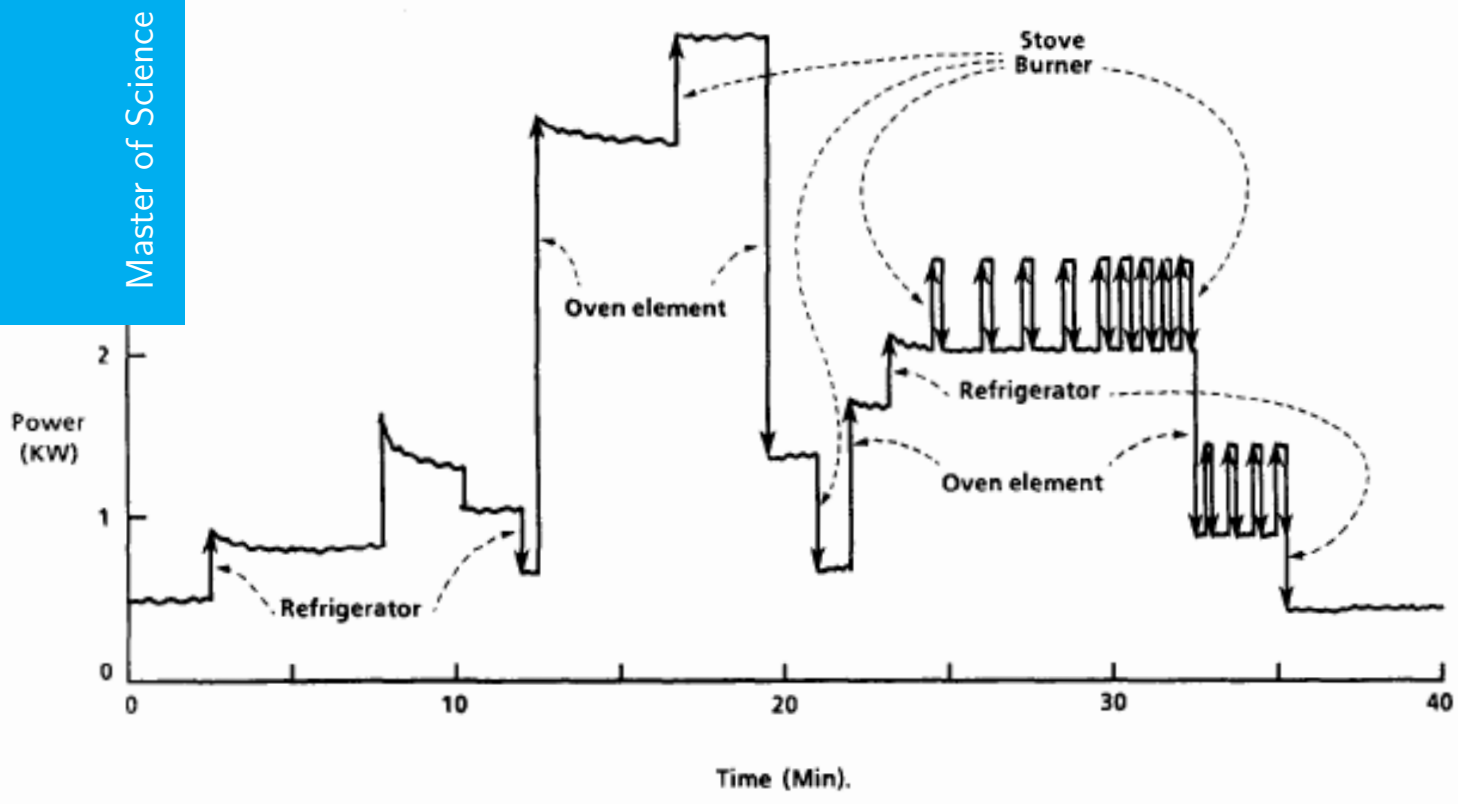
Energy disaggregation: Non-intrusive load monitoring

The search for practical methods

W.H.C. Janssen

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Energy disaggregation: Non-intrusive load monitoring

The search for practical methods

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

W.H.C. Janssen

January 11, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis is supported by Mobile Water Management.

Abstract

In this MSc. thesis steady-state disaggregation methods for the Non-Intrusive Load Monitoring are researched. The main target is to find methods that have the potential to be practically applicable. These methods are characterized by the fact that they use data that is collected by smart meters, since smart meters are more and more the standard in Western households. Appliances are modeled by Hidden Markov Models and households are modeled by super-state Hidden Markov Models or Factorial Hidden Markov Models. These models are trained by iterative K-means or expectation maximization, where iterative K-means turns out as the superior training method. Basically three disaggregation algorithms are researched: the Viterbi algorithm, the particle filter, and the newly proposed Global Transition Minimization (GTM).

The disaggregation algorithms are tested on a synthetic dataset of 6 appliances and on part of the Dutch Residential Energy Dataset (DRED) [15]. The results on the synthetic dataset vary between an accuracy of 81% for the GTM and 95% for the Viterbi algorithm. On the DRED the highest accuracy that is achieved is 77% for the particle filter. It turned out that the accuracy of the particle filter is not always improved by increasing the number of particles when real-world data is considered.

Improvements for the disaggregation algorithms by using time information data and by using reactive power as extra input data are proposed. However, both suggestions do not lead to more accurate disaggregation results. A new modeling framework, where state transitions are also included in the model, is suggested as a future research topic.

Contents

Preface

This MSc. thesis is done at the Delft Center for Systems and Control, part of the Delft University of Technology. The survey was introduced by the Mobile Water Management company. Their main business case is the development of new data acquisition techniques where humans in the loop contribute to the data acquisition. An example of their products is a smart phone application that is able to measure water heights by making pictures of staff gauges. Based on an international project proposal we came to the energy disaggregation problem. Initially the project aimed to disaggregate load consumption even when the house lacks a smart meter. Since there is a lot of progress in the roll out of smart meters in Western countries, it is assumed that smart meter data is available for the research.

Chapter 1

Introduction

1-1 General introduction

Today's society is more and more depending on electricity. Where nowadays most cars are still driven by fossil fuels and cooking is done on gas, future prediction is that electricity will replace a lot of these fossil fuels [1]. More and more people mount solar cells on their roofs, turning their homes in small electricity factories. This will lead to a more fluctuating availability of electrical energy. Additional insight in energy consumption could save energy and could give a better prediction for the power company of the energy needed.

According to the European Union 72% of the European citizens will have a smart meter for electricity by 2020 [2]. On average these smart meters can provide an energy saving of about 3% per metering point due to the additional insight consumers can get of their energy consumption [3]. Currently smart meter measurements can be collected by mobile applications or by so-called smart thermostats like Toon®. With these devices one can get insight in the total energy consumption and production (with solar cells for example). A warning can be sent when the consumption is high compared to other consumers. The lack these so-called 'smart' meters have is that only the total energy consumption of a residence can be measured. However, a consumer will not be able to effectively change his or her habits based on these measurements only. Appliance-wise energy consumption data would be the solution to this problem. One way to get these appliance-wise energy consumption values is to separately measure the energy consumption of each appliance. This would require a power meter for each appliance, which brings high purchase and installation costs. Another method is to disaggregate the total power signal entering a house through the smart meter. This thesis will be dedicated to the disaggregation of the total energy consumption.

The disaggregation of electricity consumption is also called Non-Intrusive Load Monitoring (NILM). NILM was first suggested in the 1980s by G. W. Hart [7], but only the last few years, since the upcoming availability of smart meters, the research in NILM has particularly grown. Unfortunately, up until today there is not yet an algorithm designed that can disaggregate the energy consumption of a residence accurately.

Theoretically accurate energy disaggregation is beneficial for much more than only giving

insight in appliance-wise energy consumption. Accurate load monitoring could detect aging appliances and possibly via a smartphone application suggest a consumer to replace an appliance by a newer model in order to save energy and money over time. Another possible application of NILM is to detect malfunctioning appliances or appliances that need maintenance in order to save energy, i.e. a freezer that needs to be de-iced. At last, due to the larger availability of sustainable power sources the energy supply rate will be more and more fluctuating and with it will be the price of electric energy. If people have more insight in which appliances consume a lot of energy, they can choose to only turn on these appliances when the supply of energy is high and the price is low. NILM could thus help to regulate the supply-demand market.

Figure 1-1 shows a schematic overview of where the disaggregation algorithm, which is the topic of this thesis, is placed in the real world situation. Smart meter measurements and data about the consumption patterns of individual appliances is used within the disaggregation algorithm to generate the disaggregated appliances-wise energy consumption data. Based on this data together with historical consumption data, energy price information data (based on the availability of energy), and other usage data (e.g. usage data of comparable households) an advice is given to the consumer. The generation of this advice is not a topic of this thesis.

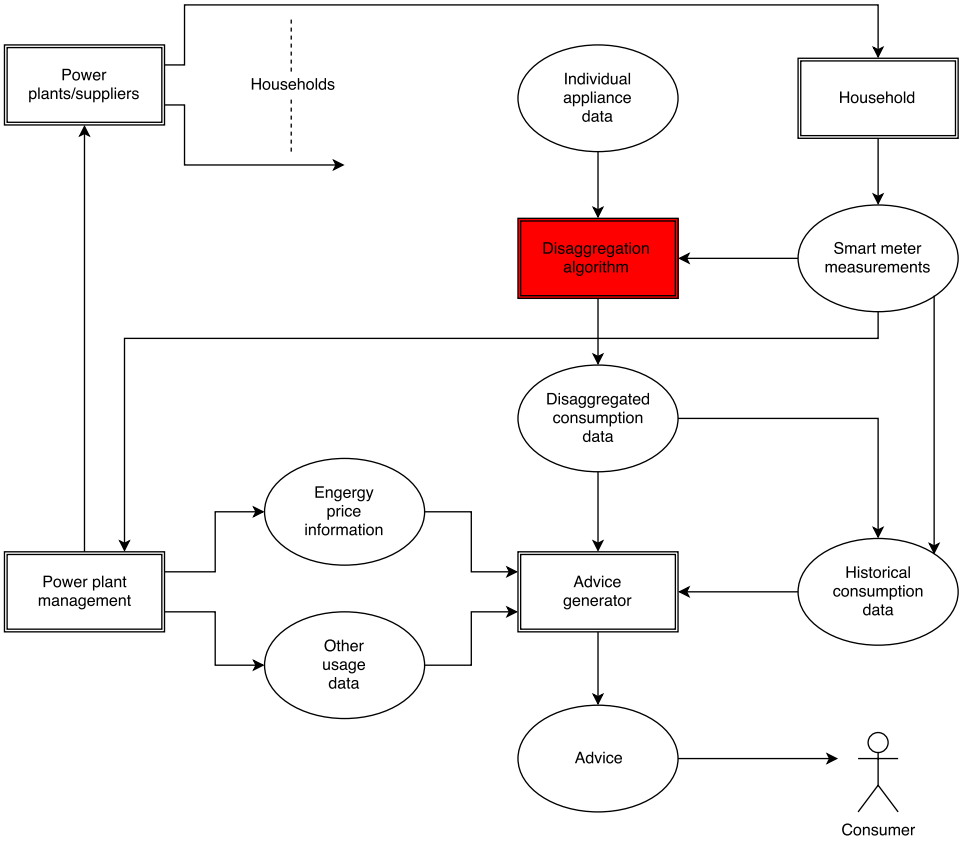


Figure 1-1: The place of the disaggregation algorithm in the real-world situation.

