+ and GapBIDE for frequent sequential pattern mining), the performance of the enhanced WSDD with periodic mining was only benchmarked against the WSSD without periodic mining. For the results of those benchmark, see chapter 6.

5.6 CONCLUSION OF CHAPTER 5

In conclusion it can be said that designing/implementing or adapting the algorithms WSDD, BIDE+, PrefixSpan as well as GapBIDE was a prerequisite for the benchmarks described in the next chapter. With the implementation a solid base for those benchmarks exists.

Since the different aspects of WSDD are divided into the different subchapters dealing with the respective aspects like wildcarding and periodic pattern mining, the following enumeration is going to summarize the main aspects of the newly developed algorithm:

- WSDD is an optimized brute-force algorithm for mining frequent sequential patterns in a continuous sequence of smart home event data (for the source code, see the CD accompanying this paper).
- In addition to that, WSDD is able to mine wildcarded frequent sequential patterns and periodic sequential patterns.
- The algorithm accepts and needs the following input parameters:
    - A list of all the events, ordered chronologically.
    - An integer MIN_LENGTH of the patterns to be mined, i.e. the minimum number of events that a pattern has to have.
    - An integer MAX_LENGHT of the patterns to be mined, i.e. the maximum number of events that a pattern can have.
    - An integer MINSUPCOUNT which is the minimum support count a pattern has to have to be considered frequent. This is applied as a post-processing step after all the patterns and their support count where mined: the ones that do not make the cut are just not reported by WSDD (even though the support count would be available).
    - A Boolean WILDCARD_ACTIVATION and an integer WILDCARD_LENGTH, which specify whether (in addition to the found frequent patterns) patterns with wildcards should be mined and how long a wildcard is.
    - A Boolean PERIODIC_MINING_ACTIVATION, a double MINSUPPERIODIC and an integer GRACE_PERIOD_IN_PERCENT which specify whether (in addition to frequent pattern mining) also periodic pattern mining should be done for the found patterns. MINSUPPERIODIC is the minimum support (not support count) a pattern must fulfill to be considered periodic while GRACE_PERIOD_IN_PERCENT specifies a percentage by which the interval between two consecutive occurrences of a pattern can vary.
- The algorithm returns a HashMap where the keys are the patterns and the values are the support counts of the patterns. If periodic mining is activated, the pattern as the key is enriched by the interval and the interval start information and the value is the fraction of the patterns which are periodic compared to the support count of the pattern.
- Since the amount of different patterns is not very large even though real-life data from Aizo was used in this research project, WSDD is surprisingly fast for a brute-force algorithm.

Additionally the following two restrictions were detected for BIDE+, PrefixSpan and GapBIDE:

- Of the four algorithms benchmarked in the next chapter, WSDD is the only one which fulfills the requirement of showing where a wildcard was inserted / detected in a pattern. While BIDE+, PrefixSpan and GapBIDE can deal with wildcards, they do not output the position where the wildcard occurs. This is a disadvantage of BIDE+, PrefixSpan and GapBIDE, which was only recognized after all the implementation effort was done. Additionally, for both PrefixSpan as well as BIDE+ wildcarding cannot be deactivated. Only GapBIDE offers this possibility, as does WSDD.
- WSDD is the only one of the four algorithms which calculates the correct support count in only one pass. The other three algorithms only calculate the correct support count for maximal-length patterns, a fact detected only after all the implementation effort was done and caused by the way

of building a sequence database out of the continuous sequence of smart home events proposed by Chikhaoui, et al. (2010).

Such restrictions are not completely surprising given the fact that the three algorithms PrefixSpan, BIDE+ and GapBIDE are generic ones not explicitly developed for frequent sequential pattern mining in smart homes.

While the two disadvantages mentioned above could not be remedied completely for this research project (the problem with the wildcard position could not be dealt with at all while for the wrong support count only a workaround was found), other aspects of the generic frequent sequential pattern mining algorithms could be adapted to deal with requirements of this research project. These include:

- The definition of a minimum length for the patterns. This was implemented as a post-processing step by omitting found patterns shorter than the user-specified parameter MIN_LENGTH.
- The Output is written in the CSV format, which makes the analysis in spread sheet applications much easier.
- And finally there is a workaround proposed to deal with the problem of the wrongly calculated support count for non-maximal patterns. This workaround proposes to run PrefixSpan, BIDE+ or GapBIDE multiple times for the same smart home events with different values for MIN_LENGT / MAX_LENGTH (e.g. five separate runs with the values 2, 3, 4, 5 and 6 respectively), where MIN_LENGTH = MAX_LENGTH. While this workaround eliminates the problem of the wrong support counts, it increases the run times.
  Another workaround, if one is willing to use non-overlapping patterns as input, is the usage of GapBIDE with wildcarding deactivated (by setting the maximum length of wildcards to 0). This also reports correct support counts but certain patterns are not found because of generating non-overlapping instead of overlapping patterns.

6 EXPERIMENTS & BENCHMARKS

6.1 INTRODUCTION TO CHAPTER 6

While the chapters 4 and 5 deal with manual data analysis and the design, adaption and development of algorithms to mine frequent and periodic sequential patterns, this sixth chapters is going to present the results of the benchmarks done with those algorithms.

For the benchmarks both the run times as well as the memory consumption was measured. The different algorithms were run multiple times and with various different parameter settings. To get representative results, a selection of five different smart homes with various amounts of events was chosen.

For each benchmark series the results are summarized in the following subchapters.

6.2 SETUP FOR THE EXPERIMENTS

All the following benchmarks were done on a PC workstation with the following specifications:

- OS: Windows 8 Pro x64
- CPU: Intel Xeon E3-1230 v3, 4 physical / 8 logical cores @ 3.3 GHz
- RAM: 16 GB, DDR3 @ 1866 MHz
- SSD: Samsung SSD 840, 540 MB/s sequential reading

- IDE: Eclipse, Kepler Service Release 1
- Java: 1.7.0_45 (64-Bit)
- Python: 2.7.5 (64-Bit)
- DBMS: Microsoft SQL Server 2012, Enterprise Edition (64-Bit)

Because it is not practical to edit the source code every time another manual experiment needs to be done, a tentative design for a simple GUI was designed and implemented to run manual tests and experiments:

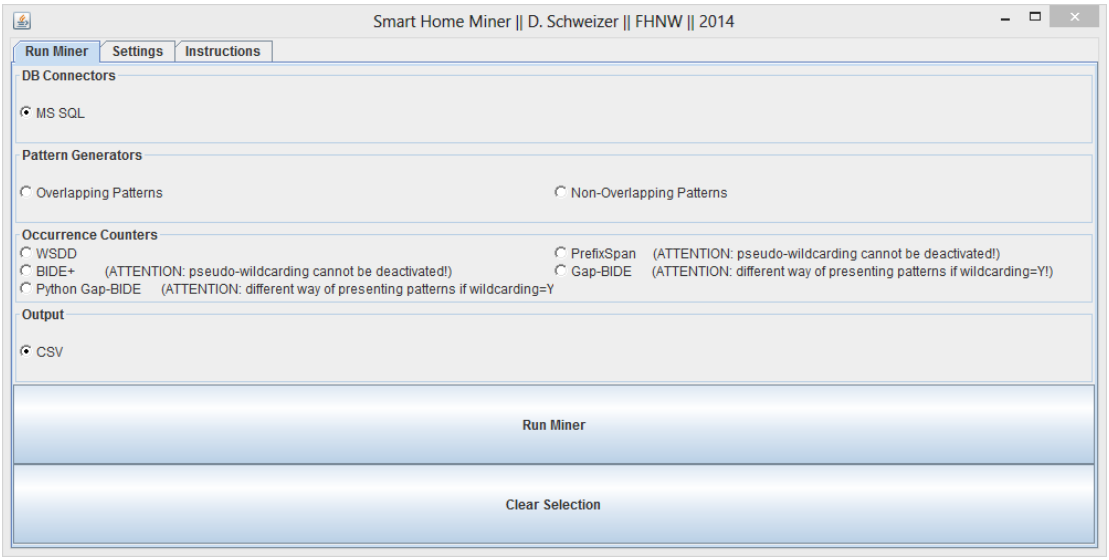


FIGURE 33: THE GUI FOR MANUAL TESTRUNS AND EXPERIMENTS

Additionally and for larger unattended benchmarking, where hundreds of different parameter sets were tried out, the previously mentioned TestrobotSmartHomeMiner class was used. The results were centrally collected for later analysis in Excel. This analysis is going to be presented in the following subchapters.

6.3 THE BENCHMARKS REGARDING RUN TIME

The four algorithms were compared regarding their performance when a minimum support of 0.01 was specified. For the initial benchmark non-overlapping patterns with a minimal length of 2 and a maximal length of 7 events were used. Where possible, i.e. for WSDD and GapBIDE, the wildcarding capabilities were disabled.

These parameter settings are considered as representative for normal mining needs in the Aizo smart homes because:

- The minimum support of 0.01 (or 1%) returns, for most of the smart homes of the Aizo dataset, only a few dozen patterns (the concrete amount depends on the smart home). This parameter setting allows the algorithms employing pruning technics (PrefixSpan, BIDE+ and GapBIDE) to run faster than when a low minimum support (e.g. 0.001 or less) is chosen. Higher values for minimum support (0.1 or above) produce no patterns for the Aizo data set and can therefore not be used.
- Frequent patterns consisting of less than two events are not interesting when trying to find out what constitutes normal behavior.

- Patterns of more than seven sequential events seem rather unlikely to occur on a day-to-day basis if one is considering the fact that manual events are what this research project is looking for.
- Non-overlapping patterns were used not because they are regarded as the better alternative, quite the contrary, but because this results in the lower number of patterns and therefore faster run times. These can be compared to the longer run times if overlapping patterns are used to study the impact a bigger input has on the run times.

The analysis includes the Aizo smart home with the fewest events (no. 7), three smart homes with an average amount of events (no. 50, 17 and 2) and the smart home with the most events (no. 11):

Smart Home	No. of events	Category
7	1173	Few
50	22498	Average
17	34485	Average
2	42129	Average
11	156121	High

FIGURE 34: THE THREE CATEGORIES OF SMART HOMES USED FOR THE BENCHMARKS

With the settings mentioned above, the following run times (in seconds) are achieved:

	7	50	17	2	11
WSDD	0	0	0	0	0
PrefixSpan	0	0	0	0	0
BIDE+	0	0	0	0	8
GapBIDE	0	0	2	0	73

TABLE 27: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7

This can be presented as the following line chart:

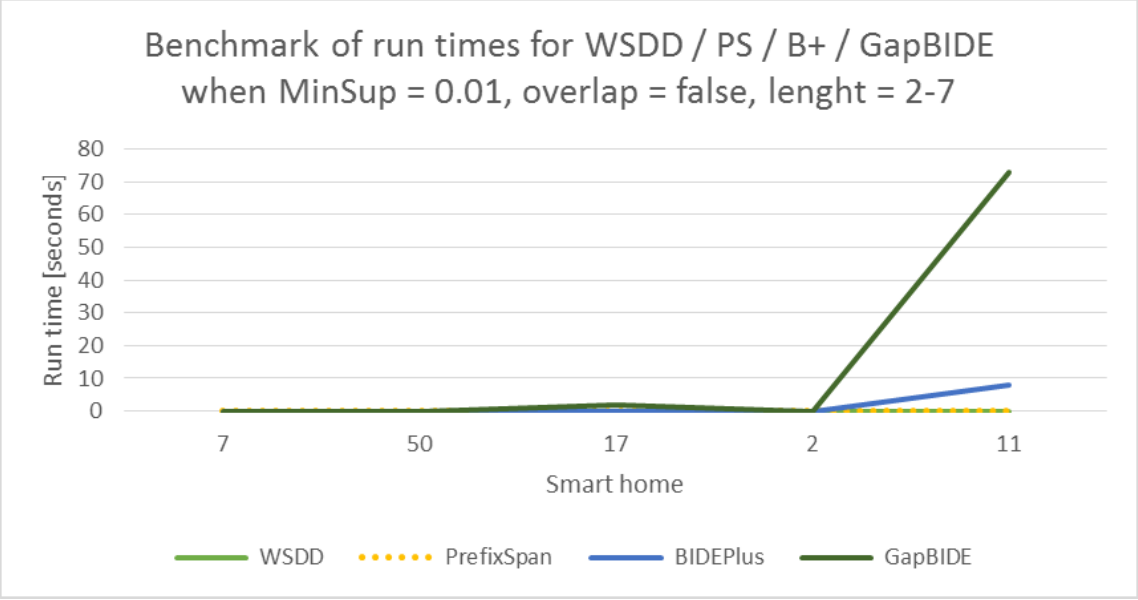


FIGURE 35: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7

As can be seen, for small and average amounts of events, all four algorithms show acceptable run times of under three second. However, the increase of events between smart home 17 and 11 is only handled well by WSDD and PrefixSpan. The other two algorithms, especially GapBIDE, are showing very long run times.

If smart home no. 11 as a representative of a smart home with a lot of events is ignored, the following picture presents itself:

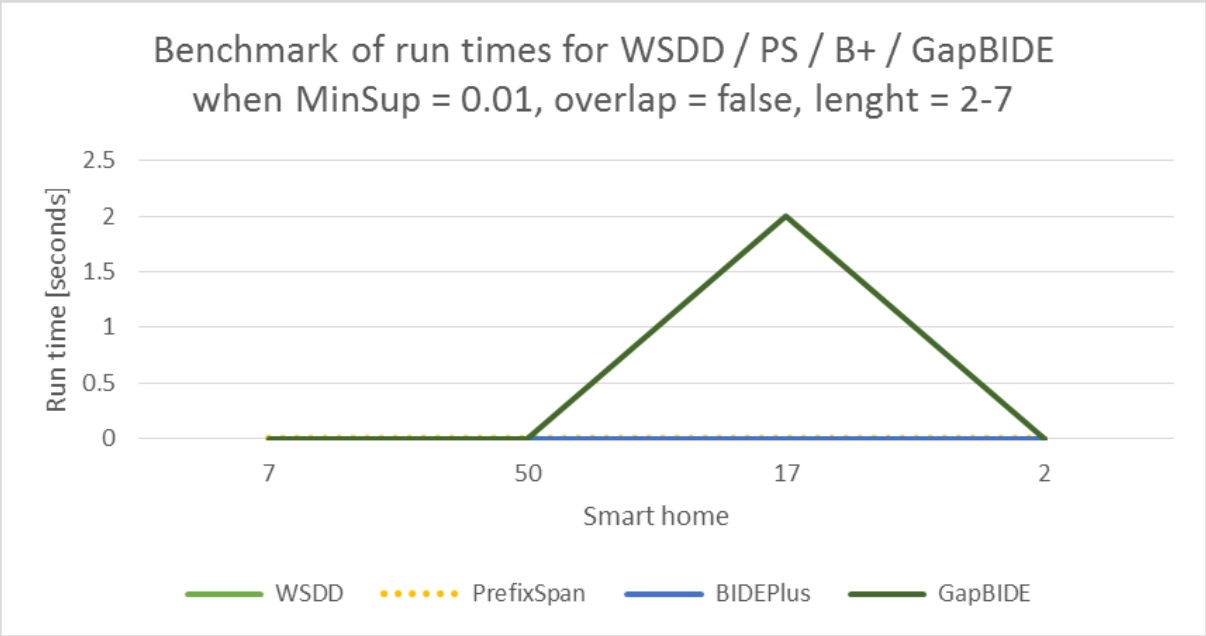


FIGURE 36: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7 (2)

If the minimum support is decreased from 0.01 to 0.001, the run times of BIDE+ and GapBIDE increase even further:

	7	50	17	2	11
WSDD	0	0	0	0	0
PrefixSpan	0	0	0	0	1
BIDE+	0	3	1	2	27
GapBIDE	0	5	3	5	92

TABLE 28: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=FALSE, LENGTH=2-7

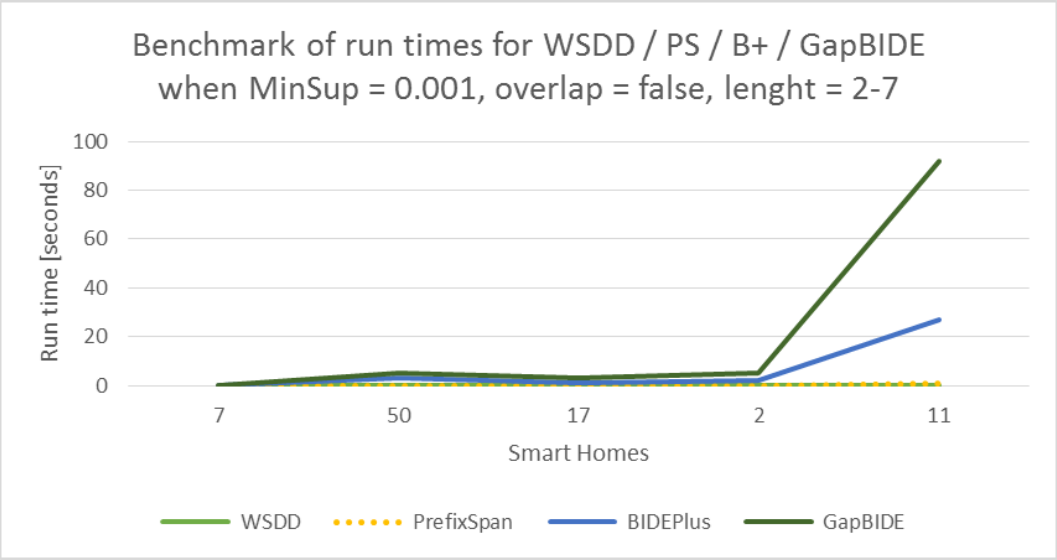


FIGURE 37: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=FALSE, LENGTH=2-7

Without no. 11, the diagram for the run times looks as follows:

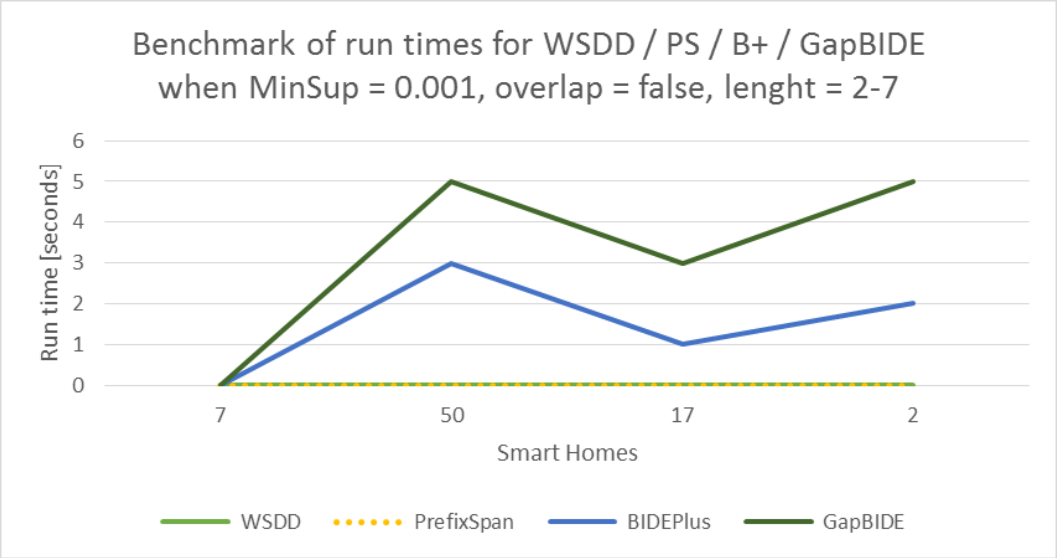


FIGURE 38: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=FALSE, LENGTH=2-7 (2)

As expected, the lowering of the minimum support increased the number of found patterns dramatically (shown in the following table are the number of patterns found by GapBIDE):

	7	50	17	2	11
MinSup = 0.01	40	19	3	4	12
MinSup = 0.001	824	252	159	176	138

TABLE 29: NUMBER OF PATTERNS FOUND BY GAPBIDE FOR MINSUP=0.01 AND MINSUP=0.001

This also explains why BIDE+ and GapBIDE have longer run times for smart home no. 50 than for smart home no. 17 even though there are more events in the database for the later: the number of found patterns is significantly higher for the first mentioned smart home (no. 50) and the pruning could be applied less effective, which increased the run time.