1-2 Problem statement

As mentioned in Section 1-1 the research to NILM is gaining more interest over the last years. Several approaches and several methods have been suggested, but up until today none of them has proven to be able to successfully solve the disaggregation problem for a real-life situation. It is far too ambitious for this thesis to come up with a disaggregation algorithm that does that, but it is possible to figure out which approaches and methods can potentially solve the NILM problem and which methods will not be able to do that. Therefore, the problem statement for this thesis is: **Which methods might contribute to a practical solution of the NILM problem?** According to [5] there are six criteria for a NILM disaggregation method in order to be able to come up with a practical solution, i.e. to disaggregate a real-world dataset. The criteria are listed from most important to least important for this thesis;

1. The signatures of the appliances may only be based on the active power sampled at 1 Hz.
2. The minimum acceptable accuracy of the NILM algorithm is 80%-90%.
3. The algorithm should perform in real time.
4. The method should be scalable in the sense of robustness and number of appliances up to 20-30 devices.
5. The algorithm should work with various types of appliances: ON/OFF appliances, multi-state appliances, and continuously consuming appliances.
6. No algorithm training should be necessary.

The first criterion is important since the current available smart meters are not able to measure higher frequencies than 1 Hz. The fourth criterion is important since a simple household includes at least about 20 appliances. The method should be robust, i.e. accuracy should not drop with an increasing number of appliances.

This thesis tries to come up with methods that 'might contribute' to a practical solution. Such a method should obey the first criterion and it should be made realistic that the other criteria can be satisfied, with or without the use of other methods.

1-3 Related work

According to [4] and others all NILM disaggregation methods can be categorized as either *load classification* or *load separation*. In the case of load classification methods, first a step of signature detection is applied for each appliance. Distinct features of an appliance, which can be steady-state or transient features, form a signature for that appliance. In a second step these signatures are used to recognize appliances from an aggregated power signal. On the other hand in the case of load separation methods, the disaggregation is based on matching the power signal of individual appliances with the aggregated power signal. Based on the

first criterion of Section 1-2 all load classification methods can be ruled out since they need high-frequency data in order to obtain transient features for the signatures. Hence, only load separation methods, which are steady-state methods, are eligible for solving the practical NILM problem.

All steady-state methods rely on probabilistic models and optimize the state sequence for each appliance within the aggregated power signal. Steady-state methods can be subdivided into two approaches. The first one is based on Hidden Markov Models (HMMs) and the second one on Non-negative Matrix Factorization (NMF) [13]. A Hidden Markov Model (HMM) is a probabilistic model in which the system being modeled is assumed to be a Markov process. In the case of the NILM problem appliances can be modeled with HMMs and a household can be modeled as a super-state HMM or a Factorial Hidden Markov Model (FHMM). On the other hand NMF tries to minimize the norm

$$\|\mathbf{Y} - \mathbf{X}\mathbf{A}\|_{\mathrm{F}}^2 \tag{1-1}$$

with respect to $\mathbf{X}$ and $\mathbf{A}$. Here $\mathbf{Y}$ is a matrix with the aggregated power measured by the smart meter, $\mathbf{X}$ are typical consumption patterns of individual appliances, and $\mathbf{A}$ is a binary matrix that indicates which columns of $\mathbf{X}$ contribute to $\mathbf{Y}$. At first sight NMF does not seem to have any relation with probabilistic models, but the following derivation makes clear it does. Assume that $y_{t,d}$ is an element of the matrix $\mathbf{Y}$ that can be factored as $\sum_k x_{t,k}a_{k,d}$, where $x_{t,k}$ and $a_{k,d}$ are elements of $\mathbf{X}$ and $\mathbf{A}$ respectively. Assume that the total energy consumption at time t on day d is given by a Gaussian distribution:

$$P\left(y_{t,d}\left|\sum_{k=1}^{K} x_{t,k}a_{k,d}, \sigma^2\right.\right) \sim \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_{t,d} - \sum_{k=1}^{K} x_{t,k}a_{k,d}\right)^2}{2\sigma^2}\right) \quad (\forall\, t, d) \tag{1-2}$$

The maximization of the likelihood with respect to $\mathbf{X}$ and $\mathbf{A}$ is given by

$$\begin{aligned}\underset{\mathbf{X},\mathbf{A}}{\operatorname{argmax}} \sum_{t=1}^{T}\sum_{d=1}^{D} \log P\left(y_{t,d}\left|\sum_{k=1}^{K} x_{t,k}a_{k,d}, \sigma^2\right.\right) &= \underset{\mathbf{X},\mathbf{A}}{\operatorname{argmin}} \sum_{t=1}^{T}\sum_{d=1}^{D}\left(y_{t,d} - \sum_{k=1}^{K} x_{t,k}a_{k,d}\right)^2 \\ &= \underset{\mathbf{X},\mathbf{A}}{\operatorname{argmin}} \|\mathbf{Y} - \mathbf{X}\mathbf{A}\|_{\mathrm{F}}^2\end{aligned} \tag{1-3}$$

Hence, solving the NMF problem is the same as maximizing a likelihood and the maximization of a likelihood is also the way disaggregation based on HMMs is performed.

1-4 Contributions of this thesis

As mentioned in Section 1-2 the main aim of this thesis is to contribute in the search for practical NILM methods. Since the results of disaggregation algorithms based on NMF are not promising [13], e.g. the disaggregation with only 4 appliances resulted in an average accuracy of about 35%, NMF will not be a topic of this thesis. This thesis treats the HMM-based approaches with as disaggregation algorithms the (sparse) Viterbi algorithm and the particle filter, since these two algorithms seemed promising from the literature survey [17]. Where in other studies most of the time a single model and a single algorithm is tested on different

instances of data, this thesis tests different models and different algorithms on the same data such that a qualitative comparison between the methods is possible.

Further, a hybrid method between super-state HMM and FHMM is introduced and tested. Also minor improvements, like the use of time-dependent transition matrices for HMM and the use of reactive power data besides the use of active power data, are proposed and evaluated.

At last a new optimization method, the Global Transition Minimization (GTM), is introduced that is based on the minimization of the number of state transitions. For testing the disaggregation algorithms a synthetic dataset is used and also the Dutch Residential Energy Dataset (DRED) dataset that is provided in [15] is used.

1-5 Structure of the thesis

In Chapter 2 an overview of the existing disaggregation methods, that are promising from the literature, and an overview of the required HMMs for these disaggregation methods is given. Also the training of HMMs is treated. In Chapter 3 the new GTM is introduced as well as three suggestions for improvements on the existing methods. All methods from Chapter 2 and Chapter 3 will be tested with synthetic data in Chapter 4. In Chapter 5 the real-world DRED will be used to test the two best performing disaggregation methods from Chapter 4. At last the results from Chapter 4 and Chapter 5 will be reviewed in Chapter 6, where also future work in the field of NILM will be discussed.

Chapter 2

Existing disaggregation methods

This chapter will give an overview of disaggregation methods that are already present in the literature. Hidden Markov Models (HMMs) are used to model appliances and super-state HMMs or Factorial Hidden Markov Model (FHMM) are used to model a household. Training of the models is done with expectation maximization or with iterative K-means. The (sparse) Viterbi algorithm and the particle filter are presented as disaggregation algorithms.

2-1 Hidden Markov Models

2-1-1 General introduction

The following description of HMMs is given based on [9]. HMMs are statistical models in which the system being modeled is assumed to be a Markov process with hidden states. This means that the Markov property, $P(x_t|x_{1:t-1}) = P(x_t|x_{t-1})$, holds for the state transitions. Here x_t denotes the state of the system at time t and $x_{1:t-1} = (x_1, x_2, \ldots, x_{t-1})$. The states of a Markov process are discrete. This property is perfect for the modeling of appliances for steady-state methods, since these steady-states are also discrete. The speciality of a Hidden Markov Model (HMM) compared to a normal Markov chain is that the true states are hidden, i.e. they cannot be measured directly. The measurement of state x_t is denoted by y_t. Any measurement distribution can be imposed on y_t, but most of the time, and also in this thesis, y_t is a normal distribution with mean x_t. Figure 2-1 shows a schematic representation of an HMM.

Mathematically an HMM can be described as follows:

$$\pi_i = P(x_1 = i) \tag{2-1}$$

$$A_{i,j} = P(x_t = j|x_{t-1} = i) \tag{2-2}$$

$$y_t \sim \mathcal{N}\left(x_t, \sigma^2_{x_t}\right) \tag{2-3}$$

Here π_i is the probability that at $t = 1$ the Markov chain is in the i^{th} state and $A_{i,j}$ is the entry of the transition matrix A that represents the probability that when at $t-1$ the system

is in state i the system will be in state j at t. At last y_t is given by a normal distribution with mean x_t and standard deviation σ_{x_t}.

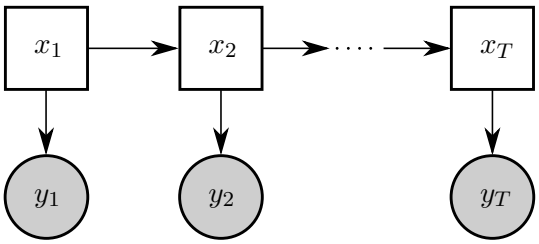


Figure 2-1: Schematic representation of an HMM, where x_t are the hidden states and y_t are the measurements.

2-1-2 Super-state Hidden Markov Models

An HMM can be used to model a single appliance. The different steady-state power consumption levels form the states of the HMM on which the parameters of the HMM, as described in Section 2-1-1, can be trained. In order to model a complete household with HMMs the super-state Hidden Markov Model is introduced [11]. Consider two appliances with both an ON and an OFF state. The combined system of the two appliances has four states: both appliances ON, both appliances OFF, appliance 1 ON and appliance 2 OFF, and the other way around. The combined system can be considered as a normal HMM. Of course more appliances with more than two states can be included in the super-state HMM, but the size of the HMM increases exponentially. Assume that a super-state HMM is created with ten appliances with two states each. The resulting super-state HMM has $2^{10} = 1024$ states. This is the downside of super-state HMM; the storage of system matrices requires a lot of memory.

In practice the creation of a super-state HMM is done by using the Kronecker product ($\otimes$) on the system matrices and vectors. For example if two-state appliance 1 with transition matrix $A^{(1)}$ and appliance 2 with transition matrix $A^{(2)}$ will form a super-state HMM the Kronecker product will give the transition matrix of the super-state HMM: $A = A^{(1)} \otimes A^{(2)}$. This can be written out as:

$$A = \begin{bmatrix} A^{(1)}_{1,1}A^{(2)} & A^{(1)}_{1,2}A^{(2)} \\ A^{(1)}_{2,1}A^{(2)} & A^{(1)}_{2,2}A^{(2)} \end{bmatrix} \tag{2-4}$$

Hence, the individual entries of $A^{(1)}$ are multiplied with $A^{(2)}$. The result is that the new super-states are combinations of states of individual appliances. Since we are dealing with probabilities the entries of individual transition matrices are between 0 and 1. For that reason the entries of A will get smaller and smaller if the number of appliances grows. Eventually this can result in numerical underflow, i.e. the precision of the computer is unable to differentiate the non-zero entries obtained by multiplication from 0. To overcome this issue the system parameters can be converted to log-space [14] and a dual of the Kronecker product in log-space can be applied.

2-1-3 Factorial Hidden Markov Models

Another option to model a complete household is by using an FHMM. In the case of the FHMM every appliance has its own Markov chain. The propagation through these chains goes independently for each appliance. A schematic representation of the FHMM is shown in Figure 2-2. The FHMM consists of N HMMs, which are N individual appliances. The measurement y_t is a summation of the measurements of the individual chains: $y_t = \sum_{n=1}^{N} y_t^{(n)}$. The difference between the super-state HMM and the FHMM is that in the case of the FHMM y_t is a summation of measurements of the individual chains, where in the case of the super-state HMM y_t is a measurement of a superposition of states. At first sight this may not seem as much of a difference, but an FHMM does require much less memory for the storage of the transition matrices than a super-state HMM when the number of appliances (actually the number of super-states) is large. On the other hand a disaggregation algorithm for the FHMM requires much more computation time, which will become clear later in this thesis.

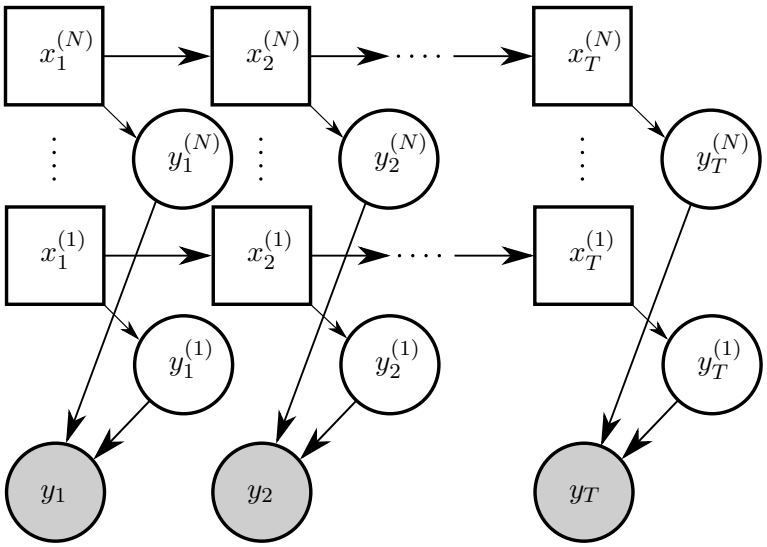


Figure 2-2: Schematic representation of an FHMM, where $x_t^{(n)}$ are the hidden states and y_t are the measurements.

2-2 Training of Hidden Markov Models

2-2-1 Training of Hidden Markov Models from states

In order to create an HMM of an appliance the system matrices of the HMM need to be trained correspondingly. Although the sixth criterion in Section 1-2 says that no training is allowed, it is not possible to create a model without training. For this thesis it is assumed that training data of individual appliances is available. In [14] it is further assumed that the number of states an appliance has is known and that even the power consumption value of each state is known. Suppose that for some appliance the state data $x_{1:T}$ is available for training.

An indicator function is defined as follows:

$$\mathbf{1}_i(x) := \begin{cases} 1 & \text{if } x = i \\ 0 & \text{if } x \neq i \end{cases} \tag{2-5}$$

With the use of an indicator function the HMM parameters can be written as:

$$A_{i,j} = \frac{\sum_{t=2}^{T} \left[\mathbf{1}_i(x_{t-1}) \cdot \mathbf{1}_j(x_t)\right]}{\sum_{t=1}^{T-1} \mathbf{1}_i(x_t)} \tag{2-6}$$

$$\pi_i = \frac{1}{T} \sum_{t=1}^{T} \mathbf{1}_i(x_t) \tag{2-7}$$

The value of π_i is simply the percentage of the time that $x_t = i$. This is reasonable value for π_i when it is arbitrary when a measurement sequence is started. Now assume that also the measurement data $y_{1:T}$ is available. The parameters of the Gaussian measurement distribution can be determined as:

$$\mu_i = \frac{\sum_{t=1}^{T} y_t \cdot \mathbf{1}_i(x_t)}{\sum_{t=1}^{T} \mathbf{1}_i(x_t)} \tag{2-8}$$

$$(\sigma_i)^2 = \frac{\sum_{t=1}^{T} \left[(y_t - \mu_i)^2 \cdot \mathbf{1}_i(x_t)\right]}{\sum_{t=1}^{T} \mathbf{1}_i(x_t)} \tag{2-9}$$

The result is that the output of the system is normally distributed; when $x_t = i$: $y_t \sim \mathcal{N}\left(\mu_i, (\sigma_i)^2\right)$.

2-2-2 Training of Hidden Markov Models with Expectation Maximization

In contrast with Section 2-2-1 it is now assumed that there is no state data, $x_{1:T}$, available. However, it is still assumed that it is known in how many state clusters the measurements $y_{1:T}$ should be divided. Every state cluster results in one steady-state value of the appliance. Suppose that all unknown parameters are presented by a single parameter set: $\theta = \left\{\pi_i, A_{i,j}, \mu_i, (\sigma_i)^2\right\}$ for all i, j system states. The idea is to fit the model parameters optimally to the measurement data:

$$\bar{\theta} = \text{argmax}_\theta P\left(y_{1:T}|\theta\right) \tag{2-10}$$

where $\bar{\theta}$ is the estimated parameter set for the model.

The most common approach to estimate θ is through the expectation maximization algorithm [8]. The expectation maximization algorithm is an iterative process that consists of two steps: the E(xpectation)-step and the M(aximization)-step. During the E-step the probability of each state at each time step is computed given the measurement data $y_{1:T}$ and an estimate of θ. Also the probability of a state transition between each pair of states is computed during the E-step. During the M-step all parameters of θ are updated. In the next iteration the updated value of θ is used during the M-step.

For the expectation maximization algorithm forward-backward variables need to be defined. Given that the appliance under consideration has K states, the forward variable is defined as:

$$\begin{aligned}\alpha_t(j) &= P_\theta\left(x_t = j, y_{1:t}\right)\\ &= P_\theta\left(y_{1:t}|x_t = j\right) P_\theta\left(x_t = j\right)\\ &= P_\theta\left(y_{1:t-1}|x_t = j\right) P_\theta\left(y_t|x_t = j\right) P_\theta\left(x_t = j\right)\\ &= P_\theta\left(x_t = j, y_{1:t-1}\right) P_\theta\left(y_t|x_t = j\right)\\ &= \left(\sum_{i=1}^{K} \alpha_{t-1}(i) A_{i,j}\right) P_\theta\left(y_t|x_t = j\right)\end{aligned} \tag{2-11}$$

where Bayes rule is used twice. The subscript P_θ indicates that θ is given and the notation $P_\theta\left(x_t = j, y_{1:t}\right)$ is equivalent to $P_\theta\left(\left(x_t = j\right) \& \left(y_{1:t}\right)\right)$. The backward variable is defined as:

$$\begin{aligned}\beta_t(i) &= P_\theta\left(y_{t+1:T}|x_t = i\right)\\ &= \sum_{j=1}^{K} A_{i,j} P_\theta\left(y_{t+1:T}|x_{t+1} = j\right)\\ &= \sum_{j=1}^{K} A_{i,j} P_\theta\left(y_{t+1}|x_{t+1} = j\right) P_\theta\left(y_{t+2:T}|x_{t+1} = j\right)\\ &= \sum_{j=1}^{K} A_{i,j} P_\theta\left(y_{t+1}|x_{t+1} = j\right) \beta_{t+1}(j)\end{aligned} \tag{2-12}$$

The initial values for the forward and backward variable are defined as:

$$\alpha_1(j) = P_\theta\left(y_1|x_1 = j\right) \pi_j \tag{2-13}$$

$$\beta_T(i) = 1 \tag{2-14}$$

As mentioned, in the E-step the probability of each state at each time step is computed given the measurement data sequence:

$$\begin{aligned}\gamma_t(i) &= P_\theta\left(x_t = i|y_{1:T}\right)\\ &= \frac{P_\theta\left(y_{1:T}|x_t = i\right) P_\theta\left(x_t = i\right)}{P_\theta\left(y_{1:T}\right)}\\ &= \frac{1}{P_\theta\left(y_{1:T}\right)} P_\theta\left(y_{t+1:T}|x_t = i\right) P_\theta\left(y_{1:t}|x_t = i\right) P_\theta\left(x_t = i\right)\\ &= \frac{1}{P_\theta\left(y_{1:T}\right)} \beta_t(i) P_\theta\left(y_{1:t}, x_t = i\right)\\ &= \frac{1}{P_\theta\left(y_{1:T}\right)} \alpha_t(i)\beta_t(i)\end{aligned} \tag{2-15}$$

The probability of a state transition between each pair of states at every time step is given by:

$$\begin{aligned}
\xi_t(i,j) &= P_\theta\left(x_{t-1}=i, x_t=j|y_{1:T}\right) \\
&= \frac{P_\theta\left(x_{t-1}=i, x_t=j, y_{1:T}\right)}{P_\theta\left(y_{1:T}\right)} \\
&= \frac{1}{P_\theta\left(y_{1:T}\right)} P_\theta\left(y_{1:T}|x_{t-1}=i, x_t=j\right) P_\theta\left(x_{t-1}=i, x_t=j\right) \\
&= \frac{1}{P_\theta\left(y_{1:T}\right)} P_\theta\left(y_{1:T}|x_{t-1}=i, x_t=j\right) P_\theta\left(x_t=j|x_{t-1}=i\right) P_\theta\left(x_{t-1}=i\right) \\
&= \frac{1}{P_\theta\left(y_{1:T}\right)} P_\theta\left(y_{1:t-1}|x_{t-1}=i, x_t=j\right) P_\theta\left(y_t|x_{t-1}=i, x_t=j\right) \\
&\qquad P_\theta\left(y_{t+1:T}|x_{t-1}=i, x_t=j\right) P_\theta\left(x_t=j|x_{t-1}=i\right) P_\theta\left(x_{t-1}=i\right) \\
&= \frac{1}{P_\theta\left(y_{1:T}\right)} P_\theta\left(y_{1:t-1}|x_{t-1}=i\right) P_\theta\left(y_t|x_t=j\right) \\
&\qquad P_\theta\left(y_{t+1:T}|x_t=j\right) P_\theta\left(x_t=j|x_{t-1}=i\right) P_\theta\left(x_{t-1}=i\right) \\
&= \frac{1}{P_\theta\left(y_{1:T}\right)} P_\theta\left(x_t=j|x_{t-1}=i\right) \alpha_{t-1}(i)\beta_t(j) P_\theta\left(y_t|x_t=j\right)
\end{aligned} \tag{2-16}$$

The probability $P_\theta\left(y_{1:T}\right)$ can be computed as follows:

$$\begin{aligned}
P_\theta\left(y_{1:T}\right) &= \sum_{i=1}^{K} P_\theta\left(y_{1:T}, x_T=i\right) \\
&= \sum_{i=1}^{K} \alpha_T(i)
\end{aligned} \tag{2-17}$$

In the M-step all parameters of the HMM are updated correspondingly:

$$\bar{\pi}_i = \sum_{t=1}^{T} \gamma_t(i) \tag{2-18}$$

$$\bar{A}_{i,j} = \frac{1}{Z} \sum_{t=2}^{T} \xi_t(i,j) \tag{2-19}$$

where Z is a normalizing constant to ensure that the transition matrix A is a stochastic matrix, hence all rows of A should sum to 1. The parameters μ_i and σ_i are updated by fitting a Gaussian distribution to the contribution of the measurement sequence. A measurement y_t is contributing to state i if $\gamma_t(i) > \gamma_t(j)$ for all states $j \neq i$.

The expectation maximization algorithm runs until μ has converged for every state. Although the algorithm has converged, the values of μ might not be the best way to model the appliance. If there are σ entries that are outside a certain prespecified bound, these entries can iteratively be lowered until the expectation maximization algorithm finds a new μ where all entries of σ are inside that bound. For fast results the expectation maximization algorithm should be initialized with μ close to the final value, the entries of σ should be large, and the entries of A should all be non-zero in order to make all transitions possible. However, it is also possible to initiate μ arbitrarily as long as the σ entries are large ($\max_t y_{1:T}$ for example)

and A has no 0 entries.