An even bigger impact than the decrease of the minimum support has the increase of the number of events by using overlapping patterns instead of non-overlapping patterns:

	7	50	17	2	11
WSDD	0	0	0	0	1
PrefixSpan	0	1	1	2	7
BIDE+	2	21	8	11	157
GapBIDE	1	246	131	249	4305

TABLE 30: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7

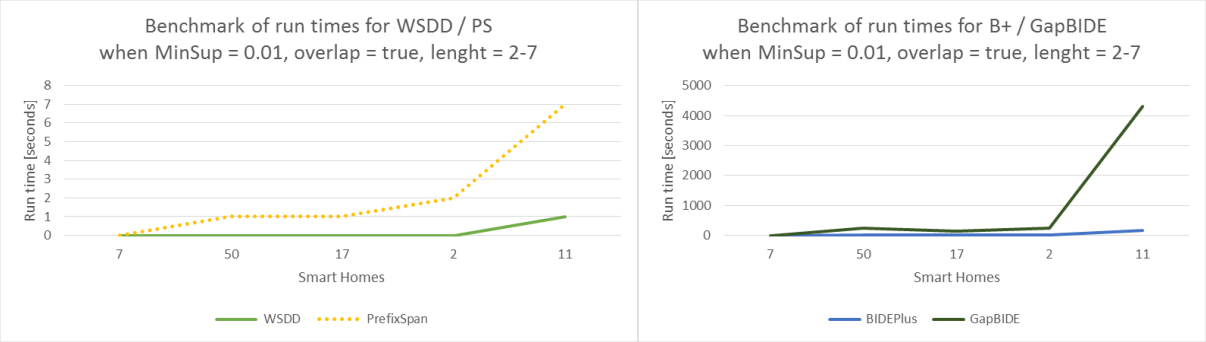


FIGURE 39: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7

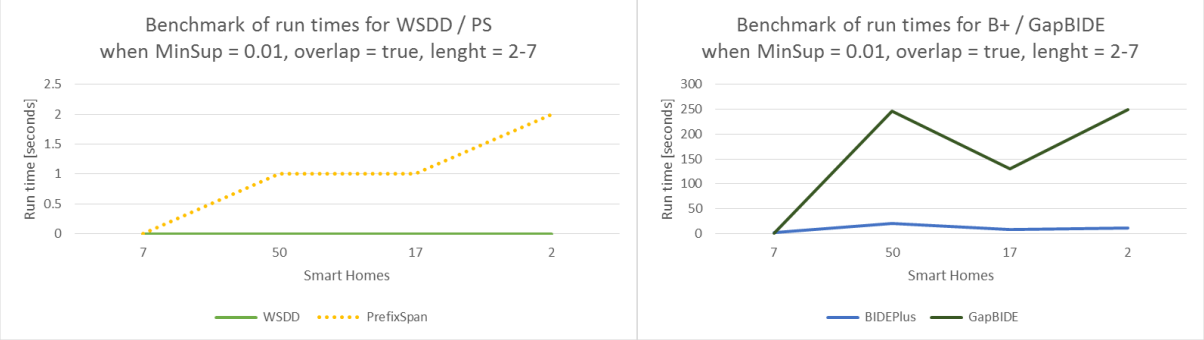


FIGURE 40: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7 (2)

Here the differences regarding run times are even bigger: while WSDD and PrefixSpan need less than 10 seconds for even the largest amount of events (smart home no. 11), BIDE+ needs already 157 seconds and GapBIDE even 4'305 seconds.

For the final benchmarking regarding run times the maximum length of the patterns was increased from 7 events to 10 events to see if and what effect this has. The results can be found in the following table:

	7	50	17	2	11
WSDD	0	0	0	0	1
PrefixSpan	0	0	0	0	1
BIDE+	1	4	0	0	45
GapBIDE	0	6	2	5	80

TABLE 31: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-10

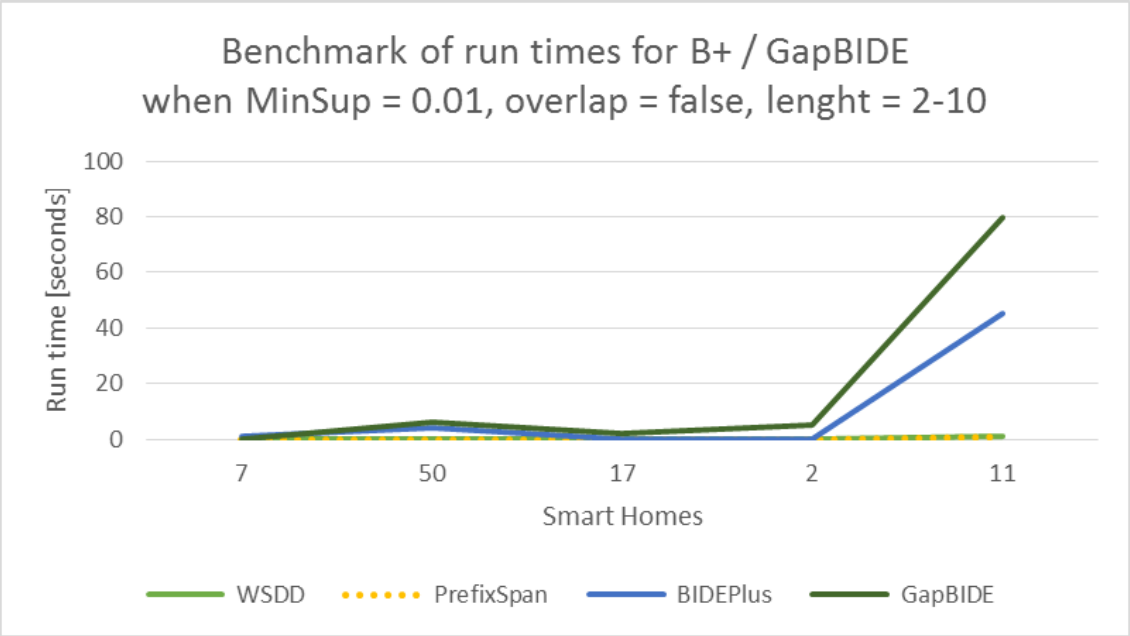


FIGURE 41: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-10

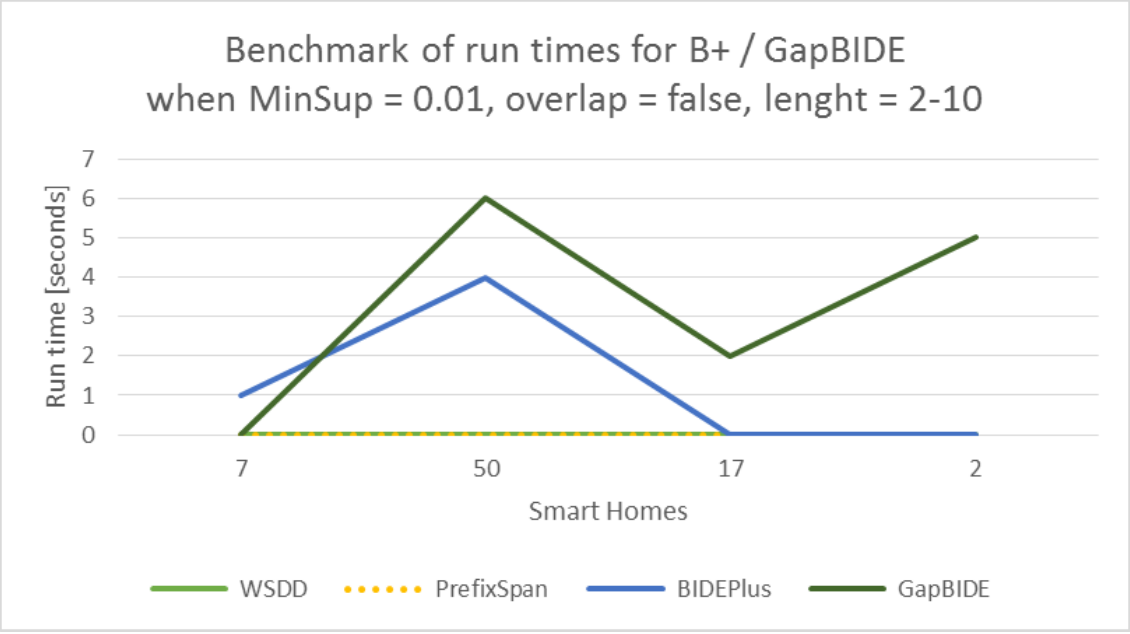


FIGURE 42: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-10 (2)

When compared to Table 27 it becomes obvious that the maximum length has an influence on BIDE+ and GapBIDE, but practically no effect on WSDD and PrefixSpan.

When reflecting upon the achieved results, it can be said that some of them were expected:

- GapBIDE as an extended and more complex version of BIDE+ was expected to have longer run times than BIDE+.
- The dependency of the run times on the number of events was also to be expected: the more events, the longer the run time. Some small discrepancies can be observed, e.g. it takes BIDE+ longer to mine the 22'498 events of smart home no. 50 than for the 42'129 events of no. 2.

However, the general trend and correlation is clearly noticeable. Some of the discrepancies may be attributed to the number of the found frequent patterns because less frequent patterns mean more pruning and therefore also less time needed for mining.

- The dependency of the run times on the minimum support was to be expected for PrefixSpan, BIDE+ as well as GapBIDE. Since those three algorithms are employing pruning technics, the higher the minimum support, the more pruning is allowed and the faster the algorithms run. Only WSDD, which does not do any pruning, is unaffected by the specified minimum support.
- The correlation of the run time versus the pattern length was also to be expected. Based on the various experiments and benchmarks done regarding run times, the following correlation coefficients of the run time and the previously mentioned variables (minimum support, pattern length and number of events in a smart home) could be calculated:

Correlation coefficient r for RunTime VS MinimumSupport		**Classification[1]**
WSDD	Division by Zero	---
PrefixSpan	-0.88489	Strong correlation
BIDE+	-0.91917	Very strong correlation
GapBIDE	-0.88933	Strong correlation
Correlation coefficient r for RunTime VS PatternLength		**Classification**
WSDD	Division by Zero	---
PrefixSpan	0.924222	Very strong correlation
BIDE+	0.924018	Very strong correlation
GapBIDE	0.97726	Very strong correlation
Correlation coefficient r for RunTime VS EventsInSmartHome		**Classification**
WSDD	Division by Zero	---
PrefixSpan	0.965156	Very strong correlation
BIDE+	0.810047	Strong correlation
GapBIDE	0.803043	Strong correlation

TABLE 32: VARIOUS CORRELATION COEFFICIENTS FOR THE MEASURED RUN TIMES

- While the dependency on the number of found frequent patterns was something that was not anticipated, it appears very logic in hindsight: if there are roughly the same amount of events to be mined and the minimum support as well as the pattern length are the same, the run times of PrefixSpan, BIDE+ and GapBIDE depend on the number of found patterns. For WSDD the number of found pattern has practically no impact because there is no pruning technic involved that could profit from a lower number of frequent patterns found.

Other results were rather unexpected:

- Fournier, et al. (2013) mention that closed sequential pattern mining algorithm often are faster than open sequential pattern mining algorithm. It was therefore surprising that PrefixSpan (open) was always faster in mining the smart home events than BIDE+ (closed).

[1] Classification according to Zöfel (2003, p. 151)

- The increase of the maximum length from 7 to 10 events has nearly no effects on WSDD and PrefixSpan regarding the absolute run time: while a strong correlation is measureable for PrefixSpan, the absolute increase in seconds is very small.
- The generally acknowledge reasoning is that a pruning strategy is needed to deal efficiently with large amounts of data when doing data mining. That's the reason for the existence of the various algorithms, like Apriori, BIDE+ or PrefixSpan, which are available for pattern mining. However, for the real life data set of Aizo containing the smart home events, the experiments show that an optimized brute-force approach like WSDD is sufficiently fast for sequential pattern mining of smart home events and that it sometimes even outperformances the other, more elaborate algorithms.
- That GapBIDE scales as bad is it does compared to BIDE+ was unexpected. The most probably explanation in the eyes of the research is that the Java code written for this research project is not efficient for a high number of events, which causes the long run times. Since this bad behavior cannot be observed in the original prototype of GapBIDE written in the programming language Python, which is provided by Li, et al. (2012), the cause for this most probably lies somewhere in the Java code written for this research project. Because the results reported by the Python version of GapBIDE are absolutely identical with the ones reported by the Java version of GapBIDE written for this research project, it can be concluded that the code must have been translated correctly but inefficiently.

In summary it can be said that, at least regarding its run time, WSDD is a viable option for frequent sequential pattern mining.

6.4 THE BENCHMARKS REGARDING MEMORY CONSUMPTION

In addition to the run time benchmarks, the same four algorithms were compared regarding their memory consumption. The memory consumption was measured with the "MemoryLogger", which is provided as part of the SPMF framework. For the initial benchmark, the same parameter settings as for the run time comparisons were used:

- A minimum support of 0.01 was specified
- Non-overlapping patterns with a minimal length of 2 and a maximal length of 7 events were used.
- For WSDD and GapBIDE the wildcarding capabilities were disabled (this is not possible for PrefixSpan and BIDE+).

The achieved results for the benchmark with these aforementioned setting look as follows (measurements are in Megabytes):

	7	50	17	2	11
WSDD	25.6	29.1	37.8	70.8	147.9
PrefixSpan	28.1	44.5	24.4	68.1	184.8
BIDE+	64.1	265	67.9	146.1	1340.5
GapBIDE	62.7	1242.6	1223.9	1342.2	1249.4

TABLE 33: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7

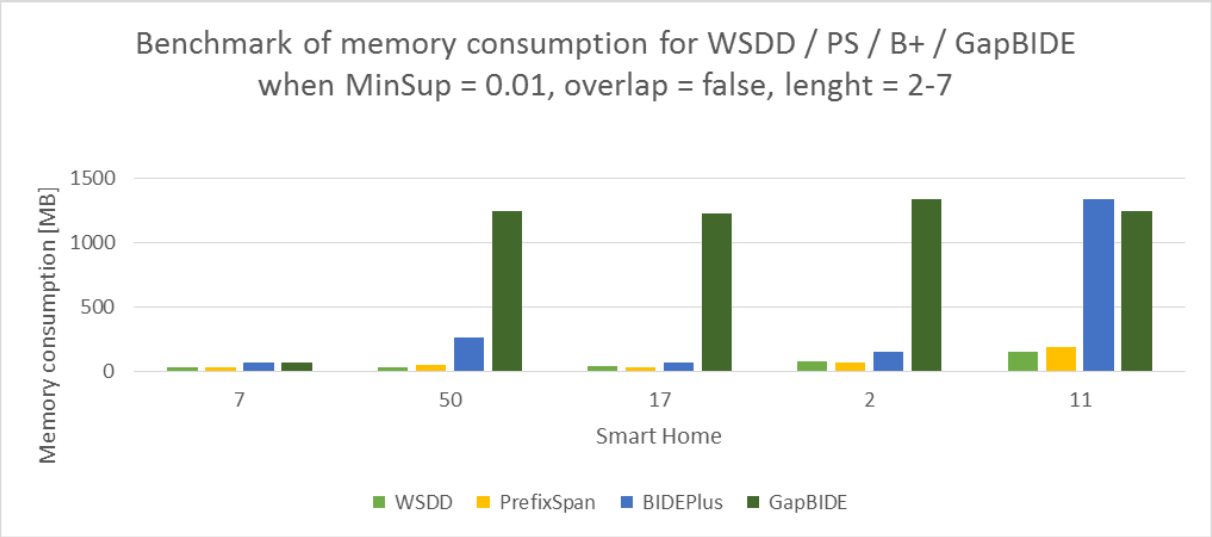


FIGURE 43: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7

As with the run times, in this initial benchmark both WSDD and PrefixSpan show good and similar results while BIDE+ and GapBIDE show worse results. GapBIDE consumes very large amounts of memory, which could be part of the explanation why the previously measured run times are that bad.

For the benchmarks presented next, the minimum support was lowered to 0.001, as was done for the run time benchmarks. The expected results were that the memory consumption, except for WSDD which always classifies all the patterns, should increase, since pruning could not cut back / omit as much patterns as it can when the minimum support is 0.01:

	7	50	17	2	11
WSDD	29.4	31.1	38.3	71.6	148.4
PrefixSpan	38.4	70.9	70.4	72.4	293.3
BIDE+	131.6	833.3	285.1	533.3	1344.5
GapBIDE	64	1246.4	1218	1290.2	1287.7

TABLE 34: MEMORY BENCHMARK FOR MINSUP=0.001, OVERLAP=FALSE, LENGTH=2-7

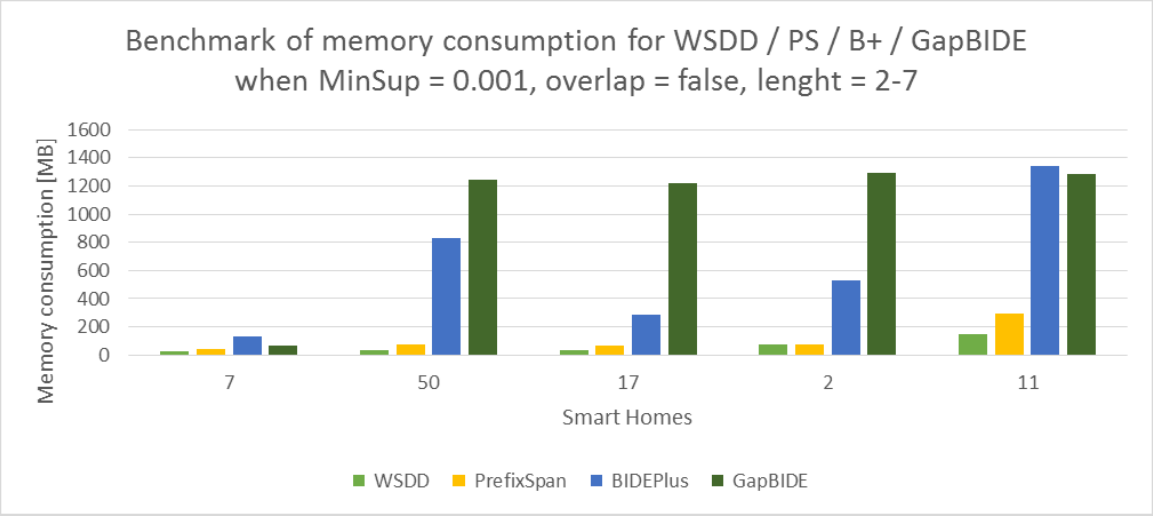


FIGURE 44: MEMORY BENCHMARK FOR MINSUP=0.001, OVERLAP=FALSE, LENGTH=2-7

Again, WSDD and PrefixSpan deliver good results, while BIDE+ and GapBIDE lag behind. The expected increase of memory consumption for PrefixSpan and BIDE+ is visible for all smart homes. GapBIDE continues to consume disproportional amounts of memory.