When the length of the observation sequence used grows, the joint probability distribution has more and more multiplications of probabilities. This may result in very small numbers and eventually numerical underflow. To overcome such problems all the multiplications in the formulas in this section can be transformed to summations in the log-space.

2-2-3 Training Hidden Markov Models with iterative K-means

In Section 2-2-2 it is assumed that the number of states an appliance has is known. However, in practice when an appliance is trained by using power measurements the number of states is unknown. It would therefore be more realistic if an optimal number of states is determined from the measurement data. A method to find the optimal number of states and at the same time the steady-state values is iterative K-means [10].

The iterative K-means algorithm places every measurement y_t in a certain cluster and increases the number of clusters until the standard deviation of every cluster is within a certain bound, σ_{th}, or until the maximum number of states, $K_{\max}$, which is specified in advance by the user, is reached. Assume that the measurements $y_{1:T}$ are available and we define $M = \max_{t=1,\dots,T} y_t$. A number N_{b} is defined as the number of 'bins' in which the range $[0, M]$ is divided. The bin centres are given by:

$$b_i = (i-1)\frac{M}{N_{\text{b}}} + \frac{M}{2N_{\text{b}}}, \quad i = 1, \dots, N_{\text{b}} \tag{2-20}$$

Next it is counted how frequently the measurement data can be placed in a certain bin:

$$\begin{aligned} \mathbf{1}_{B_i}(y_t) &= \begin{cases} 1 & \text{if } y_t \in \left[b_i - \frac{M}{2N_{\text{b}}}, b_i + \frac{M}{2N_{\text{b}}}\right] \\ 0 & \text{if } y_t \notin \left[b_i - \frac{M}{2N_{\text{b}}}, b_i + \frac{M}{2N_{\text{b}}}\right] \end{cases} \\ S[B_i] &= \sum_{t=1}^{T} \mathbf{1}_{B_i}(y_t) \end{aligned} \tag{2-21}$$

where $S[B_i]$ is the number of data points that is placed in bin B_i.

From this point an iteration is started over the number of clusters/states. Since every appliance has at least an ON and an OFF state, it is reasonable to start the iteration with at least $K = 2$. First the top K bins according to their value of $S[B_i]$ are selected. By the choice of N_{b} it may happen that bin centres are that close together such that data points originating from a single steady-state are placed in multiple bins. During the selection of the top K bins it could then happen that a single steady-state is represented by more than one bin. To overcome this issue, bin centres that are too close to an already selected bin are omitted. Bin centres are too close to each other if their difference is smaller than a prespecified uncertainty.

Next, a cluster is defined for every selected bin: $C_i \forall i \in \{1, 2, \dots, K\}$. For every data point the minimal distance to a bin centre of the selected K bins is determined. The data point is assigned to the cluster that corresponds to the bin with which it has minimal distance to its bin center: $y_t \in C_i$ if $|y_t - b_i| \leq |y_t - b_j|\,,\ \forall j \in \{1, 2, \dots, K\}$. Now an iteration starts where the optimal, in terms of describing the appliance data, K clusters are determined. When all data points are assigned to a cluster the mean, μ_i, of every cluster is computed.

All data points are again added to a cluster, but this time b_i is replaced by μ_i: $y_t \in C_i$ if $|y_t - \mu_i| \leq |y_t - \mu_j|\,,\ \forall j \in \{1, 2, \ldots, K\}$. Hereafter, again the means of the clusters are computed and data is assigned to clusters based on the newly computed μ_i values. This process continues until μ_i has converged for every i, hence the data points would not change between clusters in a next iteration. The μ_i values are now the optimal values for the K states to model the appliance. When converged the number of elements in every cluster is counted:

$$\begin{aligned} \mathbf{1}_{C_i}(y) &= \begin{cases} 1 & \text{if } y \in C_i \\ 0 & \text{if } y \notin C_i \end{cases} \\ S\left[C_i\right] &= \sum_{t=1}^{T} \mathbf{1}_{C_i}\left(y_t\right) \end{aligned} \tag{2-22}$$

where $S[C_i]$ is the number of data points that is placed in the cluster C_i.

It should be noted that the means of the clusters are now also given by:

$$\mu_i = \frac{1}{S\left[C_i\right]} \sum_{y_t \in C_i} y_t \tag{2-23}$$

Furthermore it is possible to compute the standard deviation of every cluster:

$$\sigma_i = \sqrt{\frac{1}{S\left[C_i\right]} \sum_{y_t \in C_i} \left(y_t - \mu_i\right)^2} \tag{2-24}$$

Now it is checked whether $\sigma_i \leq \sigma_{\text{th}} \quad \forall i$. If this is not the case and $K_{\max} > K$ at this point, the value of K is increased by 1 and the whole process is iterated with one more cluster. If it is the case that $\sigma_i \leq \sigma_{\text{th}} \quad \forall i$, the algorithm terminates. The remaining HMM system parameters can now be computed, the initial distribution vector is given by:

$$\pi_i = \frac{S\left[C_i\right]}{T} \tag{2-25}$$

and the transition matrix can be constructed as:

$$\begin{aligned} \mathbf{1}_{\mathbf{A}_{i,j}} &= \begin{cases} 1 & \text{if } \left(\mathbf{1}_{C_j}\left(y_t\right) = 1\right) \,\&\, \left(\mathbf{1}_{C_i}\left(y_{t-1}\right) = 1\right) \\ 0 & \text{otherwise} \end{cases} \\ S\left[A_{i,j}\right] &= \sum_{t=2}^{T} \mathbf{1}_{A_{i,j}} \end{aligned} \tag{2-26}$$

Now each element of the transition matrix A can be written as:

$$A_{i,j} = \frac{S\left[A_{i,j}\right]}{T-1} \tag{2-27}$$

Here $S\left[A_{i,j}\right]$ is the count of all transitions from state i to state j occurring between $t = 1$ and $t = T$. The $T-1$ term in the denominator is for normalizing such that A is again a stochastic matrix.

2-3 Disaggregation algorithms for Hidden Markov Models

2-3-1 General introduction about likelihood

Now that our HMMs are trained it is possible to use them for energy disaggregation. Briefly, energy disaggregation is the extraction of the energy consumption of individual appliances from a data stream of aggregated measurement data $y_{1:T}$. In the case of disaggregation algorithms for HMMs the Non-Intrusive Load Monitoring (NILM) problem can be interpreted as one to find the state sequence of the individual appliances such that the likelihood given the measurement data is maximized [9]. The likelihood for an HMM is defined as [10]:

$$\begin{aligned} P\left(y_{1:T}, x_{1:T}|\theta\right) &= P\left(x_{1:T}|\theta\right) P\left(y_{1:T}|x_{1:T}, \theta\right) \\ &= P\left(x_1|\theta\right) P\left(x_{2:T}|x_1, \theta\right) P\left(y_1|x_{1:T}, \theta\right) P\left(y_{2:T}|y_1, x_{1:T}, \theta\right) \\ &= P\left(x_1|\theta\right) P\left(x_2|x_1, \theta\right) P\left(x_{3:T}|x_2, x_1, \theta\right) P\left(y_1|x_1, \theta\right) P\left(y_{2:T}|x_{2:T}, \theta\right) \\ &= \dots \\ &= P\left(x_1|\theta\right) \cdot \prod_{t=2}^{T} P\left(x_t|x_{t-1}, \theta\right) \cdot \prod_{t=1}^{T} P\left(y_t|x_t, \theta\right) \end{aligned} \tag{2-28}$$

The likelihood function of an FHMM can be derived from this and is given by:

$$\begin{aligned} P\left(y_{1:T}, x_{1:T}^{(1:N)}|\theta^{(1:N)}\right) = \prod_{n=1}^{N} P\left(x_1^{(n)}|\theta^{(n)}\right) \cdot \prod_{t=2}^{T} \prod_{n=1}^{N} P\left(x_t^{(n)}|x_{t-1}^{(n)}, \theta^{(n)}\right) \\ \cdot \prod_{t=1}^{T} P\left(y_t|x_t^{(1:N)}, \theta^{(1:N)}\right) \end{aligned} \tag{2-29}$$

where the superscript $(1:N)$ means all N appliances. As mentioned previously, the continuous multiplication of probabilities may eventually lead to numerical underflow. To deal with numerical underflow issues often the log-likelihood function is used. The log-likelihood for an FHMM is given by:

$$\begin{aligned} \mathcal{L}\left(y_{1:T}, x_{1:T}^{(1:N)}|\theta^{(1:N)}\right) = \sum_{n=1}^{N} \log P\left(x_1^{(n)}|\theta^{(n)}\right) + \sum_{t=2}^{T} \sum_{n=1}^{N} \log P\left(x_t^{(n)}|x_{t-1}^{(n)}, \theta^{(n)}\right) \\ + \sum_{t=1}^{T} \log P\left(y_t|x_t^{(1:N)}, \theta^{(1:N)}\right) \end{aligned} \tag{2-30}$$

The maximization that solves the NILM problem is then given by:

$$\bar{x}_{1:T}^{(1:N)} = \underset{x_{1:T}^{(1:N)}}{\operatorname{argmax}}\, \mathcal{L}\left(y_{1:T}, x_{1:T}^{(1:N)}|\theta^{(1:N)}\right) \tag{2-31}$$

In the following sections the (sparse) Viterbi algorithm and the particle filter will be discussed to solve (2-31).

2-3-2 The Viterbi algorithm

Viterbi decoding, the basis of the Viterbi algorithm, is often used to find the most likely sequence of states given a sequence of measurements [9], [14]. If an HMM is considered (for an FHMM it works similar) and the system parameters are given by θ, then the optimal state sequence given the measurements $y_{1:T}$ is given by:

$$\begin{aligned}\bar{x}_{1:T} &= \underset{x_{1:T}}{\operatorname{argmax}} P\left(x_{1:T}|y_{1:T},\theta\right) \\ &= \underset{x_{1:T}}{\operatorname{argmax}} P\left(x_{1:T}|y_{1:T},\theta\right) P\left(y_{1:T}|\theta\right) \\ &= \underset{x_{1:T}}{\operatorname{argmax}} P\left(y_{1:T},x_{1:T}|\theta\right)\end{aligned} \tag{2-32}$$

From this point θ will be left out of the equations since θ is always known in advance. Since (2-32) is the maximization of (2-28) it can be concluded that the maximization of our likelihood maximization is the same as the maximization in (2-32). Since we are dealing with an HMM the Markov property holds. Therefore the most likely path to reach x_t is the most likely path to reach x_{t-1}, followed by the transition to x_t. The following derivation is useful for the Viterbi algorithm:

$$\begin{aligned}\max_{x_{1:t-1}} P\left(x_{1:t-1},x_t=j|y_{1:t}\right) &= \max_{x_{1:t-1}} P\left(x_{1:t-1},x_t=j|y_{1:t}\right) P\left(y_{1:t}\right) \\ &= \max_{x_{1:t-1}} P\left(x_{1:t-1},x_t=j,y_{1:t}\right)\end{aligned} \tag{2-33}$$

With this the overall most likely path of states can be tracked as follows:

$$\begin{aligned}\delta_t(j) &= \max_{x_{1:t-1}} P\left(x_{1:t-1},x_t=j|y_{1:t}\right) \\ &= \max_{x_{1:t-1}} P\left(x_{1:t-1},x_t=j,y_{1:t}\right) \\ &= \max_{x_{1:t-1}} P\left(y_t,x_t=j|x_{1:t-1},y_{1:t-1}\right) P\left(x_{1:t-1},y_{1:t-1}\right) \\ &= \max_{x_{1:t-1}} P\left(y_t,x_t=j|x_{t-1}\right) P\left(x_{1:t-1},y_{1:t-1}\right) \\ &= \max_i \left(P\left(y_t,x_t=j|x_{t-1}=i\right) \max_{x_{1:t-2}} P\left(x_{1:t-2},x_{t-1}=i,y_{1:t-1}\right)\right) \\ &= \max_i \left(P\left(x_t=j|x_{t-1}=i\right) P\left(y_t|x_t=j,x_{t-1}=i\right) \delta_{t-1}(i)\right) \\ &= P\left(y_t|x_t=j\right) \max_i \left(P\left(x_t=j|x_{t-1}=i\right) \delta_{t-1}(i)\right)\end{aligned} \tag{2-34}$$

The initial value, $\delta_t(j)$, is defined as:

$$\delta_1(j) = P\left(y_1|x_1=j\right)\pi_j \tag{2-35}$$

Remember that $P\left(y_t|x_t=j\right)$ can be determined since μ_j and σ_j are known and that the probability $P\left(x_t=j|x_{t-1}=i\right)$ is nothing more than $A_{i,j}$. For that reason it is possible to iteratively determine $\delta_t(j) \quad \forall t,j$ from (2-34).

Now the probability that at time T the appliance is in state j is given by $\delta_T(j)$. The expression $\underset{j}{\operatorname{argmax}}\, \delta_T(j)$ gives the most likely state at T. A similar thing could be done at $T-1$. $\underset{i}{\operatorname{argmax}}\, \delta_{T-1}(i)$ gives the most likely state at $T-1$, but this is not necessarily the state

from which the most likely state at T resulted. Using this approach the information at time T, which is available, is left out. This can be solved by keeping track of the most likely state one time step back if the state at the current time step is known. The most likely state x_{t-1} if $x_t = j$ is saved in $\psi_t(j)$:

$$\psi_t(j) = \underset{i}{\operatorname{argmax}} P\left(x_t = j | x_{t-1} = i\right) \delta_{t-1}(i) \tag{2-36}$$

Again $\underset{j}{\operatorname{argmax}}\, \delta_T(j)$ gives the most likely state at T, but if we know that this state is $\bar{j}$, then $\psi_T(\bar{j})$ gives the most likely state at $T-1$. In a similar way $\psi_{T-1}(\psi_T(\bar{j}))$ gives the most likely state at $T-2$. The complete most likely state sequence can be found by backtracking until $t = 1$.

The Viterbi algorithm as described above, when transformed to log-space, can directly be applied to a super-state HMM. As mentioned in Section 2-1-2 the transition matrix of a super-state HMM will take up a lot of memory when the number of appliances increases. This increase of memory is exponential with the number of appliances and therefore super-state HMMs will require a lot of memory to solve the practical NILM. The NILM problem when solved with a super-state HMM and the Viterbi algorithm has time complexity $\mathcal{O}\left(TK^N\right)$. On the other hand the Viterbi algorithm can also be used on FHMMs. Actually the same super-states need to be used, since otherwise the probability $P\left(y_t | x_t = j\right)$ cannot be evaluated. In the FHMM case every appliance has its own transition matrix and every iteration the correct columns for every appliance need to be selected. As a comparison: In the super-state HMM just a single column can be selected for all the appliances. With N appliances the problem has time complexity $\mathcal{O}\left(TNK^N\right)$. Using the Viterbi algorithm for a super-state HMM and for an FHMM gives exactly the same disaggregation results. Hence, the choice of using a super-state HMM or an FHMM is a trade-off between memory on the one hand and computational power on the other hand, but in both cases problems will arise when the number of appliances grows to a realistic household amount.

Another drawback of the Viterbi algorithm that is not yet discussed is the fact that it needs a measurement sequence in order to perform the disaggregation. The Viterbi algorithm cannot work in real time. An alternative formulation of the Viterbi algorithm solves this problem; the sparse Viterbi algorithm [14], [11]. The sparse Viterbi algorithm has two main improvements compared with the normal Viterbi algorithm. First, the algorithm is designed to find the most probable state at each instant, instead of finding the complete sequence of state at once. Secondly, it is assumed that the transition matrix of a super-state HMM is highly sparse, i.e. most elements are 0. When this matrix is stored with sparse matrix storage techniques less memory is required and all the 0 entries can be omitted during computation. Mathematically the sparse Viterbi algorithm is given by:

$$\rho_{t-1}(j) = P\left(y_{t-1} | x_{t-1} = j\right) \pi_j \tag{2-37}$$

$$\rho_t(j) = P\left(y_t | x_t = j\right) \max_i P\left(x_t = j | x_{t-1} = i\right) \rho_{t-1}(i) \tag{2-38}$$

The expression $\underset{j}{\operatorname{argmax}}\, \rho_t(j)$ then gives the optimal state at time t. The sparse Viterbi algorithm thus provides a state prediction for each measurement step and is therefore able to work in real time. No backtracking like for the normal Viterbi algorithm is required.

Using the sparse Viterbi algorithm does not only have advantages. The biggest drawback

of the sparse Viterbi algorithm is the accuracy that can be reached. The normal Viterbi algorithm is more accurate since it can use more measurements. Another drawback of the sparse Viterbi algorithm is that it is not able to deal with the numerical underflow issues. If all the transition matrices would be converted to log space, the 0 entries become $-\infty$ and using sparse matrix storage techniques would be useless.

2-3-3 Particle Filter

Another disaggregator that is often used as an algorithm for the NILM problem is the particle filter [5], [6]. A particle filter is a generally slow disaggregator, but the computation time scales linearly with the number of appliances. For that reason particle filter is particularly useful when the number of appliances is large. The idea of the particle filter is to generate and send particles through the solution space. Each time step every particle is weighted by the likelihood of the measurement. The particles with the higher weights, i.e. particles that are more likely, will propagate to the next time step, while particles with low weights will vanish.

Assume that at $t = 1$, M particles are initiated. The state of the n^{th} appliance at time t sampled by the m^{th} particle is denoted as $x_t^{(n)}[m]$. Based on the initial distribution vector every particle will be sampled, hence for states that have a high probability it is likely to be sampled more frequently. Before advancing to the next time step every particle is weighted first. A weight is calculated as:

$$w_t[m] = P\left(y_t | x_t^{(1:N)}[m]\right) = \frac{1}{\sigma_t\sqrt{2\pi}} \exp\left(-\frac{(y_t - \mu_t)^2}{2\sigma_t^2}\right) \tag{2-39}$$

where $\mu_t = \sum_{n=1}^{N} \mu_{x_t^{(n)}}$ and $\sigma_t^2 = \sum_{n=1}^{N} \sigma^2_{x_t^{(n)}}$. The total weight for normalization can be computed as

$$W_t = \sum_{m=1}^{M} w_t[m] \tag{2-40}$$

Now the normalized weights will be used for the resampling of particles before advancing to the next time step. The particles that sample a state that is more likely, get a higher weight. The corresponding states have a higher probability of being resampled, which means that these states have a higher probability of propagation to the next time step. In practice a random uniform distributed value between 0 and 1 is picked for every particle. For a certain particle m at time t call this random value $R_t[m]$ and introduce $w_t[0] = 0$. When the following applies, the particles is resampled as the state of the r^{th} particle:

$$\frac{\sum_{m=0}^{r-1} w_t[m]}{W_t} < R_t[m] \leq \frac{\sum_{m=0}^{r} w_t[m]}{W_t} \tag{2-41}$$

This is done for all M particles. Particles that have a high weight have a higher chance of being resampled than particles that have a low weight.

After resampling a state transition to the next time step can be made:

$$x_t^{(n)}[m] \sim P\left(\star | x_{t-1}^{(n)}[m]\right) \tag{2-42}$$

Here $\star$ represents all possible $x_t^{(n)}$ states. Hence, given the previous state $x_{t-1}^{(n)}[m]$ the next state of the n^{th} appliance for the m^{th} particle is sampled from the transition matrix $A^{(n)}$. Next, every particle is weighted again with (2-39) and resampling, based on this new weights, takes places before moving on to the next time step. This process continues until $t = T$.

Now the propagation of every particle is made clear, but the actual appliance states at every time step still need to be determined. There are multiple possibilities for determining the states, but in this thesis the state that has the highest number of occurrences under the M particles and an above average weight is the state selected: After the state transition to the next time step, t, and the calculation of weights only the particles for which its holds that

$$\frac{W_t}{M} < w_t[m] \tag{2-43}$$

are qualified to deliver the disaggregated state. The state that has the highest occurrence under these qualified particles becomes the disaggregated state at time t.

The particle filter can theoretically be applied to both a super-state HMM and an FHMM. The propagation of the particles by (2-42) is based on the transition matrix. Just like with the resampling a uniform distributed random variable between 0 and 1 is picked for every particle. This value is compared with the transition matrix in order to determine the next state of the particle. For a super-state HMM the entries in the transition matrix are getting smaller and smaller when the number of appliances increases. For that reason the particle filter can get stuck finding a feasible solution, i.e. a solution for which $W_t \neq 0$. When using the particle filter for an FHMM, the random uniform distributed variables are generated for every appliance separately. Every time step it takes $M \cdot N$ generations of these random variables compared to M generations in the case when a super-state HMM is used, but the chance of getting stuck in the case of using an FHMM is smaller, since the entries of the individual transition matrices are larger than the entries of the transition matrix of the super-state HMM. However, also for an FHMM the particle filter can get stuck. To prevent the algorithm of completely getting stuck at a time step a maximum number of trials is set. In this thesis, if no feasible solution is found after 100 trials, the particles are equally divided over the complete solution space. Hence, if M is chosen equal or larger than the number of super-states all super-states are selected at least once and the particle filter should be able to continue.

Since the particle filter is based on the propagation of randomly generated particles, the result of the particle filter may be different every time the algorithm is run on the same dataset. This is a disadvantage compared to the (sparse) Viterbi algorithm, where the disaggregation result is always the same for the same dataset.

2-4 Summary

Hidden Markov Models (HMMs) are introduced as probabilistic models where the Markov property holds. With these HMMs appliances can be modeled. In order to model a complete household a super-state HMM or a Factorial Hidden Markov Model (FHMM) can be used. Where the super-state HMM requires a lot of memory when the number of appliances is large, the FHMM can easily be stored in memory when the number of appliances is large.

HMMs can be trained in three different ways. The first option only applies when true state data is available. The HMM system parameters can simply be derived from the state data.

The second option is to use expectation maximization. In this case there is no true state data available, only measurement data, but the number of states is prespecified. The third option is to use iterative K-means. In this case again only measurement data is available and the optimal number of states to model the appliance with needs to be determined while training.

When the HMMs are trained they can be used to disaggregate power consumption. Two disaggregation algorithms that can work both on super-state HMMs and FHMMs are the (sparse) Viterbi algorithm and the particle filter. In the case of the Viterbi algorithm the most probable state sequence is determined given a measurement sequence. The Viterbi algorithm needs a complete sequence and is therefore unable to work real-time. The sparse Viterbi algorithm solves this issue at the cost of less accurate disaggregation results. The computational complexity of the Viterbi algorithm scales exponentially with the number of appliances and is therefore intractable when the number of appliances is large. The particle filter generates particles that semi-randomly propagate their way through the solution space. The run time of the particle filter scales linearly with the number of appliances and this method is therefore more tractable than the Viterbi algorithm when the number of appliances grows. The downside of the particle filter is that the disaggregation results are not constant given the same data.

Chapter 3

New methods and improvements on existing ones

In Chapter 2 existing disaggregation methods are presented. In this chapter improvements, in terms of accuracy and computational complexity, on these methods are introduced. At last a new optimization method for the Non-Intrusive Load Monitoring (NILM) problem is introduced.