The higher memory consumption for smart home no. 50 (22'498 events) compared to no. 17 (34'485 events) can be explained by the higher number of found patterns for no. 50 (exemplary results from BIDE+):

	50	17
MinSup = 0.01	71	10
MinSup = 0.001	1132	504

TABLE 35: NUMBER OF PATTERNS FOUND BY BIDE+ FOR MINSUP=0.01 AND MINSUP=0.001

For the next benchmark the number of events to mine was increased by choosing overlapping instead of non-overlapping events. The results for this look as follows:

	7	50	17	2	11
WSDD	30.8	66.7	127.6	48.3	498.1
PrefixSpan	56.3	190.6	194.6	298.5	828
BIDE+	518.6	1300.4	1355.9	1358.9	1715.9
GapBIDE	487.2	1042.5	1141.8	1327.3	1531

TABLE 36: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7

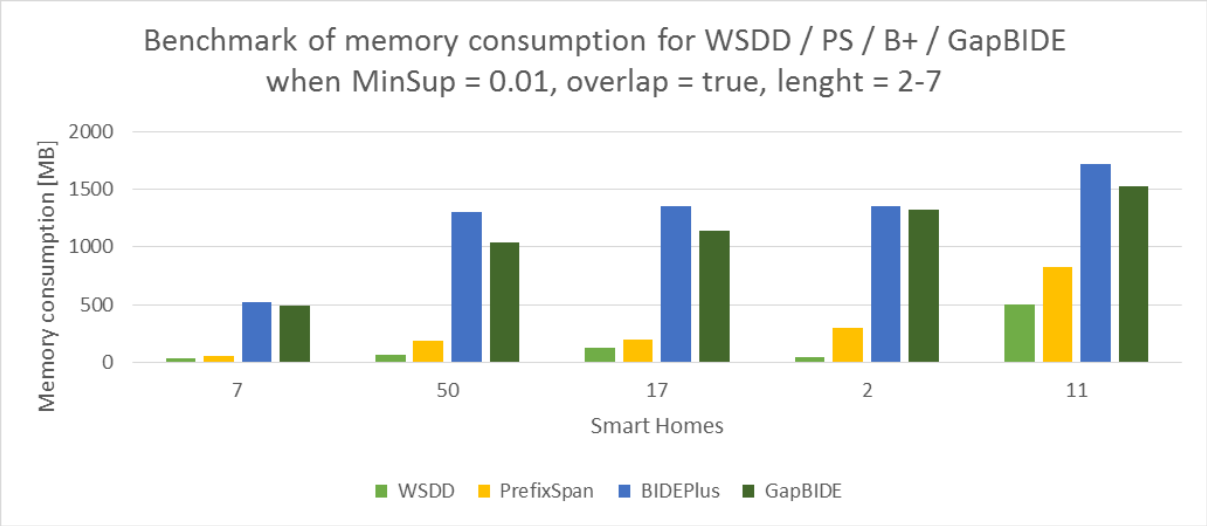


FIGURE 45: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7

As expected the memory needed increases for WSDD, PrefixSpan and BIDE+. The memory consumption for GapBIDE stays unchanged high, which is a further indication that the implementation of this algorithm in Java is inefficient.

The last benchmark was again to compare the performance if the maximum pattern length was increased from 7 to 10 events. With these adaptations, the following amount of memory was consumed by the different algorithms:

	7	50	17	2	11
WSDD	26.8	49.6	67.5	42.7	263.4
PrefixSpan	33.2	70.7	29.3	72.5	293.1
BIDE+	68.9	1319.8	137.8	166.9	1119.6
GapBIDE	64	1270.3	1295.3	1241.2	1328.2

TABLE 37: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-10

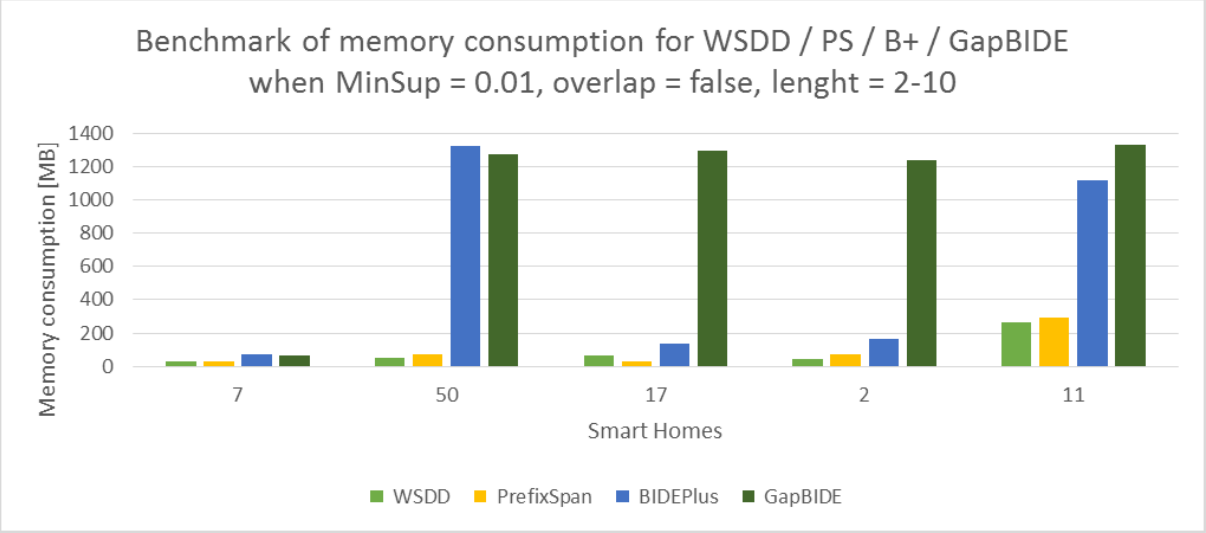


FIGURE 46: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-10

It can be seen from this benchmark that the increase of the maximal patter length can have an impact on the memory needed by the different algorithms. Especially BIDE+ is showing big differences in this regard. The previously noticed difference between smart homes no. 50 and no. 17 is increasing even further for 2- to 10-events patterns.

As can been seen from the achieved results, WSDD and PrefixSpan are again significantly better than the two BIDE algorithms. WSDD uses a little less memory in general, but the differences are quite small in this regard, except when there are a lot of events (e.g. smart home no. 11 or when overlapping patterns are used): in those situations WSDD uses up to 40% less memory.

A possible action to improve the memory consumption, regardless of the used algorithm, would be the use of a translation table. With such a translation table the more memory intensive Strings, which make up the events in the Aizo data, could be translated to less memory intensive Integers. A translation table would hold for each unique and distinct event one entry and could look something like this:

Original event as a String	Translation of event to an Integer
752:18	1
5927:113	2
3726:6487	3
...	...

TABLE 38: EXAMPLE OF A TRANSLATION TABLE TO REDUCE THE MEMORY CONSUMPTION

The creation of such a translation table could be done either as a preprocessing step in the database, in an additional piece of software, which would only do the translation, or as part of the algorithms themselves.

It can be concluded that also from a memory consumption point of view, WSDD shows good results. As with run times, the only algorithm that can achieved similar results is PrefixSpan. However, the achieved results regarding memory consumption are not as good as the ones for the run times.

6.5 THE BENCHMARKS WHEN WILDCARDING IS ACTIVATED

Besides measuring run times and memory consumption, benchmarks with different settings for the wildcarding parameters were conducted. Since only WSDD and GapBIDE support true wildcarding, only those two algorithms were compared. Additionally, since the run times of the Java version of GapBIDE, written for this research project, showed some bad results (because of some unknown inefficiencies in the Java code, which could not be pinpointed during the limited time frame of this master thesis), it was decided that the original source code of GapBIDE, written in the programming language Python by Li, et al. (2012), shall be used in these benchmarks. This allows a more meaningful comparison of the performance of WSDD and GapBIDE, because the bad performance for smart homes with a lot of events is not present in the Python version of GapBIDE.

This made it necessary to call the Python program from within the Java code. The measurements of run times is still possible, since Java can wait for the execution of an external process to end before continuing with the next Java statement. However, the way this research project measures the memory consumption, this is by using the MemoryLogger provided by the SPFM framework, is not possible since the mining process takes place outside of Java in Python. Therefore, the memory consumption for the Python version of GapBIDE was measured with the Performance Monitor of Windows 8 (Counter: Private Bites, Process: python.exe), which provides accurate and precise enough results to make a comparison of the memory consumption possible.

For the first benchmark the same settings as for some of the previous benchmarks were used. These are:

- A minimum support of 0.01
- Non-overlapping patterns with a minimal length of 2 and a maximal length of 7 events

However, this time wildcarding was activated and the wildcard length set to 1. The trial runs with these settings showed the following results in seconds:

	7	50	17	2	11
WSDD	0	1	1	1	2
GapBIDE	0	3	2	4	41
PythonGapBIDE	0	1	1	1	2

TABLE 39: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, WILDCARD=1

These run times can be visualized as follows:

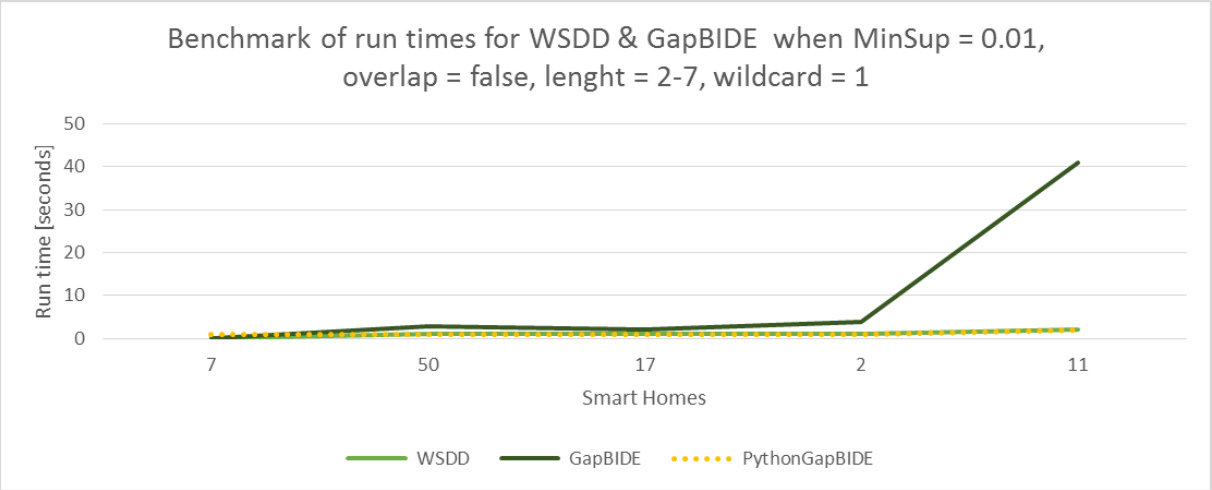


FIGURE 47: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, WILDCARD=1

As can be seen, WSDD is still the fastest available algorithm for mining patterns with wildcards, however PythonGapBIDE is as fast as WSDD. Additionally, the inefficiencies of the Java code for GapBIDE become visible for smart home no. 11 where the Python version of GapBIDE needs 2 seconds while the Java version needs over twenty times as long: 41 seconds.

When looking at the memory consumption, for the first time WSDD is clearly outperformed. For the chosen settings, the pruning technics of GapBIDE implemented in Python prove very effective regarding memory consumption (measurements in Megabytes):

	7	50	17	2	11
WSDD	38.4	38.8	119.3	207	599.4
GapBIDE	61.4	1302.8	1060.2	1294.9	1324.1
PythonGapBIDE	5	8.6	9.4	13.5	50.8

TABLE 40: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, WILDCARD=1

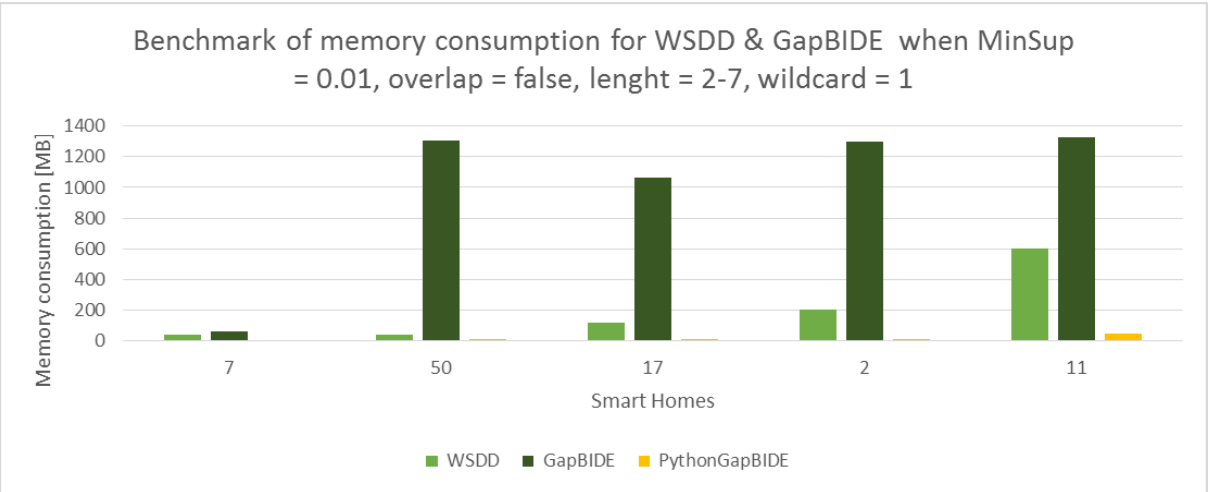


FIGURE 48: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, WILDCARD=1

When looking at the memory consumption one has to keep in mind that the measurements for the two Java algorithms also contain some memory overhead (up to about 300 MB) not directly associated with the mining process itself which is not contained in the measurements of the PythonGapBIDE. Additionally, the used method for measuring memory consumption in Python (Windows Performance Monitor) is not as precise as is MemoryLogger and might miss memory peaks because it cannot measure the memory consumption in lower intervals than 1 second. Besides that, the achieved results by PythonGapBIDE are impressive and outperform WSDD clearly regarding memory consumption.

For the next benchmark both the number of events was increased by using overlapping instead of non-overlapping patterns and the minimum support was lowered to 0.001. This produced the following results:

	7	50	17	2	11
WSDD	0	2	2	3	9
GapBIDE	1	133	76	162	2089
PythonGapBIDE	1	2	3	3	12

TABLE 41: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=1

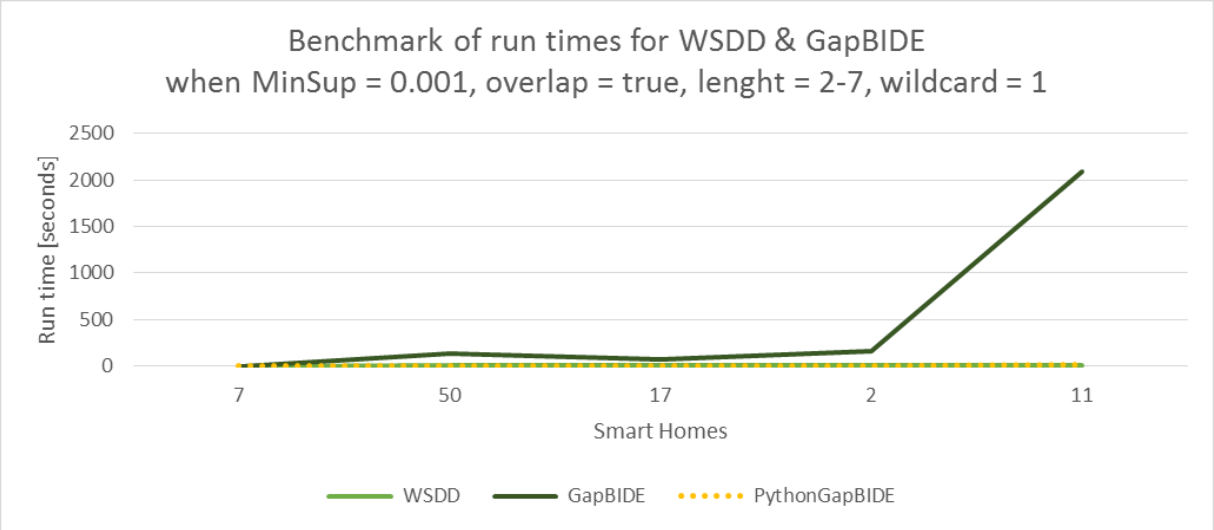


FIGURE 49: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=1

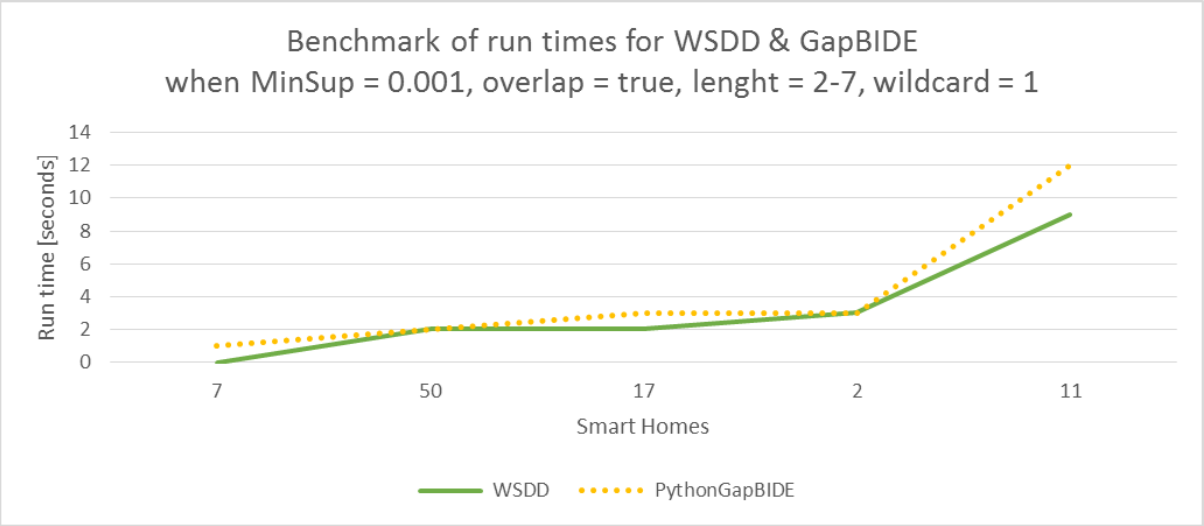


FIGURE 50: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=1 (2)

When regarding the results it can be seen that WSDD is most of the times a little faster than GapBIDE implemented in Python. The Java implementation of GapBIDE shows again that something is wrong with the Java code: the last benchmark for smart home no. 11 took over 2'000 seconds to finish.

From a memory consumption point of view an interesting thing to consider is that the increase of the events and the lowering of the minimum support favors WSDD much more than it does GapBIDE:

	7	50	17	2	11
WSDD	41.8	117.2	472.6	762.1	1463.3
GapBIDE	259.1	849.8	1193.9	1189	1467.2
PythonGapBIDE	5.3	47.9	68.2	79.4	823.9

TABLE 42: MEMORY BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=1

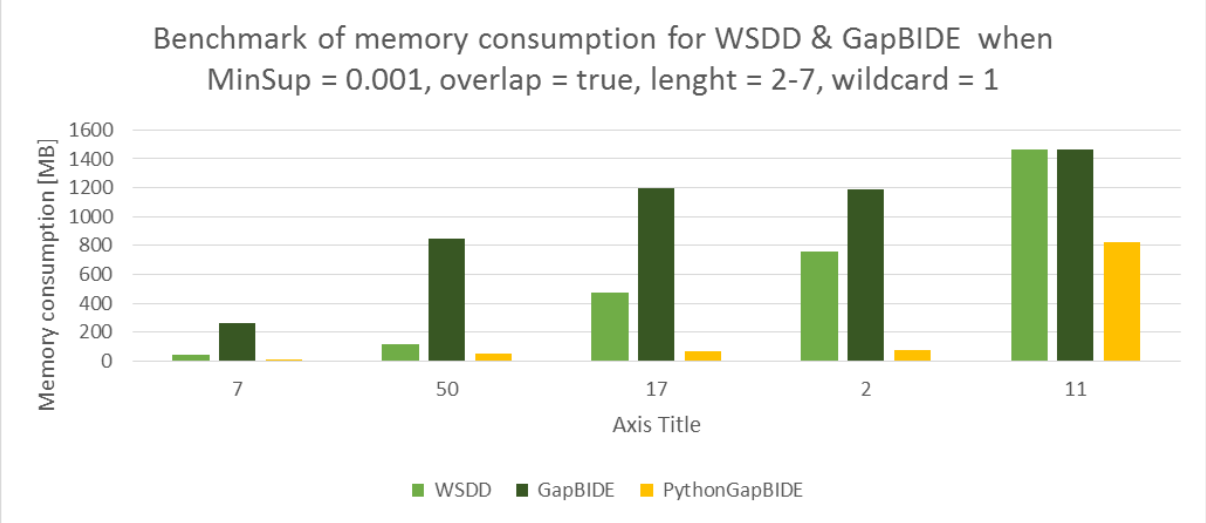


FIGURE 51: MEMORY BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=1

While all the previous benchmarks with a minimum support of 0.01 and non-overlapping patterns showed much better results of PythonGapBIDE, this is also true for most of the runs with MinSup=0.001 and overlapping patterns. However, when the number of events is really high like for example in smart home no. 11, PythonGapBIDE begins to struggle and uses disproportionately much more memory (even though the total is still less than what WSDD needs).