3-1 A hybrid method between super-state Hidden Markov Models and Factorial Hidden Markov Models

In Section 2-1 super-state Hidden Markov Models (HMMs) and Factorial Hidden Markov Models (FHMMs) were introduced as models to model a household of electric appliances. The downside of the super-state Hidden Markov Model (HMM) was the amount of memory that scales exponentially with the number of appliances and the downside of the Factorial Hidden Markov Model (FHMM) was the computational complexity. In this section a hybrid method between the super-state HMM and the FHMM is proposed.

The suggestion is to use as much memory as is available for the creation of the super-state HMM. Appliances are added to the super-state HMM in ascending number of steady states. The more appliances are added to the super-state HMM the less iterations need to be done during disaggregation. When the addition of an appliance to the super-state HMM is not possible because of memory issues, the remaining appliances and the constructed super-state HMM form an FHMM.

Assume that the household consists of N appliances. The computational complexity of the Viterbi algorithm in terms of CPU time when applied on an FHMM is $\mathcal{O}\left(TNK^N\right)$ (still assuming that every appliance has K states). Assume that n out of the N appliances can be added to the super-state HMM before memory runs out and the remaining $N-n$ appliance form an FHMM together with the super-state HMM. The computational complexity of the hybrid method is then $\mathcal{O}\left(T(N-n+1)K^N\right)$. This is a slight improvement, but when the

number of appliances grows the Viterbi algorithm will still be to slow to run in real time.

3-2 Using time information for disaggregation

Up until now only the active power consumption of an appliance is considered as input for the disaggregation algorithm. In this section it is proposed to use time information as well for more accurate disaggregation results. There are two kinds of time information when considering the NILM problem. The first one is information about how long an appliance stays in a certain state or how long a cycle of an appliance takes. For appliances that unroll a predefined program, like a washing machine, the appliance cycle is the same every time turned on with that program. On the other hand there are appliances that are influenced by their environment or the consumer that can put it off at any instant, such as a television for example. For this kind of appliances it is impossible to set a predefined time reference. Furthermore, this kind of time information is already partially included in the HMMs. States that tend to last longer than others have higher transition probabilities to themselves (the diagonal entries).

The second kind of time information that can be used is information about the likeliness an appliance is used at a certain moment. For example, it is not very likely that the television is turned on at 3 a.m. or that an electric heater is used when the outside temperature is above 30°C. This suggests that our disaggregation algorithm could be improved when the model, actually the transition matrix, it uses is able to change with time. This can be done at any scale, e.g. every season a different transition matrix or every hour of the day a different transition matrix. It might even be possible to completely rule out certain appliances or certain appliance states in advance. The lights of the Christmas will not be used during the summer, hence the disaggregation problem in the summer can be simplified by not considering the Christmas lights. The drawback is that when the Christmas lights are occasionally used in the summer, they will not be recognized by the disaggregation algorithm, since the Christmas lights are not amongst the trained appliances.

3-3 Using reactive power for disaggregation

Next to active power most smart meters have the possibility to measure reactive power too. The use of reactive power is already suggested in [16], but to the best of my knowledge reactive power is never used in combination with the Viterbi algorithm.

The instantaneous power an appliance uses can be expressed as $p(t) = i(t)u(t)$, where $i(t)$ is the time-varying current and $u(t)$ is the time-varying voltage. The active power is defined as the mean value of the instantaneous power:

$$P = \frac{1}{T}\int_0^T u(t)i(t)dt \tag{3-1}$$

Active power is the power that is due to the energy consumption of an appliance. If we define U to be the rms value of the voltage and I the rms value of the current, active power, in the

case of pure harmonic supply, can also be written as

$$P = UI\cos\phi \tag{3-2}$$

where ϕ is the phase difference between the voltage and the current. On the other hand reactive power is defined as

$$Q = UI\sin\phi \tag{3-3}$$

Reactive power is maximized when the phase difference between voltage and current is 90°. When this happens, the product of voltage and current is negative for two quarters of a cycle and for the other two quarters it is equally positive. Hence, on average there flows as much energy into the appliance as out of the appliance. So there is no net transfer of energy; only reactive power flows. However, there is power loss in the resistive lines to the appliance. Reactive power can therefore be interpreted as the power due to the current that does no useful work in the appliance. To make the story complete, complex power $\mathbf{S}$ and apparent power $S = |\mathbf{S}|$ are defined as:

$$\mathbf{S} = P + jQ = UI\left(\cos\phi + j\sin\phi\right) \tag{3-4}$$

$$S = UI \tag{3-5}$$

An appliance can be of resistive, inductive, or capacitive nature depending on its components. For appliances that are purely resistive, e.g. a kettle, the reactive power component is zero. All power is characterized as active power and is measured in Watts. Appliances that also have a strong inductive component (e.g. electrical motors) or a capacitive component (e.g. laptops) have a non-zero reactive power component. The reactive power is measured in Volt-Amps-Reactive (VAR). In theory appliances that have the same active power consumption, but differ in their reactive power consumption could be differentiated based on their reactive power consumption. By only using their active power consumption this could not have been done. The inclusion of the reactive power component in the Viterbi algorithm might lead to better disaggregation results.

The next step is how to effectively add the reactive power information to the Viterbi algorithm. The first option would be to consider every possible combination of active power and reactive power of an appliance. Assume that the active power of an appliance can be modeled/clustered with 3 steady states and the reactive power can also be modeled with 3 steady states. If there is a bijection between the active power component and the reactive power component then the complete appliance can also be modeled with 3 states, but if this is not the case the appliance model could have up to $3 \cdot 3 = 9$ states. Since a higher number of states results in more computation time and will eventually make the problem intractable, this is not a good way to include the reactive power data.

Another option to include the reactive power data into the Viterbi algorithm is Principal Component Analysis (PCA) [18]. With PCA high-dimensional data can potentially be described with less dimensions, the principal components. The most important information in the data is contained in the first principal component. Perpendicular to the first principal component is the second principal component showing the second most important information. The third principal component is perpendicular to the second and the first principal component, etc. Hence, the principal components form an orthogonal set. There are as many principal components as the dimensionality of the data, but only the first few principal

components are important for describing the data. In the case of our NILM problem the dimensionality of the data is two, i.e. the active power data and the reactive power data. With PCA the most important features can be contained in the first principal component. Based on the first principal component the HMMs can be trained. This way, both data streams are contained in our model and the number of states is still comparable to the number of states when only active power is used.

In practice the PCA should be performed on the complete training dataset and not on the dataset for each individual appliance. In this way the one principal component is found that describes all the data best. Hereafter, the training data of individual appliances can be projected on this principal component and the models can be trained based on this projection. Doing PCA for every appliance separately would lead to different first principal components and this would not make any sense for the disaggregation algorithm, since it is useless to compare data that is not in the same dimension.

The PCA is nothing more than a Singular Value Decomposition (SVD) and a multiplication. Assume that the measurement data is given by

$$Y = \begin{bmatrix} Y_P & Y_Q \end{bmatrix} \tag{3-6}$$

where Y_P is a vector with the active power data and Y_Q a vector with the reactive power data. Now a SVD can be applied on Y:

$$Y = U\Sigma V^T \tag{3-7}$$

The columns of V are the principal components. The data projected on the principal components, also called the scorings, can be computed by:

$$T = -U\Sigma \tag{3-8}$$

Since we like to reduce the dimensionality from two to one, only the first principal component is picked. The first principal component is the first column of V and the scorings related to the first principal components are in the first column of T.

3-4 Transition Minimization

3-4-1 Optimization problem

In the sections above improvements on existing disaggregation algorithms were suggested. In this section a completely new disaggregation algorithm is introduced. This new disaggregation algorithm is based on the minimization of the number of state transitions of individual appliances. Assume that it is known at time step t in what state every appliance in a household is. If the sampling frequency is high enough it is most likely that at time step $t+1$ all appliances are still in exactly the same state as at t. If this is not the case, it is most probable that only one state transition occurred between t and $t+1$.

Based on the assumption that the optimal solution of our disaggregation problem is the solution where the number of state changes is minimized an optimization problem can be formulated. Assume that the measurement data $y_{1:T}$ is available, $\mathbf{x}$ is a vector with the

steady-state levels of all the appliances concatenated, and $\mathbf{a}_t$ is a binary vector indicating which entries of $\mathbf{x}$ are responsible for y_t at time step t. The non-convex integer linear optimization problem that than can be formulated is given by:

$$\begin{aligned} \underset{\mathbf{a}_t,\, t=1,\ldots,T}{\text{minimize}} \quad & \sum_{t=2}^{T} |\mathbf{a}_t - \mathbf{a}_{t-1}|\,,\ \mathbf{a}_t \in \{0,1\}^K \\ \text{subject to} \quad & |y_t - \mathbf{x}^T \mathbf{a}_t| \leq \epsilon,\ t = 1, \ldots, T. \end{aligned} \tag{3-9}$$

In this case, K is the summation of the number of states of the individual appliances. Further the sum of the entries of $\mathbf{a}_t$ equals N, the number of appliances, and for every appliance only one entry in $\mathbf{a}_t$ can be 1 and the other should be zero, i.e. an appliance can only be in one state at a time. The value of ϵ is a prespecified measure for the uncertainty between the measurements and the true states.

There are plenty of optimization methods to solve (3-9), such as, branch & bound for mixed-integer linear programming, genetic algorithms, simulated annealing, etc. One way of solving the optimization problem (3-9) that we propose here involves finding the solution space of $\mathbf{a}_t$ that satisfies the constraint, hence all feasible solutions. Once the complete feasible solution space is found a minimization of the state transitions can then be done over the feasible solution space.

The easiest way to find the feasible solution space, given the measurement y_t and the steady-state level vector $\mathbf{x}$, is to iterate over all possible realizations of $\mathbf{a}_t$, i.e. brute force optimization. Once the constraint is satisfied the specific realization of $\mathbf{a}_t$ is saved in the feasible solution space for time t. This results in a feasible solution space consisting out of one or more realizations of $\mathbf{a}_t$ for every time step. When the number of appliances is small the iteration over all possible realizations of $\mathbf{a}_t$ is still possible, but when the number of appliances grows large finding the solution space this way will become an intractable problem. In Section 3-4-3 a more tractable method for finding the feasible solution space is introduced.

When the complete feasible solution space is determined the true minimization of (3-9) can be done. In practice so-called *cost matrices* will be used to keep track of the costs of the state transitions. These matrices include the cost for all the feasible realizations for two consecutive time steps. An initial feasible realization is selected at time step $t = 1$. The feasible realization at time step $t = 2$ that has the minimal transition cost given the initial realization is selected as the solution for $t = 2$. This continues until time step $t = T$.

The downside of this minimization is that it is very local. Given a state the cheapest, in terms of the least amount of state transitions, next state is picked, but it might be beneficial to pick a state transition that is not locally the cheapest. At first such a transition would lead to higher cost, but it might be possible that a later transition is cheaper and the overall cost will be lower. To get a solution with the true minimal number of transitions a global optimization should be performed: the Global Transition Minimization (GTM).

3-4-2 Global Transition Minimization

The Global Transition Minimization, as introduced in this thesis for the first time, is still based on the cost matrices that represent the number of state transitions between two feasible realizations. Unlike the earlier explained local minimization GTM allows transitions that are

not locally minimal. The GTM keeps track of all the possible trajectories and stores them in a trajectory register. As time goes on the number of trajectories will grow exponentially and eventually the computer will run out of memory. For that reason not all possible trajectories can be considered. A threshold, C_{th}, on the cost is introduced. Define the cost of the i^{th} trajectory at time step t as $C_t(i)$. Once the total cost of a trajectory exceeds the lowest cost of any trajectory at that time plus the threshold of the cost, the trajectory is dropped from memory. This means that if

$$C_t(i) > \min_j C_t(j) + C_{\text{th}} \tag{3-10}$$

then the i^{th} trajectory will not be considered any more in the next time step and it will be cleared out of the trajectory register. With C_{th} the amount the cost is allowed to rise above the minimum cost can be controlled. As a consequence, the number of trajectories that need to be stored will be higher when C_{th} is larger.