Out of curiosity the amount of events was further increased by the researcher to see if the disproportional rise of memory usage for PythonGapBIDE would continue (this was simulated by including the outliers of smart home no. 11 where click_type_id = '0'). The results for both run time as well as memory consumption for this ad hoc benchmark were recorded in the windows performance monitor and look side-by-side as follows:

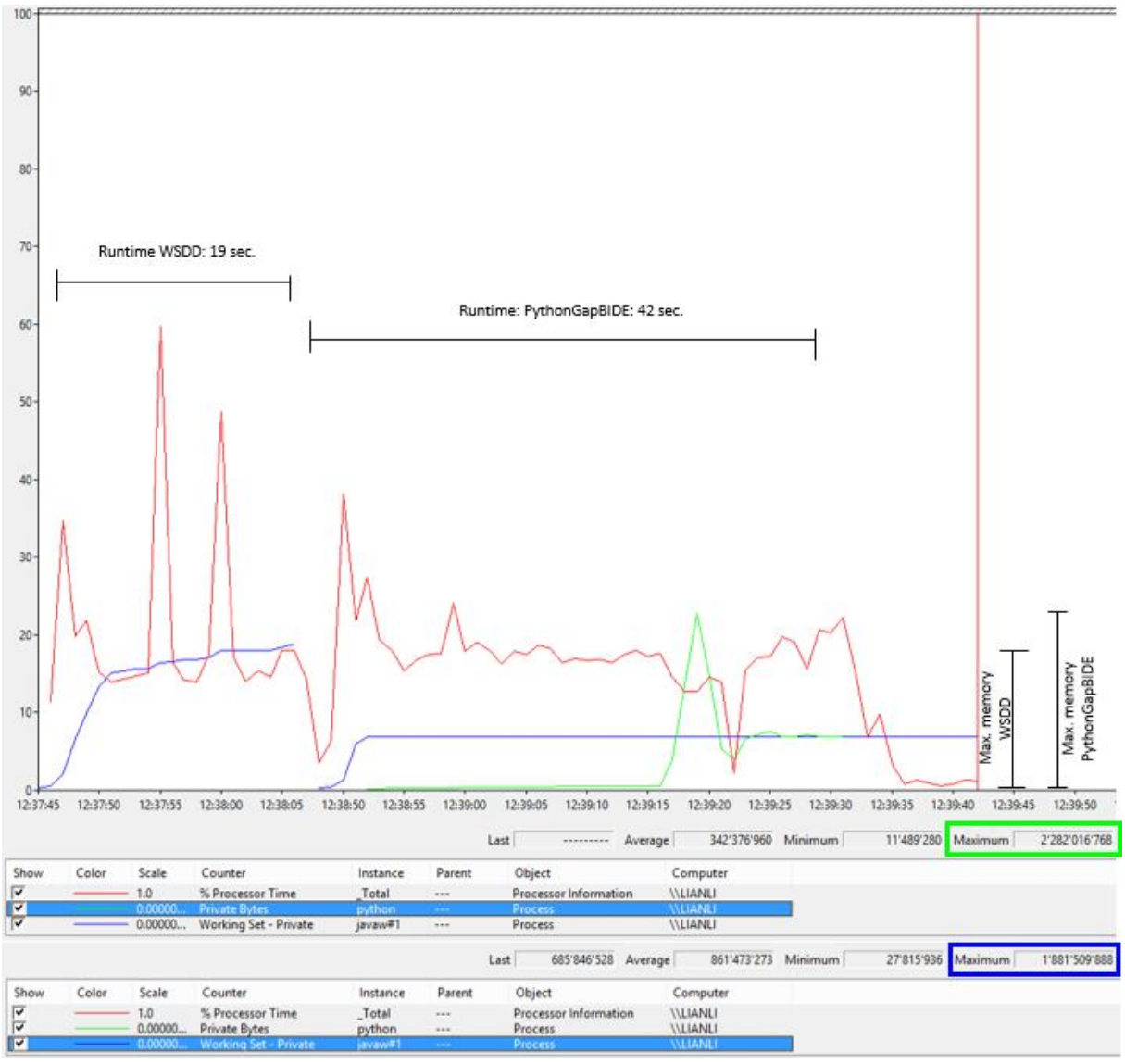


FIGURE 52: AD HOC BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=1

It can be seen that, while PythonGapBIDE showed very promising results regarding memory consumption in the previous benchmarks, when the number of events rises even more (to about 380'000) the memory consumption of PythonGapBIDE (2.28 GB) surpasses the one of WSDD (1.88 GB).

Additionally, the PythonGapBIDE's run time is roughly twice as long as WSDD's one (42 seconds for PythonGapBIDE versus 19 seconds for WSDD).

However, the much bigger disadvantage of GapBIDE (than the worse run times and the bad scaling regarding memory consumption when a lot of events are to be mined), is its previously mentioned inability to output the location where the wildcard was inserted in the pattern. For example it presents the patterns "680:6 -> 680:8", "680:6 -> 680:7 -> 680:8" and "680:6 -> 4691:87 -> 680:8" as "[Support Count Z], 680:6, 680:8" when the wildcarding of one event is allowed. WSDD on the other hand reports the patterns more intuitively as "[Support Count X], 680:6, 680:8" and "[Support Count Y], 680:6, *, 680:8".

The performance of the different algorithms and the associated problems put aside, it has to be said that wildcarding has not the potential attributed to it at the beginning of this research project because there are only few instances where wildcarding produces significantly different results from when wildcarding is deactivated. As a representative example the results from smart home no. 11 can be considered. The following table lists the top 20 patterns found in that smart home:

Support Count	Event 1	Event 2	Event 3	Event 4	Event 5
8706	752:18	752:17			
8495	752:17	752:18			
8141	752:18	*	752:18		
7913	752:18	752:17	752:18		
7807	752:17	*	752:17		
7787	752:17	752:18	752:17		
7488	752:18	*	752:18	752:17	
7335	752:18	752:17	*	752:17	
7324	752:17	752:18	*	752:18	
7320	752:18	752:17	752:18	752:17	
7169	752:17	*	752:17	752:18	
7153	752:17	752:18	752:17	752:18	
6905	752:18	752:17	752:18	*	752:18
6904	752:18	*	752:18	752:17	752:18
6806	752:17	752:18	*	752:18	752:17
6767	752:18	752:17	*	752:17	752:18
6755	752:18	752:17	752:18	752:17	752:18
6692	752:17	*	752:17	752:18	752:17
6689	752:17	752:18	752:17	*	752:17
6677	752:17	752:18	752:17	752:18	752:17

TABLE 43: THE TOP 20 WILDCARDED PATTERNS OF SMART HOME NO. 11

As can be seen, the increases of the support count of the wildcarded patterns over the non-wildcarded patterns is low. This is not only the case for smart home no. 11 but for all the other smart homes as well. It can therefore be said that wildcarding is not worth considering when mining frequent patterns in smart home event data since the increase in run times and memory consumption is not yielding significantly better results in the form of wildcarded patterns which have a much higher support than the non-wildcarded patterns.

If this is done, wildcarding disabled, the comparison regarding run times of WSDD versus PythonGapBIDE looks as follows:

	7	50	17	2	11
WSDD	0	1	1	1	2
PythonGapBIDE	3	3	4	4	14

TABLE 44: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=0

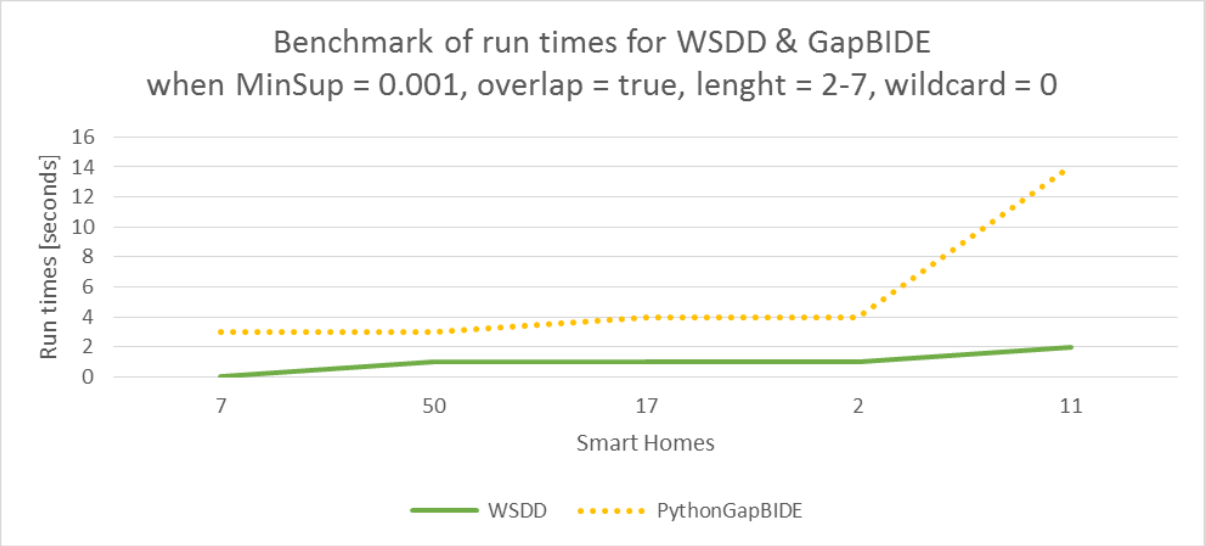


FIGURE 53: RUN TIME BENCHMARK FOR MINSUP=0.001, OVERLAP=TRUE, LENGTH=2-7, WILDCARD=0

The memory consumption for WSDD and PythonGapBIDE stays mostly the same as presented in Table 42 and Figure 51.

All in all it can be concluded that both benchmarked algorithms handle high amount of events well regarding run times but neither one of them shows very promising results regarding memory consumption. If this feature is sorely needed, despite the fact that the Aizo data set proved that only marginal different results are achieved when wildcarding is activated, and if memory consumption is an issue, the researcher proposes to look for further alternatives (i.e. either find ways to lower the memory consumption for WSDD or PythonGapBIDE or evaluate a completely different algorithm).

6.6 THE BENCHMARKS WHEN PERIODIC MINING IS ACTIVATED

The last series of benchmark wanted to compare the run times when periodic mining was activated for WSDD. Since none of the other algorithms allows periodic sequential pattern mining and the time frame of the master thesis did not allow to evaluate and implement a further generic algorithm able to mine periodic sequential patterns, only the impact that the activation of periodic mining has on WSDD was benchmarked.

First of all, the standard settings from the previous benchmarks was used again:

- A minimum support of 0.01
- Non-overlapping patterns with a minimal length of 2 and a maximal length of 7 events

For the five previously benchmarked smart homes WSDD was run with periodic mining deactivated on the one hand and with periodic mining activated, the grace period set to 10% and the minimum support for periodic patterns set to 90% on the other hand.

The achieved results with these settings look as follows:

	7	50	17	2	11
WSDD w/o periodic mining	0	1	1	1	1
WSDD with periodic mining	0	1	1	1	3

TABLE 45: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

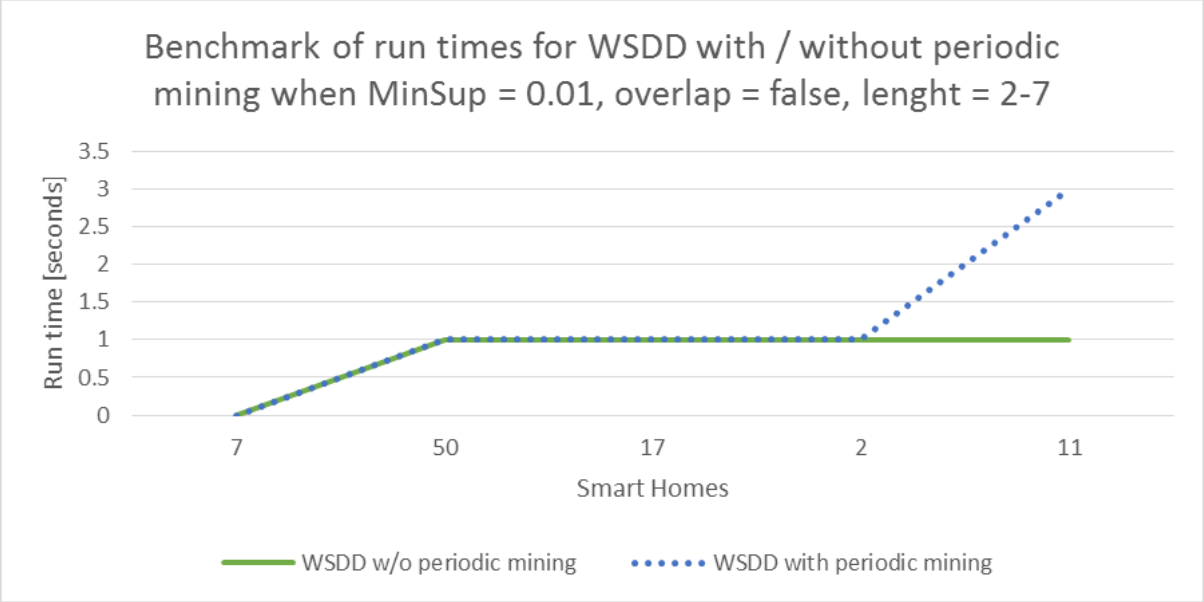


FIGURE 54: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

As can be seen, only for smart homes with a large amount of events (e.g. no. 11) the activation of periodic pattern mining is affecting the run time in any way. For a small or average number of events, the run times stay exactly the same. This is a little surprising since it was expected that at least for an average amount of events the additional mining of periodic pattern would impact the run times.

However, when the memory consumption is taken into account, the impact of periodic sequential pattern mining becomes obvious:

	7	50	17	2	11
WSDD w/o periodic mining	25.6	38.4	36.9	70.8	148.1
WSDD with periodic mining	29.4	135.1	135.1	137.4	1349.9

TABLE 46: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

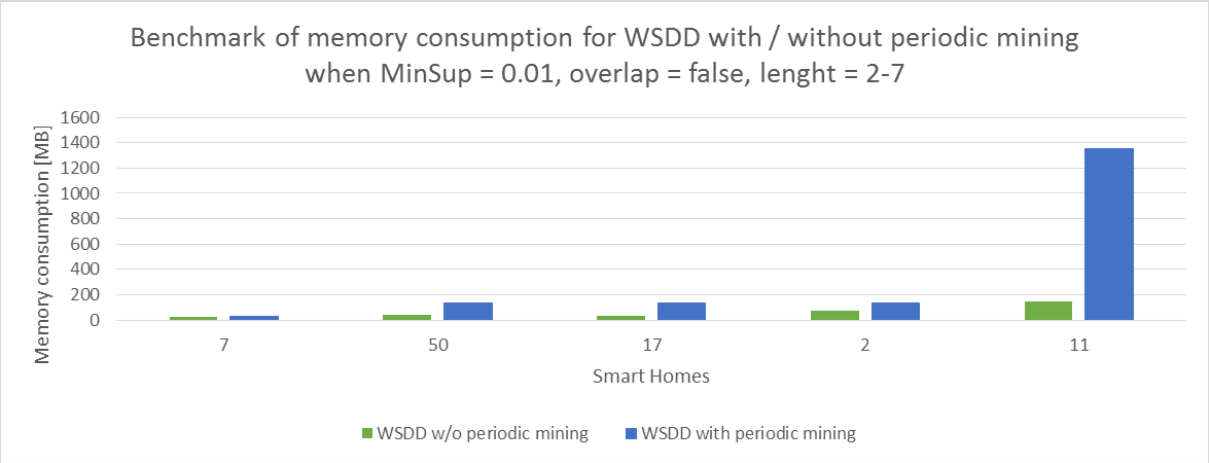


FIGURE 55: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=FALSE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

As can be seen, WSDD does not scale well when the amount of events is high. For the first time when doing benchmarks with WSDD, the memory consumption explodes.

Because of the bad results regarding memory consumption when large amounts of events are to be minded, for the next benchmark the amount of events was further increased by using overlapping instead of non-overlapping patterns to see if the high memory consumption could affect already average sized event amounts. The results for those benchmarks look as follows:

	7	50	17	2	11
WSDD w/o periodic mining	0	1	1	1	2
WSDD with periodic mining	0	1	1	1	10

TABLE 47: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

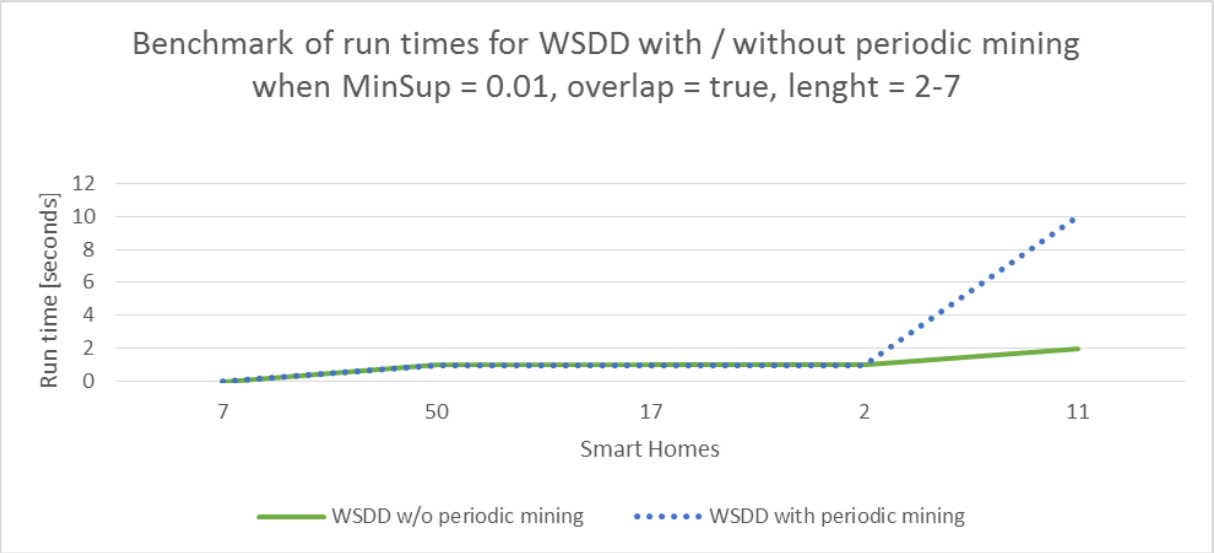


FIGURE 56: RUN TIME BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

As can be seen, the run times are not affected at all, expected for smart home no. 11, which represents smart homes with a high amount of events. But even for no. 11 the increase from two to ten seconds is not a dramatic one.

When regarding the memory consumption it becomes visible that the increase for overlapping event patterns is substantial for smart homes with an average amount of events (no. 50, 17 and 2) but does not increase much for smart home no. 11 compared to the results presented in Table 46:

	7	50	17	2	11
WSDD w/o periodic mining	30.7	67.1	128.1	45.8	494.2
WSDD with periodic mining	44.8	518.4	308.4	327.7	1477.6

TABLE 48: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

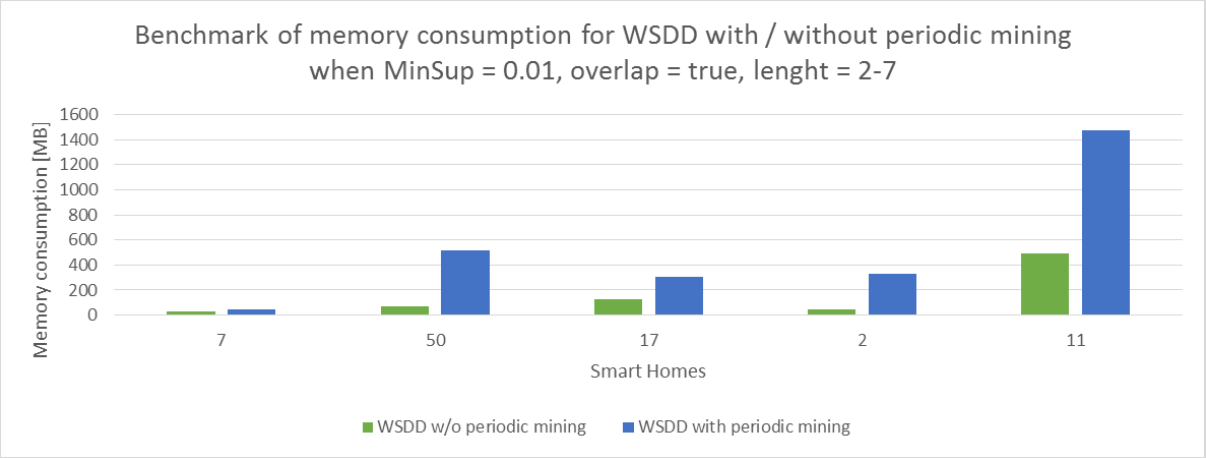


FIGURE 57: MEMORY BENCHMARK FOR MINSUP=0.01, OVERLAP=TRUE, LENGTH=2-7, PERIODIC MINSUP=90%, GRACE PERIOD=10%

Overall it can be said that even for the largest amounts of events the memory consumption stays well below 2 Gigabytes and that the run times are still very fast with a maximum of ten seconds for smart home no. 11.

When the found periodic sequential patterns are looked at, it becomes obvious that a certain aspect was not considered when designing the enhancement for WSDD for periodic sequential pattern mining. This aspect is the fact that certain patterns may be periodic but only present for a short period of time compared to the overall period of time of the events available in the database. This can be well shown with the following examples from smart home no. 50:

Support (in %)	Periodic Pattern (incl. interval and starting time)
94	3726:3064, 3726:3067, 3726:3064**@0**@01:04
94	3726:3064, 3726:3067**@0**@01:04
91	3726:3067, 3726:3064, 3726:3067, 3726:3064**@0**@01:05
91	3726:3067, 3726:3064, 3726:3067**@0**@01:05
90	3726:3064, 3726:3067, 3726:3064, 3726:3067, 3726:3064**@0**@01:05
90	3726:3064, 3726:3067, 3726:3064, 3726:3067**@0**@01:05

TABLE 49: THE PROBLEM OF PERIODIC PATTERNS WITH AN INTERVAL OF 0 MINUTES

As can be seen, all the found events fulfill a high support of 90% or more, this means at least 9 out of 10 events occur within a constant interval (+/- the grace period). However, the detected interval for every found pattern is 0 minutes (as represented by the first **@0** in the patterns). If the events of the periodic pattern 3726:3064, 3726:3067@0@01:04 are looked up in the database, it becomes obvious that the algorithm did everything right but that the events are just an abnormality:

1	start_time	event_sou	scene-/click-/s	event_id
5318	11.07.2012 01:04	3726	3064	441816
5319	11.07.2012 01:04	3726	3067	441817
5320	11.07.2012 01:04	3726	3064	441818
5321	11.07.2012 01:04	3726	3067	441819
5322	11.07.2012 01:04	3726	3064	441820
5323	11.07.2012 01:05	3726	3066	441821
5324	11.07.2012 01:05	3726	3064	441822
5325	11.07.2012 01:05	3726	3067	441823
5326	11.07.2012 01:05	3726	3064	441824
5327	11.07.2012 01:05	3726	3067	441825
5328	11.07.2012 01:05	3726	3064	441826
5329	11.07.2012 01:05	3726	3068	441827
5330	11.07.2012 01:05	3726	3068	441828
5331	11.07.2012 01:05	3726	3068	441829
5332	11.07.2012 01:05	3726	3068	441830
5333	11.07.2012 01:05	3726	3066	441831
5334	11.07.2012 01:05	3726	3064	441832
5335	11.07.2012 01:05	3726	3067	441833
5336	11.07.2012 01:05	3726	3064	441834
5337	11.07.2012 01:05	3726	3067	441835
5338	11.07.2012 01:05	3726	3064	441836
5339	11.07.2012 01:05	3726	3066	441837
5340	11.07.2012 01:05	3726	3064	441838
5341	11.07.2012 01:05	3726	3067	441839
5342	11.07.2012 01:05	3726	3066	441840
5343	11.07.2012 01:05	3726	3064	441841
5344	11.07.2012 01:05	3726	3067	441842
5345	11.07.2012 01:05	3726	3064	441843
5346	11.07.2012 01:05	3726	3067	441844
5347	11.07.2012 01:05	3726	3064	441845
5348	11.07.2012 01:05	3726	3067	441846
5349	11.07.2012 01:05	3726	3064	441847
5350	11.07.2012 01:05	3726	3066	441848
5351	11.07.2012 01:05	3726	3067	441849
5352	11.07.2012 01:05	3726	3064	441850
5353	11.07.2012 01:05	3726	3066	441851
5354	11.07.2012 01:05	3726	3064	441852
5355	11.07.2012 01:05	3726	3067	441853
5356	11.07.2012 01:05	3726	3064	441854
5357	11.07.2012 01:05	3726	3066	441855
5358	11.07.2012 01:05	3726	3067	441856
5359	11.07.2012 01:05	3726	3064	441857
5360	11.07.2012 01:05	3726	3067	441858
5361	11.07.2012 01:05	3726	3064	441859
5362	11.07.2012 01:05	3726	3067	441860
5363	11.07.2012 01:05	3726	3064	441861
5364	11.07.2012 01:05	3726	3067	441862
5365	11.07.2012 01:05	3726	3064	441863
5366	11.07.2012 01:05	3726	3067	441864
5367	11.07.2012 01:05	3726	3064	441865
5368	11.07.2012 01:05	3726	3067	441866
5369	11.07.2012 01:05	3726	3064	441867
5370	11.07.2012 01:05	3726	3067	441868
5371	11.07.2012 01:05	3726	3064	441869
5372	11.07.2012 01:05	3726	3067	441870
5373	11.07.2012 01:05	3726	3064	441871
5374	11.07.2012 01:05	3726	3066	441872
5375	11.07.2012 01:06	3726	3064	441873

FIGURE 58: THE PROBLEM OF PERIODIC PATTERNS WITH AN INTERVAL OF 0 MINUTES IN SH50

As can be seen, the events occurred all on the same day between 01:04 and 01:05 and have to be regarded as outliers (a possible explanation could be that they were wrongly reported to the database (i.e. multiple times)). All the other patterns with an interval of zero minutes were similar cases, for instance the events resulting in the periodic pattern 1280:4608, 1280:4608@0@16:18 found in smart home no. 17 look like this:

1	start_time	event_source_	scene-/click-/sensor-i	event_
29850	17.02.2013 16:14	1231	1228	2797111
29851	17.02.2013 16:14	1231	1187	2797112
29852	17.02.2013 16:18	1280	4608	2797113
29853	17.02.2013 16:18	1280	4608	2797114
29854	17.02.2013 16:18	1280	4608	2797115
29855	17.02.2013 16:19	1280	4608	2797116
29856	17.02.2013 16:19	1280	4608	2797117
29857	17.02.2013 16:19	1280	4608	2797118
29858	17.02.2013 16:20	1280	4608	2797119
29859	17.02.2013 16:20	1280	4608	2797120
29860	17.02.2013 16:20	1280	4608	2797121
29861	17.02.2013 16:20	1280	4608	2797122
29862	17.02.2013 16:20	1280	4608	2797123
29863	17.02.2013 16:20	1280	4608	2797124
29864	17.02.2013 16:33	1236	1198	2797125
29865	17.02.2013 16:33	1249	1199	2797126

FIGURE 59: THE PROBLEM OF PERIODIC PATTERNS WITH AN INTERVAL OF 0 MINUTES IN SH17

It can be concluded that all these periodic patterns with an interval of zero minutes found in the smart homes no. 11, 17, 50 and 7 are caused by outliers and are therefore useless.

An additional problem presented the fact that with the chosen parameter settings of 90% / 10% (which in theory seemed to the researcher an aggressive but realistic setting for finding only a few periodic patterns) no periodic patterns at all were found for smart homes no. 2.

Because of these two problems it can be concluded that the settings of 90% minimum support for periodic patterns and a grace period of only 10% are too aggressive. The next benchmarks were therefore run with less restrictive settings of 60% minimum support for periodic patterns and 15% of grace period. Additionally the events with an interval of zero minutes were manually filtered and the minimum pattern length reduced from 2 to 1 because for periodic pattern also single events can be of interest (e.g. every night at 22:00 the venetian blinds are closed). The achieved results were not better: although the outliers with an interval of 0 minutes can be excluded in that way, the problem is that only a few events remain, some of those with very large intervals of 14’040 to 70’500 minutes. Such long intervals of over one week are not very interesting when establishing a baseline of what constitutes normal behavior in a smart home.

A possible improvement could be achieved by the introduction of two additional user parameters: a minimum and maximum interval length (e.g. 60 minutes to 2880 minutes) and a minimum support *count* for periodic patterns (in addition to the minimum support), which would allow the specification of how

often the sequential pattern must occur overall to be mined for periodicity (e.g. a pattern that occurs only 20 times or less over all in the data set is not considered for periodic mining at all).

However, if these criteria were applied via a quick & dirty Java code change, for the smart homes no. 7. 50, 17, 2 and 11 only a handful of periodic event remained. At a first glance this could suggest that besides wildcards, which could not fulfill the potential attributed to them at the beginning of the project, also periodic sequential pattern mining is not interesting in the field of smart homes.

However, in the eyes of the researcher, periodic pattern mining is interesting also for smart homes and the most probably explanation for finding almost no periodic patterns with a support of over 60% within a grace period of up to 15% and an interval greater zero minutes, lies in the fact that the implemented way of detecting periodic patterns, which was inspired by some ideas presented by Rashidi and Cook (2009), works in theory but misses out on all the periodic patterns that are not exclusively periodic. If the following example (44, 1302:1357@1447@20:11) from smart home no. 18 is considered, it becomes clear what is meant:

Event ID	Start time	Source ID	Scene ID	Time difference in minutes	Within interval +/- 10% Grace?
2942254	05.03.2013 19:10	1302	1357		
2950302	06.03.2013 18:27	1302	1357	1397.066667	TRUE
2966226	07.03.2013 18:42	1302	1357	1455.55	TRUE
2966322	08.03.2013 18:30	1302	1357	1427.633333	TRUE
2966326	08.03.2013 18:31	1302	1357	1.416666663	FALSE
2966392	09.03.2013 18:50	1302	1357	1459.15	TRUE
2981662	10.03.2013 18:46	1302	1357	1435.516667	TRUE
2981710	10.03.2013 21:50	1302	1357	183.85	FALSE
2981752	11.03.2013 19:09	1302	1357	1279.733333	FALSE
3019852	12.03.2013 18:38	1302	1357	1408.5	TRUE
3019875	12.03.2013 22:08	1302	1357	210.3	FALSE
3019946	13.03.2013 19:18	1302	1357	1270.15	FALSE
3020018	14.03.2013 18:38	1302	1357	1399.75	TRUE
3020028	14.03.2013 19:30	1302	1357	51.76666667	FALSE
3020121	15.03.2013 19:08	1302	1357	1417.966667	TRUE
3020160	15.03.2013 20:39	1302	1357	90.88333333	FALSE
3020169	15.03.2013 20:54	1302	1357	14.95	FALSE
3020220	16.03.2013 04:18	1302	1357	444.1	FALSE
3020257	17.03.2013 23:07	1302	1357	2569.416667	FALSE
3020334	18.03.2013 18:57	1302	1357	1190.15	FALSE
3028193	19.03.2013 19:05	1302	1357	1447.916667	TRUE
3033014	20.03.2013 19:03	1302	1357	1437.9	TRUE
3068078	21.03.2013 19:04	1302	1357	1440.95	TRUE
3068129	21.03.2013 22:18	1302	1357	193.7833333	FALSE
3101818	22.03.2013 18:33	1302	1357	1214.7	FALSE
25					11
Fraction:					0.44
Fraction as integer (Support):					44

TABLE 50: EXAMPLE OF LOW PERIODIC SUPPORT FOR A NOT EXCLUSIVELY PERIODIC PATTERN

As can be seen from the column "Within interval +/- 10% Grace?", only 11 out of the sample of 25 events / 1-event "patterns" occur within the interval of 1447 minutes, which was mined by WSDD. These 11 periodically occurring events out of the total of 25 events result in the relative low periodic

support count of 44%. However, if one looks at the time period 6.3.2013 - 22.3.2013 listed above it becomes apparent that out of those 17 days the event 1302:1357 occurs on 15 days at around 20:11 (+/- 10%) and the pattern is in fact very periodic. The problem are the additional occurrences of the events outside of the interval, which reduce the support dramatically, making the pattern not periodic in the eyes of WSDD.

Additionally, the current implementation of WSDD cannot find patterns with more than one periodicity: for example if a pattern occurs every day at around 7:00 and the same pattern additionally occurs always at around 16:00, the algorithm does not delivery any useful results.

Because of these problems, it is proposed to investigate the issue of mining periodic sequential patterns with some generic periodic (sequential) pattern mining algorithm and adapt it to the smart home area. Additionally it could be worthwhile to adapt WSDD to whatever way is used by generic periodic (sequential) pattern mining algorithms to be able to do some benchmarks.

6.7 CONCLUSION OF CHAPTER 6

In conclusion it can be said that WSDD, despite being only an optimized brute-force approach, is an adequate choice for sequential pattern mining in the smart home area when memory consumption is no big issue (which is conceivable given today's situation where workstations have easily 8 GB of RAM and more and server memory ranges up to a few terabytes) and only fast run times are needed:

- WSDD paces or even outperforms the fastest generic sequential pattern mining algorithm which was benchmarked in this research project (PrefixSpan and the Python version of GapBIDE). Regarding run times, both variations of the BIDE algorithm implemented in Java are outperformed in all benchmarks by WSDD, sometimes by factors.
- Also regarding memory consumption WSDD is an interesting option: while the total amount of memory consumption can be quite high (1.5 GB to 1.9 GB for very high amounts of events), PrefixSpan and WSDD are on par for certain parameter settings and WSDD more often than not uses less memory compared to PrefixSpan. BIDE+ normally uses even more memory than PrefixSpan. Only PythonGapBIDE uses less memory than WSDD most of the times. However, all the benchmarked algorithms, including PythonGapBIDE, need large amounts of memory for large amounts of events and can therefore not be considered much better than WSDD.
- No preprocessing is needed for the input data. The continuous event data from the smart homes can be directly fed to WSDD as opposite to generic sequential pattern mining algorithms like BIDE+, PrefixSpan or GapBIDE, which need a sequence database as input.
- Even more important than the elimination of the preprocessing step is the always correctly reported support (count), for maximal as well as non-maximal patterns. BIDE+, PrefixSpan as well as GapBIDE only report the correct support for maximal patterns, which is a big handicap of those algorithms. Only through a workaround of using multiple runs for the same smart home the correct results for the support count of non-maximal patterns can be achieved (the workaround is to set MIN_LENGTH = MAX_LENGTH and repeat the mining MAX_LENGTH – MIN_LENGT + 1 times).
- While BIDE+, PrefixSpan and especially GapBIDE offer capabilities to mine wildcarded patterns, only WSDD is able to report where the wildcard was found in the pattern. This is an important aspect when comparing the baseline (in the form of the mined frequent sequential patterns) with currently happening events in a smart home. Additionally, with WSDD it is

possible to deactivate the wildcard feature (e.g. because it is not very interesting for the area of presence simulation), something which both BIDE+ as well as PrefixSpan are not able to do (however, the previously presented workaround can be used to disable wildcarding: set MIN_LENGTH = MAX_LENGTH and repeat the mining MAX_LENGTH – MIN_LENGT + 1 times). Only GapBIDE offers the same possibility as WSDD in this regard by default and without using any workarounds. This deactivation of wildcarding is also what is suggested by the researcher for the mining of sequential patterns in smart homes, since wildcarding does not present very different results (at least not for the data set available for this research project).

- WSDD offers the possibility to mine periodic patterns. While the achieved results are not (yet) perfect, with some additional effort (e.g. improve the algorithm so that it is also able to correctly mine patterns which occur very periodically at fixed intervals but additionally also at random times and are therefore currently not picked up by WSDD) an even bigger advantage can be gained compared to traditional algorithms like PrefixSpan, BIDE+ or GapBIDE.
- By changing a couple of lines of Java code, WSDD could be adapted to accept, instead of the minimum support count, a TOP n parameter to limit the number of patterns reported, which could be a more intelligent way for certain use cases. This would also eliminate the problem that too high a value for minimum support count results in no patterns being found.
- And last but not least is WSDD the easiest to understand and therefore also to implement algorithm, since it is only an optimized brute-force approach.

While WSDD can be recommended without reservations for application when memory consumption is irrelevant, the situation looks different if memory consumption is an issue. From a memory consumption point of view neither WSDD nor any of the other benchmarked algorithms could show good results for all the different combinations of parameter settings and smart homes. Therefore it is proposed to investigate the following possible improvements regarding memory consumption after this research project is finished:

1. proposition to save memory: use integers instead of strings for the representation of events.
   In a short trial, some quick & dirty Java code modifications and an extension of the database with a translation table for all the events of smart home no. 11, showed promising results regarding memory consumption. The following input parameters were used for this ad hoc benchmark:

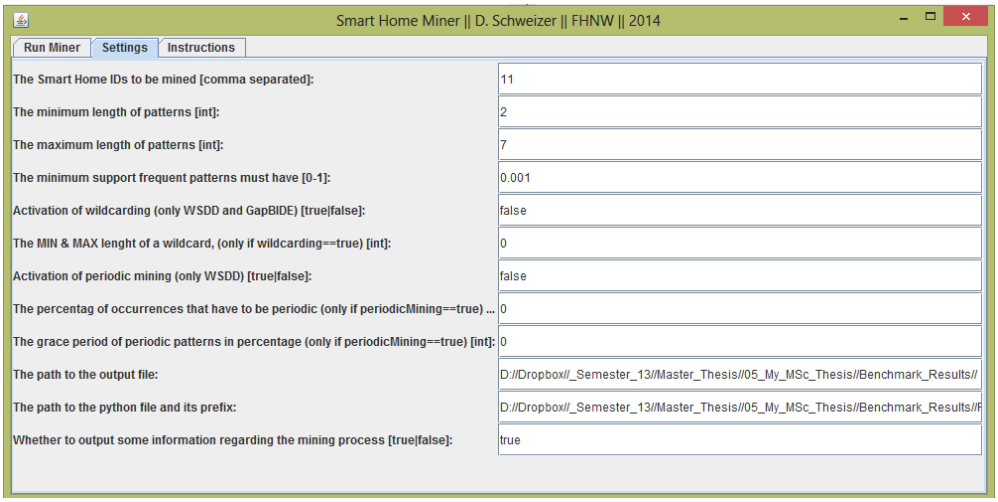


FIGURE 60: AD HOC MEMORY CONSUMPTION BENCHMARK WITH A TRANSLATION TABLE

With those settings WSDD mined the overlapping patterns of smart home no. 11 with a length of 2 to 7 events represented as integers in about 1 second (which is on par with the run times when strings are used for the event representation). The maximum memory consumption in Megabytes dropped from 501.9 to 175.2, which equals a reduction of memory consumption of 65%.

However, to fully benefit from using integers instead of strings and to possibly achieve even bigger memory savings, the whole WSDD source code would have to be rewritten to accommodate this change and the time frame of this research project did not allow that.

2. proposition: enhance the brute-force algorithm WSDD with some intelligence.
   Along the line of how Apriori and BIDE+ work, WSDD could initially only mine 2-events patterns for their support count. Next, it would prune all the patterns which do not make the cut (support count < minimum support count). Only then WSDD would continue to mine longer patterns of three and more events. However, the algorithm would only store those 3+-events patterns in the HashMap if they contain one of the frequent 2-events patterns as a basis. This would reduce the size of the HashMap considerably by employing the Apriori principle which states that a super-pattern (e.g. a 5-events pattern) can never be more frequent than a sub-pattern (2-events pattern).
3. proposition: shorten the mining timeframe.
   Currently WSDD (and all the other algorithms against which it was benchmarked) used the whole range / timeframe of available events, which consists of up to eight months. It would be possible to mine only shorter timeframes (e.g. only the most recent two months instead of eight months). This would result in fewer patterns and therefore less memory consumption. However, to achieve the same quality for the found patterns, it would be necessary to combine the newest results with ones from previous mining sessions. An even better quality could be achieved by employing something like a decay-factor for each pattern, which would allow the deletion of old and outdated frequent patterns from the repository of all known frequent patterns.
4. proposition: the two HashMaps dedupedPatterns und periodicPatterns could by combined.
   For the prototypes developed for this research project, two independent HashMaps were used to store the frequent and the periodic patterns. Since they contain redundant information, a combination of those two HashMaps would also result in a lower memory consumption if periodic mining is activated: it should be possible to reduce the memory consumption by approximately the amount of memory used by the runs without periodic mining.
5. proposition: evaluate other, more memory-considerate data types in Java than HashMaps.
   While the HashMaps proved to be quite fast, it is conceivable that there exist other data types, e.g. Trove Collections (source: http://trove.starlight-systems.com/), which use less memory.
6. proposition: evaluate other programming languages than Java.
   Check if a WSDD implementation in other programming languages (e.g. PERL) could be less memory intensive than in Java.