Eventually the optimal trajectory, i.e. the trajectory with minimal cost, needs to be determined from the trajectory register. This can easily be done at $t = T$ as

$$\bar{j} = \operatorname*{argmin}_j C_T(j) \tag{3-11}$$

where $\bar{j}$ is the optimal trajectory. However, it is beneficial to clear up the trajectory register every time possible so the computer will not run out of memory and the algorithm can run faster. It is possible to clear up the register of possible trajectories once only a single feasible realization is in the trajectory register for a time step. Assume that the GTM removed all trajectories with higher cost than the minimum plus C_{th} according to (3-10) at time step t. It might now be the case that for time step $t - \tau$ for any $\tau \geq 0$ only one feasible solution is present in the trajectory register. In this case the trajectory in the register that has minimal cost up to $t - \tau$ is stored to the final trajectory and only the trajectories starting from $t - \tau$ need to be considered from now on. The GTM can be considered as a kind of dynamic programming, because the problem of finding the complete optimal trajectory is split into parts without loss of optimality. The parts are not determined in advance, but during the optimization itself.

3-4-3 Determination the solution space

In theory the GTM in Section 3-4-2 will find the optimal trajectory, but as mentioned in Section 3-4-1 finding the feasible solution space will be an intractable problem when the number of appliances is large. It is simply not possible to iterate over all possible realizations of $\mathbf{a}_t$ for every t. Hence, only part of all possible realizations can be considered and within these realizations the feasible solution must be present, otherwise the optimal trajectory cannot be found.

Contrary to the GTM algorithm where the feasible solution space is determined before starting with the minimization, the feasible solution space is now determined simultaneously with the minimization. As mentioned in Section 3-4-1 it is most likely that there are no state changes between time step t and time step $t+1$ if the sampling frequency is high enough. For that reason a maximum number of transitions is introduced, $S_{\max}$. Given an initial realization

of $\mathbf{a}_t$ only the realizations that have $S_{\max}$ or less transitions from $\mathbf{a}_t$ are considered for $\mathbf{a}_{t+1}$. Assume that a household has $N = 30$ appliances with each $K = 2$ states. The total number of realizations is $2^{30} = 1.0737 \cdot 10^9$. If $S_{\max} = 3$ and the initial realization is known, there are

$$\binom{N}{S_{\max}} = \frac{N!}{S_{\max}!(N - S_{\max})!} = \frac{30!}{3!(30-3)!} = 4060 \tag{3-12}$$

combinations of appliances that change their state. Every appliance has 2 states and every combination consists of changing $S_{\max} = 3$ appliances: $2^3 \cdot 4060 = 32480$ realizations will be considered. This is far more tractable than when all possible realizations would be considered.

The only problem at this point is the availability of the initial value. Since the minimization is global and runs in parallel with the search for the feasible solution space, it is not possible to select the true realization $\mathbf{a}_t$ as the initial value in the search for feasible realizations at $t+1$, since the true realization $\mathbf{a}_t$ is still unknown at t. However, it is possible to come up with an initial value as follows. The optimal trajectory up until t can be computed. The realization at t of that optimal trajectory acts as initial value for finding the feasible solution space at $t + 1$. Since the initial value picked in this way might not be the value of the final optimal trajectory, it is required to select $S_{\max}$ a little higher than the maximum number of transitions you would expect at a time step. Otherwise it is possible that the true state is not amongst the feasible solutions. In the case that no feasible solution is selected, because all solutions that are checked for feasibility lie outside ϵ from the total power consumption, the solution with the minimal distance to the total power consumption is marked as the only feasible solution for that time step.

3-5 Conclusions

A hybrid method between super-states Hidden Markov Models and Factorial Hidden Markov Models is introduced. This hybrid method gives a slight improvement in computational complexity, but when the number of appliances grows the disaggregation will still be intractable.

Time information and reactive power data is proposed as extra data for the disaggregation algorithm. The use of reactive power data will potentially only contribute in the accuracy of the disaggregation results, while the use of time information can also contribute in the reduction of the computational complexity.

At last a new optimization method, Global Transition Minimization, to solve the NILM problem is introduced. This optimization methods minimizes the number of state transitions between two successive time steps within a feasible solution space.

Chapter 4

Synthetic case study

In Chapter 2 and Chapter 3 various disaggregation methods have been introduced. In this chapter these disaggregation algorithms will be qualitatively compared in terms of accuracy and computation time required. Before using real-world data in Chapter 5, synthetic data is used first in this chapter.

4-1 Setup

Before our disaggregation algorithms will be tested on real-world data a synthetic dataset is generated for testing. Data for a single household with six fictional appliances is generated. These fictional appliances try to simulate a real-world counterpart: a cooker, a television, a washing machine, a microwave, a charger, and an amplifier. The appliances have different numbers of steady states and the steady states have different power levels. Table 4-1 shows the steady-state levels for every modeled appliance. Of course a measurement on an appliance will not always lead to an exact steady-state value. For that reason an uncertainty percentage is introduced on every steady-state level. Including an uncertainty in this way resembles the real-life situation of measurement noise. The total power consumption is a summation of the power consumption of the individual appliances. The various power levels in Table 4-1 are chosen in such a way that it is far from obvious that the total power consumption is caused by a specific appliance combination.

Table 4-1: Steady-state power levels of six synthetic appliances in W

Cooker	Charger	Television	Microwave	Washing machine	Amplifier
0	0	0	0	0	0
1500	80	1	300	100	100
		200	500	500	200
				1000	300
					400

4-2 Data creation

The appliances as described in Section 4-1 should generate some data on which the disaggregation algorithms can be tested. The synthetic data is generated by a MATLAB script. For all appliances the parameters of the Hidden Markov Models (HMMs) are specified. The transition matrices are the most important system parameters and are shown in Figure 4-1. As an example take the transition matrix of the cooker. If the state of the cooker is 0 W at t the state of the cooker at $t+1$ is also 0 W with a probability of 0.9995 and 1500 W with a probability of 0.0005. The rows of all the transition matrices correspond to the steady-state levels shown in Table 4-1.

$$\begin{bmatrix} 0.9995 & 0.0005 \\ 0.002 & 0.998 \end{bmatrix}$$

transition matrix of cooker

$$\begin{bmatrix} 0.9995 & 0.0005 \\ 0.002 & 0.998 \end{bmatrix}$$

transition matrix of charger

$$\begin{bmatrix} 0.98 & 0.02 & 0 \\ 0.0001 & 0.995 & 0.0049 \\ 0.0002 & 0.002 & 0.9978 \end{bmatrix}$$

transition matrix of television

$$\begin{bmatrix} 0.99 & 0.003 & 0.007 \\ 0.0299 & 0.97 & 0.0001 \\ 0.0299 & 0.0001 & 0.97 \end{bmatrix}$$

transition matrix of microwave

$$\begin{bmatrix} 0.999 & 0 & 0 & 0.001 \\ 0.001 & 0.999 & 0 & 0 \\ 0 & 0.005 & 0.99 & 0.005 \\ 0 & 0 & 0.01 & 0.99 \end{bmatrix}$$

transition matrix of washing machine

$$\begin{bmatrix} 0.996 & 0.001 & 0.001 & 0.001 & 0.001 \\ 0.007 & 0.99 & 0.001 & 0.001 & 0.001 \\ 0.007 & 0.001 & 0.99 & 0.001 & 0.001 \\ 0.007 & 0.001 & 0.001 & 0.99 & 0.001 \\ 0.007 & 0.001 & 0.001 & 0.001 & 0.99 \end{bmatrix}$$

transition matrix of amplifier

Figure 4-1: Transition matrices of the artificial appliances

With the initial distribution vector, the transition matrix, and the steady-state levels available for each appliance it is possible to generate synthetic data. Assume that T time samples of data need to be created. For every appliance T uniform random numbers between 0 and 1 are picked. This number is compared with the cumulative row summation of the transition matrix or in case of $t = 1$ with the cumulative summation of the initial distribution vector. As an example the cumulative row summation of the transition matrix of the amplifier is shown in Figure 4-2. Assume at a certain time step t the amplifier consumes 200 W; hence the third row of the cumulative transition matrix is under consideration. Let us define the uniform random number that is picked for the amplifier at $t+1$ as R. If $R \leq 0.0070$ the amplifier makes a state transition to the 0 W state, if the random number satisfies $0.0070 < R \leq 0.0080$ the amplifier makes a state transition to the 100 W state, and if $0.0080 < R \leq 0.9980$ the amplifier remains in the 200 W state, etc. With the comparison of

uniform random numbers with the entries of the rows of the cumulative transition matrices the synthetic dataset is generated.

$$\begin{bmatrix} 0.996 & 0.997 & 0.998 & 0.999 & 1.000 \\ 0.007 & 0.997 & 0.998 & 0.999 & 1.000 \\ 0.007 & 0.008 & 0.998 & 0.999 & 1.000 \\ 0.007 & 0.008 & 0.009 & 0.999 & 1.000 \\ 0.007 & 0.008 & 0.009 & 0.010 & 1.000 \end{bmatrix}$$

Figure 4-2: Cumulative transition matrix for the amplifier

The last step is to add measurement noise to the steady-state levels. In Section 4-1 it is already mentioned that the measurement noise is within a percentage of the steady-state value. The noise is chosen to be normally distributed, which is also a realistic assumption. Hence, the uncertainty percentage of the steady state must be converted to the normal distribution. For this the properties of the normal distribution can be used. It is known that with a probability of about 99.7% a data point falls within $\pm 3\sigma$ from the mean value. Hence, to get 99.7% of the measurement noise covered by the normal distribution, the steady-state value multiplied by the uncertainty percentage should be equal to 3σ. That is

$$3\sigma = p \cdot \mu \quad \Rightarrow \quad \sigma = \frac{p}{3} \cdot \mu \tag{4-1}$$

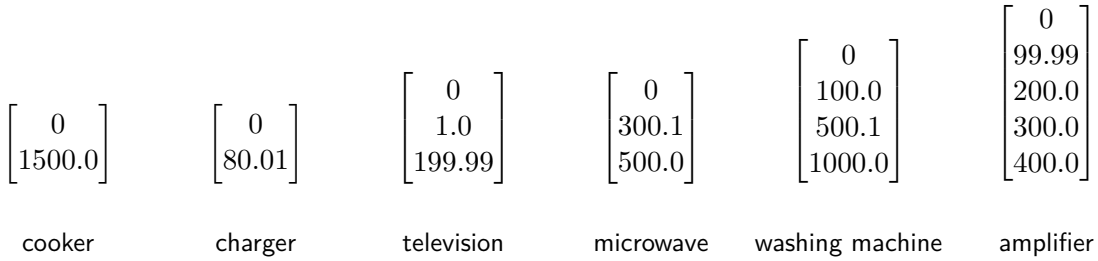
here p is the uncertainty percentage (with for example 5% uncertainty $p = 0.05$) and μ the steady-state value. Hence, by using μ as the steady-state values as shown in Table 4-1 and σ from (4-1) the values generated by $\mathcal{N}(\mu, \sigma^2)$ have a probability of 99.7% of being within the uncertainty percentage from μ.