Finally the researcher proposes to investigate the potential of periodic patterns in more detail, since the simple approach which is currently employed by WSDD yields nearly no useable periodic patterns. It is suggested to look for and evaluate some generic algorithm(s) able to mine periodic (sequential) patterns and adapt it/them to the smart home conditions (e.g. that there is only one continuous sequence of events).

7 CONCLUSION

7.1 INTRODUCTION TO CHAPTER 7

In this final chapter the main findings are summarized as the project's contribution to the body of scientific knowledge and an outlook for possible future research projects, building on the findings of this project, is presented. Additionally, a summary of what was done for this project is given.

7.2 CONTRIBUTION TO THE BODY OF SCIENTIFIC KNOWLEDGE

The following subchapters contain the different contributions of this research project to the body of scientific knowledge.

7.2.1 THE OPTIMAL SET OF PARAMETERS FOR SEQUENTIAL PATTERN MINING IN SMART HOMES

The first contribution to the body of scientific knowledge consists of the parameter settings which proved to be the most useful for the data available and the recommendation of the fastest algorithm found for frequent sequent pattern mining.

Since the data available for this research project are real-life data collected by Aizo, it is assumed that the following parameter settings hold for smart home event data in general:

- Pattern length of 2-5 events:
  Frequent sequential patterns consisting only of one event are only interesting for periodic mining but not for frequent sequential pattern mining and the minimum length is therefore proposed to be set to 2. Patterns with more than 5 events proved to be in all but one instance repetitions of shorter 2-, 3- or 4-events patterns, therefore 5 is proposed as the maximum length.
- Minimum support of 0.01 – 0.001:
  Higher minimum support values than 0.01 produce almost no frequent patterns for certain smart homes while lower values than 0.001 produce a lot of uninteresting patterns that occur too few times. The following numbers of overlapping 2- to 5-events patterns can be mined from the Aizo dataset with three different minimum support values:

Smart Home	Minimum support		
	0.01	0.005	0.001
2	5	23	240
3	13	38	343
5	25	49	328
6	33	78	506
7	79	190	1841
9	26	75	509
10	15	43	347
11	23	47	209
17	10	28	226
18	17	28	264

20	15	33	251
23	16	78	374
24	6	18	255
26	49	106	607
28	16	22	135
30	20	29	168
34	26	101	2714
39	20	60	460
40	13	27	361
50	30	78	352
54	6	20	221
55	20	32	333
56	12	39	408
94	18	45	595

TABLE 51: THE NUMBER OF OVERLAPPING 2- TO 5-EVENTS PATTERNS FOUND FOR DIFFERENT MINIMUM SUPPORTS

- Overlapping patterns:
  Overlapping patterns are recommended because certain patterns are only found by mining overlapping patterns and are missed when mining non-overlapping patterns.
- Wildcarding should be deactivated:
  Since wildcarding does not generate significantly different results but increases both run times as well as memory consumption, it is in most cases not worth to activate it.
- Periodic mining should be deactivated:
  Since almost no periodic patterns can be found by WSDD with at least moderately strict parameter values for minimum support and grace period, it is suggested to disable it. However, it is strongly recommended to adapt the way periodic patterns are mined and then periodic mining should be re-evaluated.

An additional benchmark for these "sweet spot" settings mentioned above shows the following adjusted results in seconds (WSDD was run only once, the other three algorithms four times each, for the values MIN_LENGTH / MAX_LENGTH = {2,3,4,5}, and the run times cumulated to assure a correct support count):

	7	**34**	**94**	**3**	**9**	**6**	**26**	**56**	**20**	**24**	**50**	**40**	**39**	**2**	**55**	**17**	**28**	**30**	**18**	**5**	**54**	**10**	**11**	**23**
WSDD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PrefixSpan	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	3	5	5	10	5
BIDE+	1	0	0	0	2	2	2	2	2	2	5	4	6	6	7	4	23	6	25	25	12	21	43	35
PythonGapBIDE	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	9	5	10	10	9	10	17	16

TABLE 52: ADJUSTED RUN TIME BENCHMARK FOR MINSUP=0.005, OVERLAP=TRUE, LENGTH=2-5

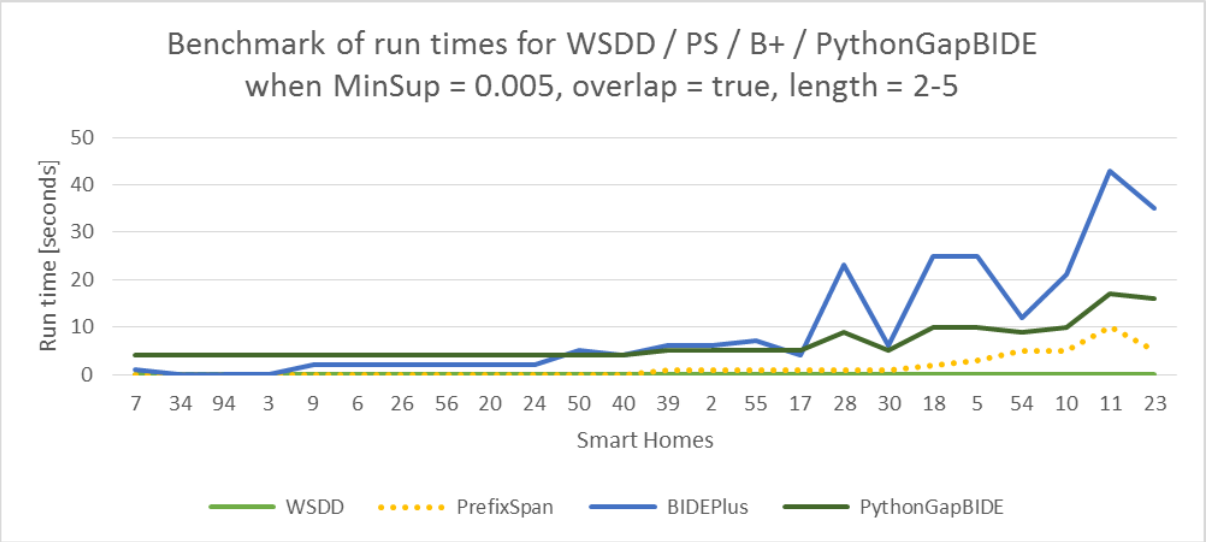


FIGURE 61: ADJUSTED RUN TIME BENCHMARK FOR MINSUP=0.005, OVERLAP=TRUE, LENGTH=2-5

As can be seen, WSDD is the best choice regarding run times with those settings. This corresponds with all the other benchmarks conducted for this research project and from a run time point of view it can therefore be concluded that using elaborate algorithms like PrefixSpan or GapBIDE is not only an overkill regarding complexity but also slower than using an optimized brute-force approach in the form of WSDD.

7.2.2 THE PROOF OF THE THESIS STATEMENT

The second contribution is the (dis)prove of the thesis statement. For repetition, the thesis statement of this thesis is called:

> It is possible to learn, in an efficient way, frequent and periodic usage patterns and preferences of smart home inhabitants only from event data available within a smart home.

The part of this statement that was proven correct is the one about learning frequent usage patterns of smart home inhabitants in an efficient way, *if* one is looking only at the run times and memory consumption is a secondary concern. This part can be fulfilled especially well by WSDD, which finds frequent sequential patterns very fast (e.g. for all the real-life smart home event data available for this project the brute-force approach needs less than a second per smart home to find between 18 and 190 frequent overlapping patterns of 2 to 5 events each). Memory consumption could be a secondary concern if e.g. Aizo does the mining on a central dedicated server infrastructure outside of the smart home and offers the mined patterns as a service to the smart home inhabitants.

However, if efficiency means efficient memory usage (i.e. low memory consumption), none of the benchmarked algorithms showed good results for all the different benchmarks conducted for this research project: the Python version of GapBIDE showed the most promising results overall, with the exception that WSDD scales better for very large amount of events. Chapter 6.7 includes different propositions which could help to reduce the memory consumption of the different algorithms, especially WSDD, and the researcher proposes to investigate those possibilities should memory consumption be

an issue (e.g. if it is planned to do the mining inside smart home components with limited memory available).

One part of the statement which could not be conclusively proven or disproven as part of the research project is the aspect of mining periodic sequential patterns. While a few periodic patterns could be found in the initial manual data analysis done for this research project and while in theory an interesting aspect, the implemented enhancement to WSDD for mining periodic pattern does not work as planned and only a hand full of periodic patterns was detected by the algorithm in all the events of all the different smart homes. The researcher therefore proposes to re-investigate the aspect of periodic sequential mining in a follow-up research project (see also further elaborations in chapter 6.6 regarding the problems encountered during this project).

7.2.3 THE ANSWERS TO THE RESEARCH QUESTIONS

The third contribution comes in the form of the answers to the research questions. Those are divided into separate subchapters per research question.

7.2.3.1 WHAT EXACTLY IS A PATTERN AND WHAT KIND OF PATTERNS EXIST INSIDE THE DATA?

The data available for this research project come from Aizo, a Swiss company producing smart home components. They collect the data of 24 different private smart homes as part of a pilot project and a time period of eight months was provided for this research project.

Inside those data sequential event patterns can be found. A sequential pattern is for example the turning on of a lamp, then opening the venetian blinds before the same lamp is extinguished. This would look something like "100:200 -> 200:300 -> 100:400" in the database from Aizo, where the first number is the ID where the event was triggered (e.g. the light switch in the bed room) and the second number, delimited by a ':', is the state (e.g. on or off). Additionally a timestamp is recorded with each event. A real-life example from smart home no. 2 is the following 3-events pattern: "78:7 -> 78:9 -> 105:8", which translates to "Kitchen lighting switch:Mood1 -> Kitchen lighting switch:Off -> Ground floor bath room lighting switch:Mood1".

Additional patterns can be found in the power consumption curves of the smart homes (see chapters 4.3 and 0). However, this research project did not investigate those patterns in depth.

7.2.3.2 WHAT REGULARITIES DO THE PATTERNS HAVE?

Two different regularities could be found inside the smart home date:

- frequent sequential patterns and
- periodic sequential patterns or events.

While the project achieved to mine frequent patterns with all four algorithms (WSDD, PrefixSpan, BIDE+ and GapBIDE), only WSDD would in theory be able to mine periodic patterns. However, due to design-time weaknesses of the enhancements for WSDD regarding periodic pattern mining, the achieved results are not satisfactory and it is proposed to further investigate this topic (see chapter 6.6).

7.2.3.3 HOW CAN THE DATA BE ENCODED FOR THE DIFFERENT ALGORITHMS?

For WSDD, which was explicitly developed for mining frequent sequential patterns in smart home data, the input can be one continuous list of all events, ordered chronologically. Such a list can be gained with a simple SQL query from the relational database storing the smart home information.

For the well-known frequent sequential pattern mining algorithms PrefixSpan and BIDE+ as well as for GapBIDE, the data needs to be encoded as a sequence database. This means that data preprocessing is needed: the one continuous sequence of events has to be divided into n sequences with a length of the maximal number of events that a patterns shall have. Because of that division, wrong support counts are calculated (see chapter 5.3 and especially Table 17). Therefore a workaround was designed by this research project: run PrefixSpan, BIDE+ or GapBIDE multiple times with different input sets (e.g. if patterns of two to five events are wanted, run the algorithm once with a sequence database with all the 2-events patterns, then with all the 3-events patterns, next with all 4-even patterns and finally with all 5-events patterns), via post-processing only maximum-length pattern are reported for each run and finally all the maximum-length patterns are combined. While this workaround works fine and reports the correct support counts, it has the disadvantage that it increases the run times.

7.2.3.4 WHAT ARE SUITABLE METHODS AND ALGORITHMS TO FIND PATTERNS IN THE SMART HOME AREA?

As was shown in chapter 6, all the four algorithms presented in this paper are able to find frequent sequential patterns. While all of them in general are suitable, the run times of the different algorithms vary greatly: BIDE+ needs the longest while both PrefixSpan and GapBIDE (at least its Python implementation) show significantly faster run times. However, even faster run times are achieved with WSDD and it is therefore suggested to use this simple but optimized brute-force method for mining frequent sequential patterns in smart home event data.

For finding periodic sequential patterns, of the four benchmarked algorithms only WSDD is in theory suitable. However, the achieved results were disappointing due to design time weaknesses and it is therefore suggested to spend some more effort to optimize WSDD's extension for periodic sequential pattern mining and possibly to compare it to a generic state-of-the-art periodic (sequential) pattern / event mining algorithms (see also chapter 6.6 for details and suggestions).

Additionally it is proposed to investigate methods to reduce the memory consumption for WSDD (or GapBIDE / PrefixSpan if one of those algorithms is chosen despite the slower run times), because all three algorithms use considerable amounts of memory (up to roughly 2 GB for smart homes with a lot of events). For suggestions of how to reduce memory consumptions, see chapter 6.7.

7.2.4 USAGE EXAMPLES OF VARIOUS FOUND PATTERNS

The last contribution are use cases for some of the found patterns. Exemplary, the patterns found for smart home no. 50 when using the sweet spot settings proposed in chapter 7.2.1 are used in the following use cases.

- Use case one: save energy by shutting down unneeded appliances.
  The following pattern was found a total of 603 times:

Event no.	Source ID	Scene ID	Source clear-text	Scene clear-text
1	3187	3006	Switch_W	Mood1
2	3181	3048	Ceiling light	Mood1
3	3181	3034	Ceiling light	Off
4	3187	3010	Switch_W	Off

TABLE 53: SAMPLE PATTERN FOR ENERGY SAVING

If the smart home would detect that the first three events were executed, but not the last one (event no. 4, the switching off of Switch_W) *and* if the smart home detects that a specifiable amount of other patterns were executed since the event no. 3 was executed, it could execute event no. 4 autonomously to save electricity in the assumption that the smart home inhabitant forgot to execute this forth event.

- Use case two: increase comfort of smart home inhabitants through automation.
  The following pattern was found a total of 282 times:

Event no.	Source	Scene	Source clear-text	Scene clear-text
1	3181	3048	Ceiling light	Mood1
2	3181	3034	Ceiling light	Off
3	3187	3006	Switch_W	Mood1

TABLE 54: SAMPLE PATTERN FOR COMFORT INCREASE

If the smart home would detect that the first two events (3181:3048 and 3181:3034) are the last two events executed in the smart home, it could either ask the smart home inhabitant if it now should activate Mood1 for Switch_W or autonomously execute this third event.
Additionally, the smart home could ask the inhabitant if he/she always wants to execute events no. 2 and 3 simultaneously in the future if the previous event was the activation of Mood1 of ceiling light without being asked about it. If the inhabitant answers the question with yes, the event no. 2 (the switching off of ceiling light) would be reprogrammed to also activate Mood1 for Switch_W (event no. 3). If the inhabitant answers with no, an additional question could be asked to the inhabitant: shall the smart home never ask about this pattern / automation in the future? If this second question is answered with yes, the pattern would be put on a permanent black list and ignored from there on.

- Use case three: presence simulation.
  If activated, a presence simulator could simulate the presence of smart home inhabitants even though they are away (e.g. on holidays or out on a Saturday evening) to prevent burglars from breaking in. This could be achieved by randomly executing the top n (e.g. 10) frequent events which involve only lightning activities during the night and only shading activities during the day (because it would be useless to execute e.g. access, climate or security events since they cannot be perceived from outside the building). The top 10 patterns for smart home no. 50 consisting exclusively of lighting events are:

Support count	Event 1	Event 2	Event 3
2887	3181:3048	3181:3034	
1072	3181:3034	3187:3010	
982	3187:3006	3181:3048	
946	3181:3034	3181:3048	
938	3187:3006	3187:3010	
885	3181:3048	3181:3034	3187:3010
871	3187:3010	3187:3006	
842	3187:3006	3181:3048	3181:3034
781	3181:3048	3181:3034	3181:3048
747	3181:3034	3181:3048	3181:3034

TABLE 55: SAMPLE PATTERNS FOR PRESENCE SIMULATION

These events could by executed randomly and at random times during predefined time frames specified by the smart home inhabitant(s) in the evening and early morning, which could be for example 18:00 – 23:00 and 06:00 – 08:00. Analogously the shading events could be simulated during a time frame at daytime.

It can be argued that for presence simulation it is enough to mine 1-event "patterns" since the execution is random anyways and it would therefore be enough to just execute the top n events (= top n 1-event "patterns") in a random order. However, the execution of 2- to 5-events patterns could possibly create a more realistic illusion of presence than the execution of random single events.

- Use case four: notify nurses of an assisted living home or relatives if none of the frequent patterns / events are detected.
  If none of the top n patterns as presented in Table 55 are being executed in a user-specified period of time and time-frame (e.g. for the last 4 hours between 6:00 and 22:00), then an alarm is triggered and a nurse of the assisted living home or a relative of the inhabitant is notified.
  In addition to the top n 2- to 5-events pattern, it could also be useful to mine the top m 1-event "patterns" since it would be enough to detect that none of those frequent events (= 1-event "patterns") happened. For example the top 5 events for smart home no. 50 are:

Support count	Source	Scene	Source clear-text	Scene clear-text
3584	3181	3034	Ceiling light	Off
3583	3181	3048	Ceiling light	Mood1
2453	3187	3010	Switch_W	Off
2202	3187	3006	Switch_W	Mood1
638	3190	3006	Switch_B	Mood1

TABLE 56: SAMPLE PATTERNS FOR NURSE / RELATIVE NOTIFICATION

However, it has to be noted that the mining of periodic patterns could by a big benefit for this use case and it is therefore suggested to look into that topic since the current version of WSDD delivers insufficient results (see chapter 6.6 for details). To a lesser degree this is also true for

all the three previously presented use cases which could profit from knowledge in the form of periodic sequential patterns and / or events.

The following two SQL queries (and adaptations thereof) were necessary for translating the found source / scene IDs to clear-text names used in this chapter:

```
SELECT event_sources.event_source_id, name, inst_id
FROM event_sources, devices
WHERE event_sources.event_source_id IN ('3187', '3181')
AND event_sources.event_source_id = devices.event_source_id

SELECT scene_id, name
FROM scenes
WHERE scene_id IN ( '3006', '3048', '3034', '3010')

-- Frequent sequential pattern: 3187:3006, 3181:3048, 3181:3034, 3187:3010
```

The German names of the event sources and scenes where translated to English by the researcher.

7.3 SUMMARY OF THIS RESEARCH PROJECT

Since this is the last chapter of the research project at hand, a summary of the goal of this project as well as the steps undertaken to reach it is due. Those main steps of this master thesis were:

- The assignment of this master thesis was to find an efficient way to mine patterns in smart home data to establish a baseline for what constitutes normal behavior of smart home inhabitants. Such a baseline in the form of patterns is needed for a smart home to become more intelligent than today's smart homes are. If a future smart home knows what its inhabitants normally do and prefer, it can deduce and in the end autonomously apply rules that go along with the preferences.
- For the purpose of mining patterns in smart home data, access to the power consumption and event data from a pilot project by Swiss company Aizo, which produces the digitalSTROM components for smart homes, was negotiated. Via a non-disclosure agreement this research project got access to eight month worth of data for 24 private smart homes.
- After an initial manual analysis of the available data and a literature review regarding data mining, especially pattern mining in smart homes, it was decided to focus on mining frequent and periodic sequential patterns in the event data.
- The concrete question this research project is trying to answer therefore goes as follows: is it possible to learn frequent and periodic usage patterns (which inherently contain the preferences) of smart home inhabitants only from event data available within a smart home in an efficient way?
- After an in-depth manual data analysis confirmed that patterns are available in the events recorded in the Aizo data, a first brute-force algorithm was designed and implemented in Java. Since the performance of this algorithm was bad and the found frequent patterns were reported multiple times, different optimizations took place, which resulted in the WSDD algorithm.
- To benchmark the performance of WSDD, an interface to the open-source Java framework SPMF (Sequential Pattern Mining Framework) by Fournier-Viger, et al. (2013), which implements the well-known PrefixSpan (Pei, et al., 2004) as well as the BIDE+ (Wang & Han, 2004) algorithm, was written.

- Additionally, the GapBIDE algorithm (Li, et al., 2012) was implemented in Java by the researcher to be able to run additional benchmarks where the wildcarding of patterns can be controlled (something both BIDE+ as well as PrefixSpan lack). Besides the implementation of GapBIDE, WSDD was enhanced to be able to mine wildcarded patterns, too.
- Since PrefixSpan, BIDE+ and GapBIDE all need a sequence database as input instead of the one continuous sequence of events as available in a smart home, a way of transforming the one big continuous sequence into n smaller (sub-)sequences was needed. For that purpose the approach suggested by Chikhaoui, et al. (2010) was used, which consists of a fixed-sized window that is slid over the big sequence to generate overlapping or non-overlapping smaller sequences.
- Next a further enhancement of WSDD was implemented to enable it for periodic sequential pattern mining since the initial data analysis showed some periodic patterns / events in the smart home events.
- After all this development was done, extensive benchmarking began. During the benchmarking, additional enhancements had to be implemented to deal with certain shortcomings. This included the introduction of a minimum length parameter as well as a post-processing step for PrefixSpan, BIDE+ and GapBIDE to eliminate the shortcoming of wrongly reported support counts for non-maximal patterns caused by employing the proposition of Chikhaoui, et al. (2010). Another adjustment was the introduction of the external call to the original prototype of GapBIDE written in Python by Li, et al. (2012) because the Java version developed for this project showed shortcomings when dealing with a lot of events.
- After all the algorithms were finally reporting the same patterns for the same input parameters, the following results could be found:
    - It is overkill to employ sophisticated algorithms like PrefixSpan, BIDE+ or GapBIDE for mining frequent sequential patterns: not only are they more complex to understand and implement, they are also often slower than the optimized brute-force algorithm WSDD. Additionally WSDD is more flexible (e.g. the results could be easily filtered according to a top n parameter instead of a minimum support count without increasing run times or memory consumption).
    - Also from a memory consumption point of view WSDD is a viable option since PrefixSpan and BIDE+ normally use more memory than does WSDD. Only the Python version of GapBIDE could often outperform WSDD in this regard. However, also GapBIDE shows bad results regarding memory consumption if the number of events is very high. All in all it can therefore be said that none of the benchmarked algorithms shows good results for all different sets of input parameters. Therefore the researcher proposes various options to improve the memory consumption, which can be investigated in a follow-up research project if it is decided that memory consumptions of up to about 2GB are an issue which has to be remedied.
    - The wildcarding feature is only adequately fulfilled by WSDD since GapBIDE does not show (in the patterns output by the algorithm) where a wildcard was found. This leads to the problem that a pattern without any wildcarded events in it counts towards the same support count as a pattern with wildcards in it (e.g. “100:200 -> 300:400” and “100:200 -> 700:800 -> 300:400” count towards the support count of “100:200 -> 300:400” if one event is allowed to be a wildcard). This is not what is needed for smart

home automation. Regardless of that, the potential of wildcarding proved to be not as big as anticipated because in the Aizo data set only very small differences in the patterns found could be observed if wildcarding was activated. It is therefore suggested to also deactivate wildcarding for WSDD, which would report the position of a detected wildcard correctly (i.e. the aforementioned pattern "100:200 -> 300:400" counts only towards the support count of "100:200 -> 300:400", while "100:200 -> 700:800 -> 300:400" counts towards the separate support count of pattern "100:200 -> * -> 300:400").

- The goal of mining periodic sequential patterns was not reached. Because of design time weaknesses of how periodic patterns are detected, only with very loose parameter settings (which produce too unreliable results) could any patterns be detected. It is therefore recommended to further research the issue of periodic sequential pattern mining, e.g. by adapting a generic periodic (sequential) pattern mining algorithm to deal with smart home event data and also enhance WSDD to better detect periodic patterns before benchmarking the algorithms.
- The last step of this research project was writing down the insights gained as the contribution of this research project towards the scientific body of knowledge. With the knowledge gained in this project (and possibly after optimizing the memory consumption as well as periodic sequential pattern mining), follow-up research projects can now deal with the questions of how to best derive rules from the patterns, identify triggers, predict the next event(s) and finally how the improvements and positive effects like energy savings, comfort increases or better protection against burglars can be realized as part of a multi-agent-system smart home.

Another way to summarize what was done for this research project can be had by looking at the approach of Fayyad, et al. (1996), which describes how one can come from data to knowledge:

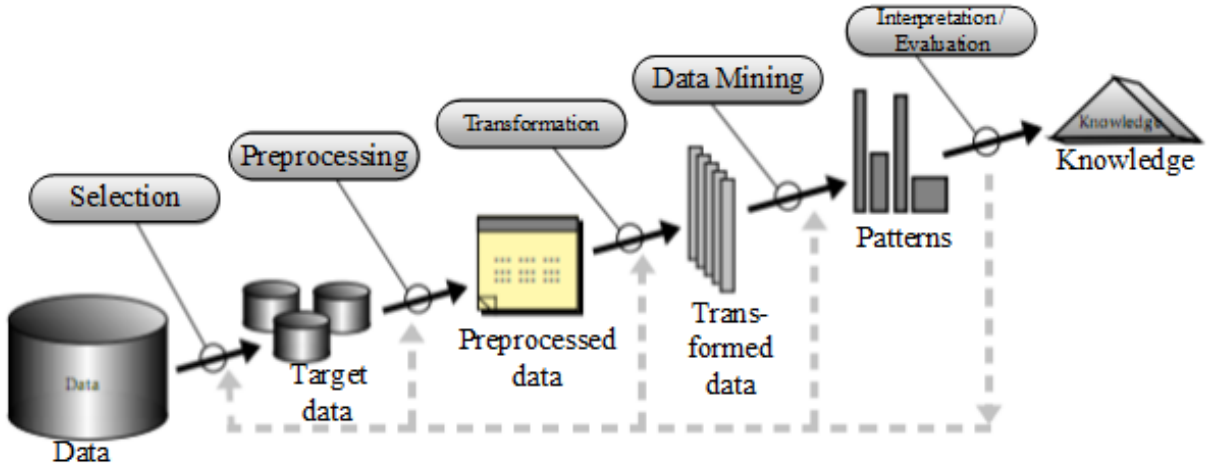


FIGURE 62: HOW TO COME FROM DATA TO KNOWLEDGE ACCORDING TO FAYYAD, ET AL. (1996)

The data provided by Aizo in the form of a relational database was the starting point for the following activities of this research project:

- Selection: the events table as well as app_button_clicks and sensor_events were identified as containing the event data, which this project was interested in.

- Preprocessing: the one continuous sequence of events per smart home was divided into n sequences of length m, which make up the sequence database for PrefixSpan, BIDE+ and GapBIDE.
- Transformation: currently no transformation is used. This would for example be needed if the algorithm translates the input data from strings to integers to reduce the memory consumption.
- Data mining: the learning of frequent sequential or periodic sequential patterns by employing the WSDD, PrefixSpan, BIDE+ or GapBIDE algorithm.
- Interpretation (what was done in the chapters benchmarking and with defining the use cases in the conclusion): by analyzing the found patterns, necessary adaptions in the data mining process could be made and knowledge could be derived.

This way of looking at things emphasizes the data mining character that this research project had.

7.4 CONCLUSION OF CHAPTER 7

The final chapter of this research project is mainly concerned with presenting and summarizing the findings of the master thesis and bringing the project to an end.

Overall it can be said that the generic goals of this research project according to Saunders, et al, (2009) are reached:

- ✓ Formulate and clarify your research topic.
- ✓ Critically review the literature.
- ✓ Understand your (research) philosophy and approach.
- ✓ Formulate your research design.
- ✓ Negotiate access to data.
- ✓ Collect data (done by Aizo).
- ✓ Analyze the data.
- ✓ Write the project report.

Therefore the research project can be regarded as a success. Especially the findings regarding run times are to be considered positive.

However, as explained in detail in the previous (sub-)chapters, certain aspects investigated as part of the project need to be looked at again or in more detail (especially periodic pattern mining and possibly also ways to decrease the memory consumption). Additionally, further steps are needed before the gained knowledge can be turned into usage in a real smart home. As a proposal, those steps are shortly outlined in the next chapter.

7.5 OUTLOOK AND THE NEXT STEPS

After this master thesis, follow-up research projects can build upon the findings here. To be more specific, to achieve the positive effects mentioned in the introduction like decreasing the energy consumption of a smart home or increasing the comfort of smart home inhabitant, (at least) the following steps will be necessary:

- Research the issue of periodic sequential pattern mining in a smart home environment.
- If memory consumption is an issue, research ways to minimize it (see the different propositions in chapter 6.7).

- Deduce and formulate rules from the detected patterns. This also includes trigger identification and event prediction, which are complex topics of their own.
- Store those rules in an appropriate format, e.g. object logic or PROLOG.
- Realize and implement all this in a multi-agent smart home to apply the rules in a real smart home (i.e. automation of the smart home appliances). To save electricity or increase the comfort, it is important that the newly introduced agents for pattern mining and rule execution are implemented as a predictive control systems (= before rules are applied, future needs and / or events are considered).
- Test and measure the increase of the positive effects in a real smart home.

BIBLIOGRAPHY

Agrawal, R. & Srikant, R., 1994. Fast algorithms for mining association rules. *Proc. 20th Int. Conf. Very Large Data Bases, VLDB,* Volume 1215, pp. 487-499. [Online] Available at: http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf [Accessed 24 November 2013].

Alam, M., Reaz, M. & Ali, M., 2012. A Review of Smart Homes - Past, Present, and Future. *IEEE Transactions on systems, man, and cybernetics,* 42(6), pp. 1190 - 1203.

Boutellier, R. & Gassmann, O., 2010. *Was ist der Unterschied zwischen qualitativer und quantitativer Forschung?* St. Gallen: Universität St. Gallen. [Online] Available at: www.tim.ethz.ch/education/courses/courses_fs_2010/course_docsem_fs_2010/papers/T14_Qual_Quan_Forschung_BLondiau_Falge [Accessed 28 May 2013].

Chikhaoui, B., Wang, S. & Pigot, H., 2010. A New Algorithm Based On Sequential Pattern Mining For Person Identification In Ubiquitous Environments. *KDD Workshop on Knowledge Discovery from Sensor Data,* pp. 19-28. [Online] Available at: http://users.cs.fiu.edu/~lzhen001/activities/KDD_USB_key_2010/workshops/W11%20SensorKDD%2710/p19-Chikhaoui.pdf [Accessed 17 October 2013].

CKW, 2012. *CKW verlängert ihr Pilotprojekt Smart Metering.* [Online] Available at: http://www.ckw.ch/internet/ckw/de/medien/news/archiv/2012/verlaengerung_smartmetering.html [Accessed 6 June 2013].

DeMaria, M., 2002. Home smart home. *Network Computing,* 13(1), pp. 55-60.

DeVaus, D. A., 2002. *Surveys in Social Research.* 5th ed. London: Routledge.

Dounis, A. & Caraiscos, C., 2009. Advanced control systems engineering for energy and comfort management in a building environment - A review. *Renewable Sustainable Energy Reviews,* 13(6–7), pp. 1246–1261.

Fayyad, U., Piatetsky, G. & Smyth, P., 1996. From Data Mining to Knowledge Discovery in Databases. *AI Magazine,* 17 (Fall), pp. 37 – 54. [Online] Available at: http://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1996-Fayyad.pdf [Accessed 20 July 2013].

Fournier-Viger, P., Gomariz, A., Soltani, A. & Gueniche, T., 2013. *SPMF: Open-Source Data Mining Platform.* [Online] Available at: http://www.philippe-fournier-viger.com/spmf/ [Accessed 24 November 2013].

Ghosh, S., Biswas, S., Sarkar, D. & Sarkar, P. P., 2010. Mining frequent itemsets using genetic algorithm. *International journal of artificial intelligence & applications,* 1(4), pp. 133-143.

Holland, J. H., 1975. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* Oxford: U Michigan Press.

Intertek and the Department of Trade and Industry, 2003. *Smart Home – A Definition.* Westminster: Department of Trade and Industry.

Jahn, M., Jentsch, M., Prause, C., Pramudianto, F., Al-Akkad, A. & Reiners R., 2010. *The Energy Aware Smart Home.* Sankt Augustin: Fraunhofer Institute for Applied Information Technology.

Keshavamurthy , B. N., Khan, A. M. & Toshniwal, D., 2013. Privacy preserving association rule mining over distributed databases using genetic algorithm. *Neural Computing and Applications,* 22(1), pp. 351-364.

Li, C., Yang, Q., Wang, J. & Li, M., 2012. Efficient Mining of Gap-Constrained Subsequences and Its Various Applications. *ACM Transactions on Knowledge Discovery from Data,* Volume 6, Article 2. [Online] Available at: http://dl.acm.org/ft_gateway.cfm?id=2133362&ftid=1185408&dwn=1&CFID=374308700&CFTOKEN=55391249 [Accessed 29 October 2013].

Obitko, M., 1998. *Introduction to Genetic Algorithms.* [Online] Available at: http://www.obitko.com/tutorials/genetic-algorithms/ [Accessed 14 May 2013].

Ott, T., 2013. *Discussion about the suitability of algorithms for pattern detection in the smart home environment* [Interview] (28 June 2013).

Pei, J. et al., 2004. Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING,,* 16(10). [Online] Available at: http://www.philippe-fournier-viger.com/spmf/prefixspan.pdf [Accessed 24 November 2013].

Rashidi, P. and Cook, D. J., 2009. Keeping the Resident in the Loop: Adapting the Smart Home to the User. *IEEE transactions on systems, man , and cybernetics - Part A: Systems and humans,* 39(5), pp. 949-959.

Rashidi, P., Cook, D. J., Holder, L. B. & Schmitter-Edgecombe, M., 2011. Discovering Activities to Recognize and Track in a Smart Environment. *IEEE Transactions on knowledge and data engineering,* 23(4), pp. 527-539. [Online] Available at: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3100559/ [Accessed 24 November 2013].

Saunders, M., Lewis, P. and Thornhill, A., 2009. *Research Methods for Business Students.* 5th ed. New Jersey: Prentice Hall.

Trochim, W. M., 2006. *Research methods knowledge base.* [Online] Available at: http://www.socialresearchmethods.net/kb/dedind.php [Accessed 21 April 2013].

Vaishnavi, V. & Kuechler, W., 2004. *Design Science Research in Information Systems.* [Online] Available at: http://desrist.org/design-research-in-information-systems/ [Accessed 21 April 2013].

Wang, J. & Han, J., 2004. BIDE: Efficient Mining of Frequent Closed Sequences. *Proceedings of the 20th International Conference on Data Engineering.,* pp. 79-90. [Online] Available at: http://www.philippe-fournier-viger.com/spmf/icde04_bide.pdf [Accessed 24 November 2013].

Wang, L., Wang, Z. & Yang, R., 2012. Intelligent Multiagent Control System for Energy and Comfort Management in Smart and Sustainable Buildings. *IEEE transactions on smart grid,* 3(2), pp. 605-617.

Wang, Z., Wang, L. & Yang, R., 2010. *Multi-agent Control System with Intelligent Optimization for Smart and Energy-efficient Buildings.* Toledo: University of Toledo.

Wu, X., Zhu, X., He, Y. & Arslan, A. N., 2013. PMBC: Pattern mining from biological sequences with wildcard constraints. *Computers in Biology and Medicine,* Volume 43, p. 481–492. [Online] Available at: http://www.sciencedirect.com/science/article/pii/S0010482513000504# [Accessed 29 October 2013].

Xie, F. et al., 2010. Sequential Pattern Mining with Wildcards. *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI),* pp. 241-247. [Online] Available at: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=5670041 [Accessed 29 October 2013].

Youngblood, R., 1986. 'Smart House' May Take Over In The 1990s. *Chicago Tribune*, 03 May 1986. [Online] Available at: http://articles.chicagotribune.com/1986-05-03/news/8602010304_1_wiring-system-smart-house-nahb [Accessed 23 April 2013].

Zöfel, P., 2003. *Statistik für Psychologen.* München: Pearson Studium.

FIGURES & TABLES

Title page: http://www.sigmadesigns.com/zwavenextgen/ [Accessed 23 November 2013].

Thesis Map Icon: http://mobtalker.wikia.com/wiki/File:Start-icon.png [Accessed 23 04 2013]

Thesis Map Icon: http://www.easyicon.net/language.en/icondetail/535943/ [Accessed 23 04 2013]

Thesis Map Icon: http://www.gewerbe-ehningen.de/images/kacheln/design.png [Accessed 23 04 2013]

Thesis Map Icon: http://www.blogger.com/profile/09690322836614683517 [Accessed 23 04 2013]

Thesis Map Icon: http://www.alpari.co.uk/newsroom/wp-content/themes/volt/scripts/timthumb.php?src=http://www.alpari.co.uk/newsroom/wp-content/uploads/2012/10/analysis.gif&w=110&h=106&zc=1&q=100 [Accessed 02 07 2013]

Thesis Map Icon: http://greendrycleaningindiana.com/wp-content/uploads/2012/10/Recycle-Tag.jpg [Accessed 07 05 2013]

Thesis Map Icon: http://www.clker.com/clipart-new-conclusion.html [Accessed 07 05 2013]

Thesis Map Icon: http://www.iconpng.com/png/pictograms/erlenmeyer-flask.png [Access 07 05 2013]

APPENDIX

PATTERNS IN THE POWER CONSUMPTION DATA

Since this project did not focus on the patterns available in the power consumption data but only on the ones in the event data, the initial data analysis done regarding patterns in power consumption data was removed from the main body of the work but is presented for the sake of completeness here in the following subchapters of the appendix.

NUMBER OF POWER MEASUREMENTS PER SMART HOME

The following SQL query can be used to get a general overview of key figures of the available power measurement data:

```
-- Get the minimum, the average and the maximum power consumption as well as the
number of power measurements in the DB per installation
SELECT AVG(avg_power) AS average, MIN(avg_power) AS minimum, MAX(avg_power) AS
maximum, MAX(avg_power)-MIN(avg_power) AS "range",  COUNT(avg_power) AS "no. of
measuerements", inst_id AS installation
FROM power_measurements
GROUP BY inst_id
ORDER BY range
```

The results of this query look as follows:

average	minimum	maximum	range	no. of measurements	installation
56.15	0.00	1'828.17	1'828.17	697'751	6
78.40	0.00	1'947.92	1'947.92	2'367'832	55
82.86	0.02	1'999.55	1'999.53	67'450	94
45.86	2.07	2'073.60	2'071.53	492'106	26
61.84	0.00	2'071.73	2'071.73	134'470	39
56.97	0.00	2'357.77	2'357.77	2'592'570	5
56.02	0.00	2'412.73	2'412.73	2'980'358	56
55.63	0.00	2'414.62	2'414.62	513'501	30
79.97	0.15	2'475.45	2'475.30	242'610	10
150.00	0.00	2'576.90	2'576.90	540'195	17
81.49	15.00	2'789.52	2'774.52	648'349	9
25.67	0.00	2'781.28	2'781.28	492'863	24
81.79	3.30	3'070.12	3'066.82	399'157	50
39.63	0.00	3'101.30	3'101.30	556'147	3
78.85	4.00	3'249.13	3'245.13	156'062	40
108.02	4.00	3'561.85	3'557.85	306'800	18
35.47	0.00	4'246.07	4'246.07	2'909'075	2
71.81	0.00	4'271.88	4'271.88	1'298'398	11
57.34	0.03	4'316.53	4'316.50	311'101	20

83.82	0.00	4'564.17	4'564.17	1'370'774	28
49.56	0.00	4'808.32	4'808.32	2'205'076	54
25.01	0.00	10'966.60	10'966.60	3'807'290	23

TABLE 57: NO., MIN., MAX. AND AVG. OF POWER MEASUREMENTS PER SMART HOME

If the above results are depicted in four column charts, they look as follows:

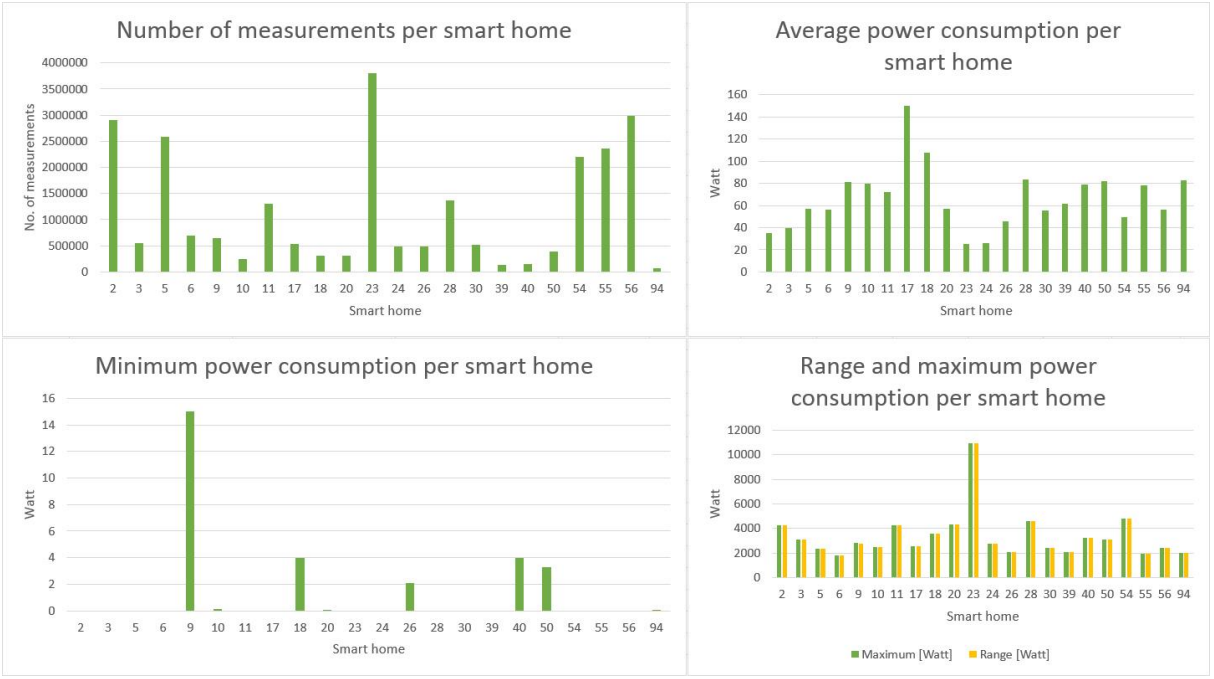


FIGURE 63: NO., MIN., MAX. AND AVG. OF POWER MEASUREMENTS PER SMART HOME (COLUMNS)

As can be seen from these results and visualization, there are big differences between the different smart homes regarding the number of recorded power measurements as well as the minimum, average and maximum measurement. However, as visible in the following chart, there is only a positive correlation between the maximum measurement (bright yellow line) and the range (dotted green line):

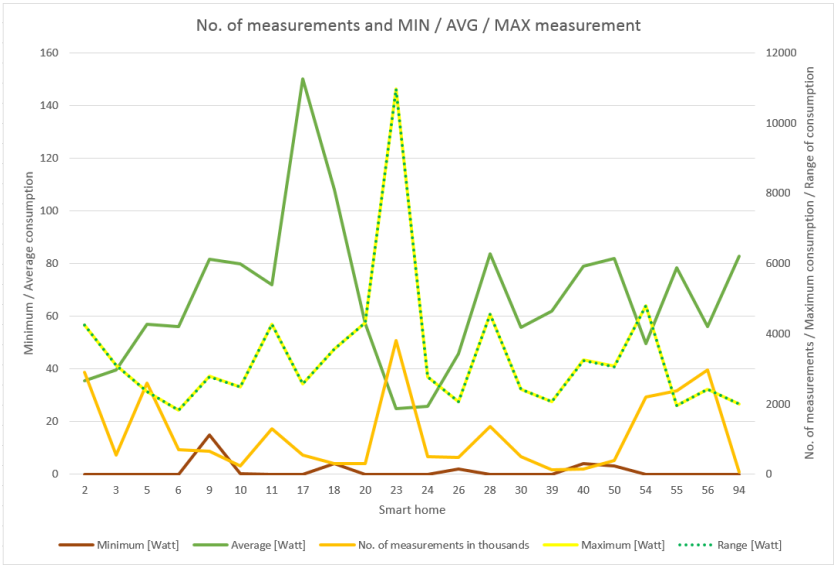


FIGURE 64: NO., MIN., MAX. AND AVG. OF POWER MEASUREMENTS PER SMART HOME (LINES)

All the other variables are uncorrelated, which was expected since none of the variables have a logical dependency from each other. The strong correlation between range and maximum can be explained with the fact that all the smart homes have a very low minimum power consumption near zero Watt and since $Range = maximum - minimum$, it becomes clear that if minimum is mostly zero the formula turns into $Range = maximum$, explaining the strong positive correlation of these two variables. The independence / weak correlation of the variables can also be seen from the following calculation of the correlation coefficients r and the scatterplot matrix done with the statistics software R:

	average	minimum	maximum	range	measurements
average	1.000000	0.228798	-0.319657	-0.320004	-0.396194
minimum	0.228798	1.000000	-0.087587	-0.089304	-0.235077
maximum	-0.319657	-0.087587	1.000000	0.999999	0.550018
range	-0.320004	-0.089304	0.999999	1.000000	0.550340
measurements	-0.396194	-0.235077	0.550018	0.550340	1.000000

TABLE 58: CORRELATION COEFFICIENTS OF NO., MIN., MAX. AND AVG. OF POWER MEASUREMENTS

Variable 1	Variable 2	Correlation coefficient r	Classification of correlation[2]	Interpretation
average	minimum	0.228798	Weak	Independent variables
average	maximum	-0.319657	Weak	Independent variables
average	range	-0.320004	Weak	Independent variables
average	measurements	-0.396194	Weak	Independent variables
minimum	maximum	-0.087587	Very Weak	Independent variables
minimum	range	-0.089304	Very Weak	Independent variables
minimum	measurements	-0.235077	Weak	Independent variables
maximum	range	0.999999	Very Strong	Range depends on maximum because minimum is 0
maximum	measurements	0.550018	Medium	Chance for higher measurements rises with number of measurements
range	measurements	0.550340	Medium	Chance for higher measurements rises with number of measurements

TABLE 59: INTERPRETATION OF THE CORRELATION COEFFICIENTS

[2] Classification according to Zöfel (2003, p. 151)

Presented as a scatterplot matrix the correlation looks as follows:

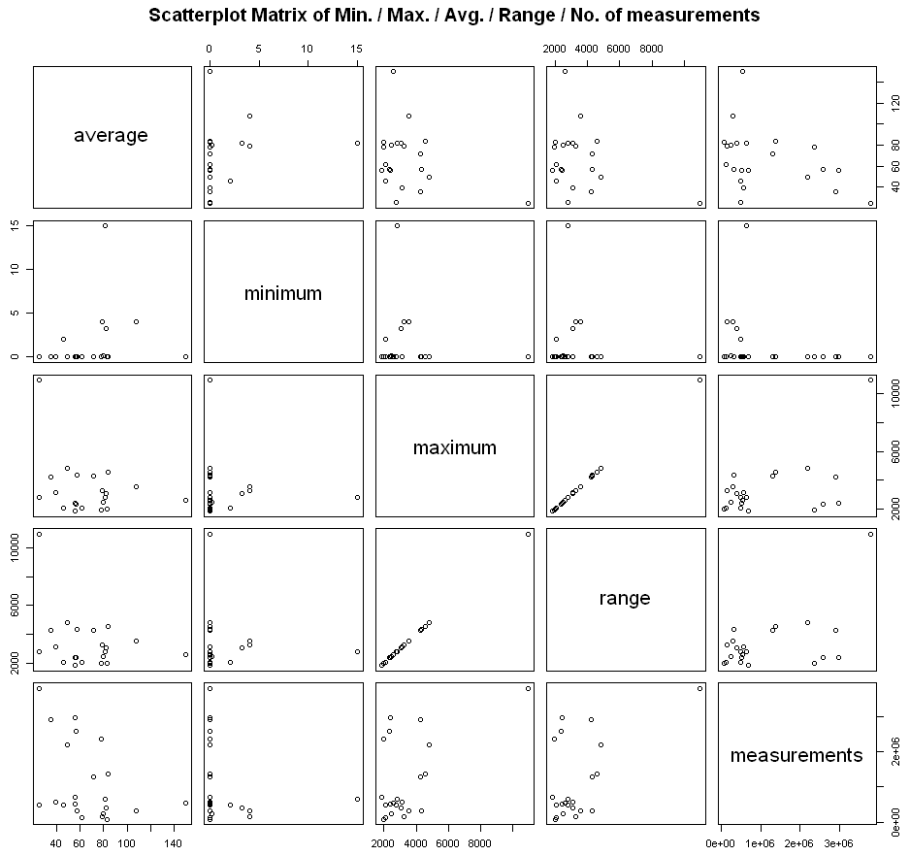


FIGURE 65: SCATTERPLOT MATRIX OF NO., MIN., MAX. AND AVG. OF POWER MEASUREMENTS

When looking at the scatterplot matrix and the correlation coefficients, it becomes obvious that what at first sight of Figure 64 might appear as additional dependencies (e.g. between the number of measurements and range / maximum at smart home 23 or the one between the number of measurements, range / maximum and the average at smart home 28) have to be interpreted as coincidental.

For further analysis this means that only the dependency between maximum and range needs to be kept in mind.

POWER CONSUMPTION CURVES

For the purpose of looking at the different power consumption curves, for each smart home the consumption of a whole day, 13 March 2013, was analyzed with the following SQL query:

```
DECLARE @inst_id int
DECLARE smart_homes_cursor CURSOR FOR
       SELECT DISTINCT inst_id
       FROM events
OPEN smart_homes_cursor
FETCH NEXT FROM smart_homes_cursor
INTO @inst_id
WHILE @@FETCH_STATUS = 0
BEGIN
```

```
            SELECT pm.time, SUM(pm.avg_power) AS "Power consumption per minute",
@inst_id "Installation"
            FROM power_measurements AS pm
            WHERE inst_id = @inst_id
            AND time BETWEEN '2013-03-13' AND '2013-03-14'
            GROUP BY time
            ORDER BY time
            FETCH NEXT FROM smart_homes_cursor
            INTO @inst_id
END
CLOSE smart_homes_cursor
DEALLOCATE smart_homes_cursor
```

Above query returns a total of 24 result sets, one for each smart home, containing 1440 measurements each (24 hours x 60 minutes). Since that would be too big to be include here, these numbers are omitted and only the line graphs for smart homes 2 through 26 are presented in the following figure:

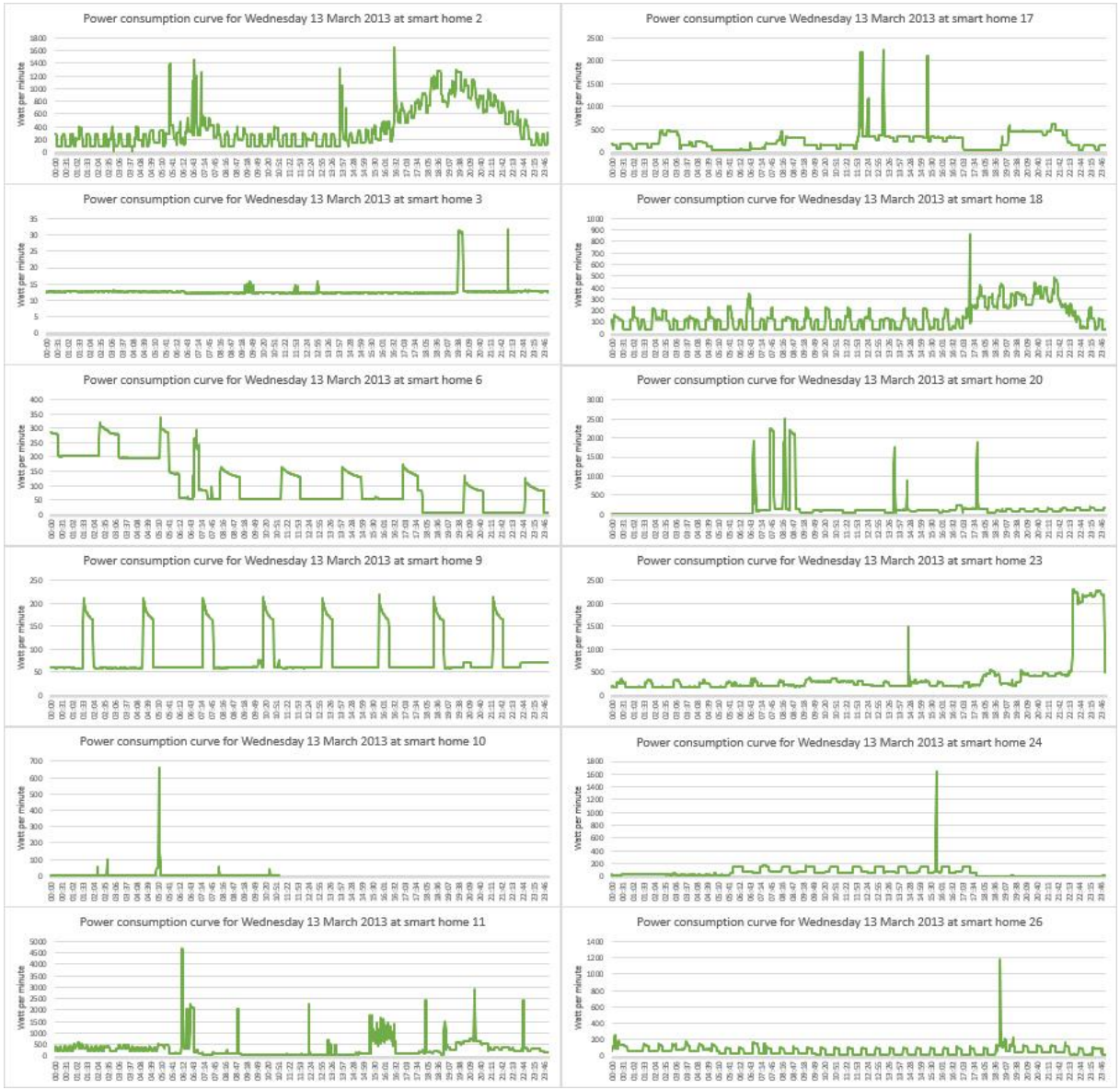


FIGURE 66: POWER CONSUMPTION CURVES PER SMART HOME FOR 13 MARCH 2013 (PART 1)

For the remaining smart homes, no. 28 through 94, see the next page:

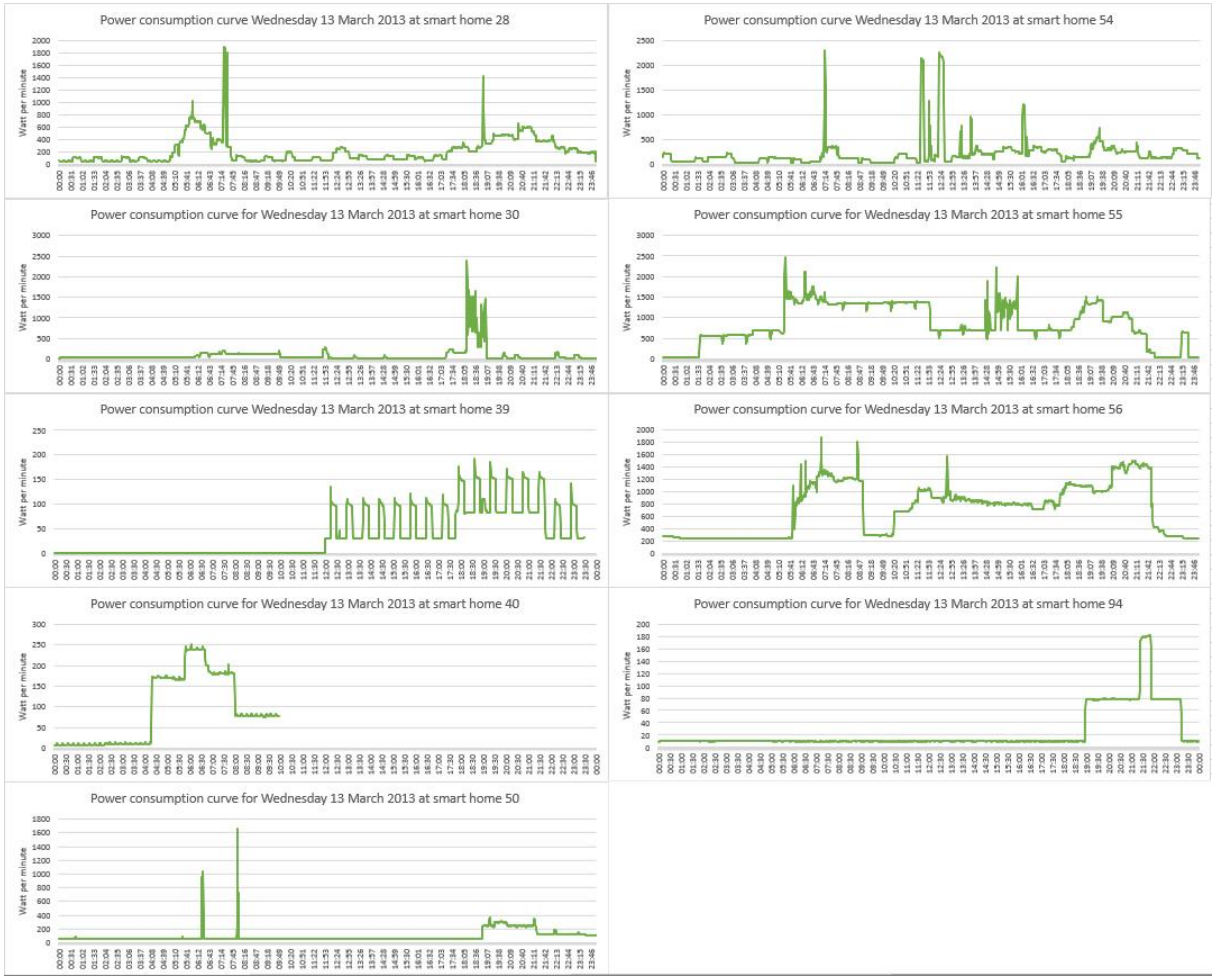


FIGURE 67: POWER CONSUMPTION CURVES PER SMART HOME FOR 13 MARCH 2013 (PART 2)

For the missing smart homes, these are no. 5, 7 and 34, the data for the 13th of March 2013 is unavailable. However, the available data and the resulting power consumption curves for the other 21 smart homes show nicely that every smart home has his own power consumption signature.

From this analysis it can be seen that certain smart homes have a quite regular power consumption resulting in more obvious patterns while others have rather irregular consumptions.

THE JAVA SOURCE CODE

To avoid a bloating of the paper, the roughly 3'000 lines of Java source code written for this research project can be found on the CD accompanying this paper.

ADDITIONAL EXTERNAL RESOURCES

For the Java source code written for this project to work, the following additional and external sources are needed:

- Java Runtime Environment (tested with 1.7.0_45, 64-Bit): http://www.java.com/de/download/manual.jsp [Accessed 27 Januar 2014]
- SPMF source code from: http://www.philippe-fournier-viger.com/spmf/spmf.zip [Accessed 29 December 2013]
- Python Version of GapBIDE: https://code.google.com/p/pygapbide/source/browse/trunk/pygapbide.py [Accessed 29 December 2013]
- The JDBC driver for MSSQL (tested with MSSQL 2012): http://go.microsoft.com/fwlink/?LinkId=245496 [Accessed 29 December 2013]
- The confidential Aizo data set stored in a relational MSSQL database (tested with MSSQL 2012). To get access to the confidential data set contact Holger Wache at FHNW.