4-3 Model training

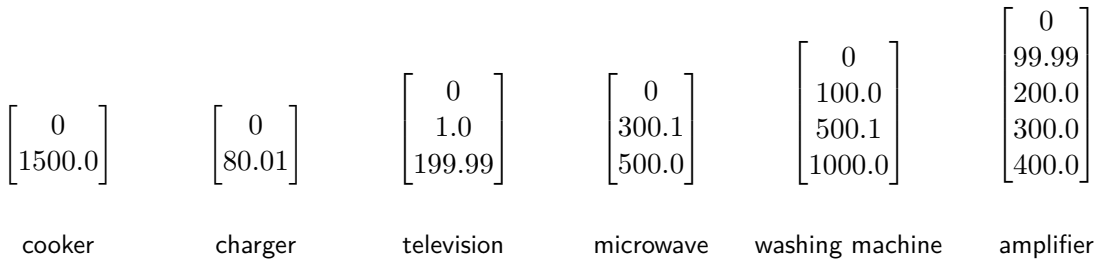
In the section above it is explained how data is generated, but with this data it is not possible to run the disaggregation algorithm right away. Models of the individual appliances need to be trained. In the line of the rest of this thesis HMMs will be used as models for the appliances. In Section 2-2 two useful methods for training HMMs were presented: expectation maximization and iterative K-means. In this section these two methods will be compared by training HMMs based on the synthetic data.

Synthetic data for all six appliances will be generated. The measurement uncertainty is 5%. It is assumed that the data is 1 Hz and that a single day of data will be sufficient; therefore 86400 samples of data will be used for every appliance. The expectation maximization algorithm requires that the number of states K is prespecified. The number of states of every appliance, as can be derived from Table 4-1, will be given to the expectation maximization algorithm while the iterative K-means algorithm starts with $K = 2$. Further we select $K_{\max} = 5$ and $N_{\mathrm{b}} = 1000$ for the iterative K-means algorithm. Both algorithms output all the HMM system parameters. The steady states found by both algorithms are shown in Figure 4-3. There are no differences between the results of both algorithms and when

compared with Table 4-1 the results are correct. Another factor by which the algorithms can be compared is the required run time. Table 4-2 shows the run times of both algorithms for training the different appliances. Both algorithms take more time when the number of states grows, but for the expectation maximization algorithm the relation between run time and number of states is more of an exponential character whereas for the iterative K-means algorithm it has a linear character. Already for a two-state appliance the iterative K-means algorithm is about 4 times faster than the expectation maximization algorithm. Hence, the fact that iterative K-means needs to iterate over the number of states does not make it slower compared to expectation maximization.



(a) Expectation Maximization



(b) iterative K-means

Figure 4-3: Steady-state values found by the training algorithms

Table 4-2: Run time of both algorithms for training the appliances in seconds

	Expectation Maximization	iterative K-means
cooker	7.05	1.74
charger	6.92	1.67
television	14.29	2.53
microwave	36.02	2.49
washing machine	49.48	3.51
amplifier	112.96	4.78

4-4 Disaggregation

4-4-1 General intro

When all appliances are trained, power data can be disaggregated. In real-world datasets there might be an offset between the aggregated power measurement and the summation of the power readings of the individual appliances. Possible reasons for this offset are the loss in the household wiring between the smart meter and the appliances, measurement noises, and too inaccurate measurements. There are methods to remove this offset, but since in this chapter synthetic data is considered the smart meter measurements are just a summation of the measurements of the individual appliances:

$$y_t = \sum_{n=1}^{N} y_t^{(n)} \tag{4-2}$$

21600 samples, which resembles 6 hours of data, will be created with 5% uncertainty as described in Section 4-2. The aggregated power signal, which is thus a summation of the power signal of the individual appliances, is shown in Figure 4-4. The aggregated data will be disaggregated by the disaggregation algorithms and the results will be compared with the ground truth data generated by the data creator.

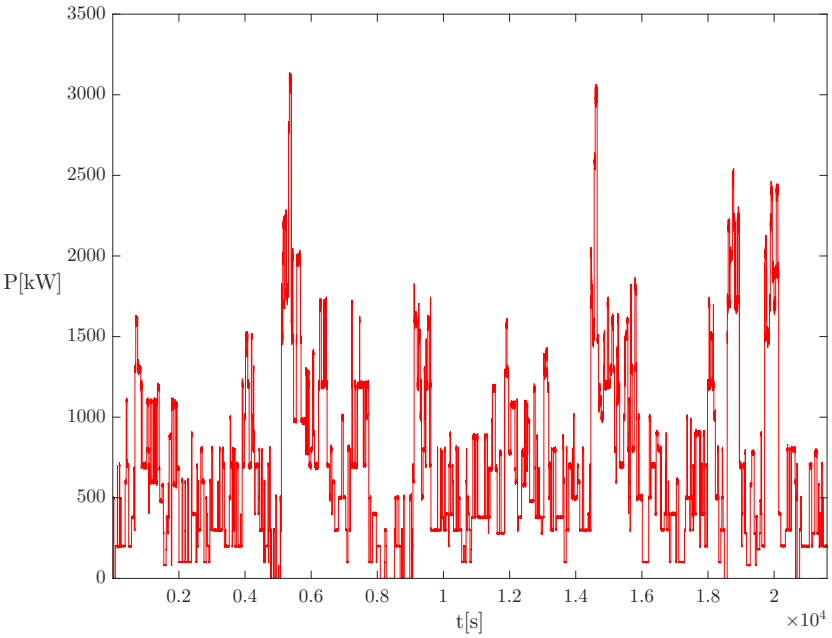


Figure 4-4: Aggregated synthetic power data of the six appliances

For a disaggregation algorithm two elements are important: accuracy and run time. If an algorithm is accurate, but it take ages to get the results it is useless. On the other hand if an algorithm is fast, but the results are inaccurate it is useless too. In Section 4-4-2 the accuracy of all proposed algorithms will be compared. In Section 4-4-3 the run time of the proposed algorithms will be compared.

4-4-2 Accuracy

To effectively compare the accuracy of the algorithms a measure for accuracy required. In [12] the estimation accuracy is defined as

$$\left(1 - \frac{\sum_{t=1}^{T}\sum_{n=1}^{N}\left|\hat{y}_t^{(n)} - y_t^{(n)}\right|}{2 \cdot \sum_{t=1}^{T}\sum_{n=1}^{N} y_t^{(n)}}\right) \cdot 100\% \tag{4-3}$$

where $\hat{y}_t^{(n)}$ is the estimated power consumption by the disaggregation algorithm of appliance n at time step t and $y_t^{(n)}$ is the ground truth data. If the result of (4-3) is 100% every appliance is disaggregated completely correctly. The factor 2 is introduced in the denominator since power that is assigned to the wrong appliance is counted twice as an error. Assume that there are two appliances and at a certain time step appliance 1 consumes 100 W and appliance 2 consumes 0 W. As the disaggregation algorithm assigns 50 W of power to both of them an error of 50 W is made for both appliances. This is a total error of 100 W. However, in reality only 50 W of power is assigned wrong. When the summation over the appliances is omitted it is also possible to compare the accuracy of the disaggregation algorithm for different appliances:

$$\left(1 - \frac{\sum_{t=1}^{T}\left|\hat{y}_t^{(n)} - y_t^{(n)}\right|}{2 \cdot \sum_{t=1}^{T} y_t^{(n)}}\right) \cdot 100\% \tag{4-4}$$

The accuracy of the Viterbi based disaggregation algorithms is compared using (4-3). The results are shown in Table 4-3. The maximum accuracy that is possible to reach on this dataset is 99.33%, due to the noise present. This is computed by plugging in the true state data of the appliances in (4-3). Hybrid Viterbi is the hybrid method between super-state HMMs and Factorial Hidden Markov Models (FHMMs) as described in Section 3-1. In this experiment the cooker, charger, television, microwave, and washing machine are added to a super-state Hidden Markov Model (HMM) and this super-state HMM forms a Factorial Hidden Markov Model (FHMM) together with the amplifier. From Table 4-3 it can be seen that the accuracy for the Viterbi algorithm when applied to a super-state HMM, an FHMM, or a hybrid model is exactly the same. This is in accordance with the theory that these algorithms lead to exactly the same disaggregation results. The accuracy of the sparse Viterbi algorithm is much lower than the normal Viterbi algorithm as would be expected.

Table 4-3: Accuracy of various Viterbi based disaggregation algorithms on the synthetic dataset

	Accuracy from (4-3)
Viterbi super-state HMM	94.88%
Viterbi FHMM	94.88%
hybrid Viterbi	94.88%
sparse Viterbi super-state HMM	64.30%
sparse Viterbi FHMM	64.30%

The number of super-states is $2 \cdot 2 \cdot 3 \cdot 3 \cdot 4 \cdot 5 = 720$. To measure the influence of the number of particles on the disaggregation accuracy the accuracy is determined for different numbers of particles, varying from 1 till 10 times the number of super-states. For the particle filter for the super-state HMM and the particle filter for the FHMM the results are shown in Table 4-4. The results for the super-state HMM and the FHMM are comparable and there is on average only a slight decrease in accuracy with the decrease of the number of particles.

Table 4-4: Accuracy based on (4-3), the percentages shown are the means of 5 runs

Particles	Particle Filter super-state HMM	Particle Filter FHMM
720	84.45%	82.22%
1440	86.53%	85.36%
2880	88.85%	89.31%
4320	90.70%	90.84%
5760	89.29%	90.25%
7200	91.42%	91.31%

At last the accuracy of the Global Transition Minimization (GTM) needs to be checked. The first scenario would be with the threshold on the cost equal to zero, $C_{\text{th}} = 0$, and the maximum number of transitions equal to six, $S_{\max} = 6$. Recall that every trajectory with cost higher than the cost of the current minimum is dropped and every possible state change is considered. However, for this scenario an extraordinary amount of memory is required and even a high-performance cluster with access to 4TB of RAM is not able to complete the disaggregation. This might directly seem to be the end of the GTM as disaggregator. However, the problem lies in the steady-state levels of the trained appliances. The television has two steady states, 0 W and 1 W, that are so close together that any total power consumption that is not approximately 0 or 1 W can be caused by either of these two states. This results in the storage of both states in the solution space for every time step and eventually this results in memory overflow. This problem is solved when the television is trained as a two-state appliance, see Figure 4-5:

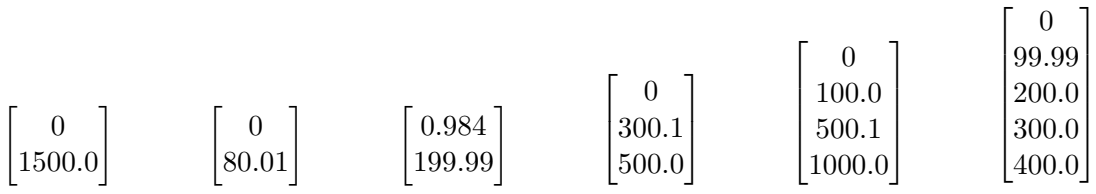


Figure 4-5: Steady-state values when the television is a modeled as a two-state appliance.

An error of 0.984 W is made every time step the television is in the 0 W state, but this is a small error compared to the noise values of the states of other appliances. With this new model for the television the GTM with $C_{\text{th}} = 0$ and $S_{\max} = 6$ reaches an accuracy of 78.56%. However, still it is not possible to increase the value of C_{th} due to memory issues. On the other hand it is possible to lower the value of $S_{\max}$. The results are shown in Table 4-5.

A striking result is the fact that the accuracies for $S_{\max} = 5$ and for $S_{\max} = 4$ are higher than for $S_{\max} = 6$. The only conclusion that can be made based on these results is that the selection of the feasible solution space when $S_{\max} = 6$ is worse than when $S_{\max} = 5$; when $S_{\max} = 6$ too many feasible solutions are selected. This suggests that the selection of the feasible solution space could be improved for more accurate results. Obviously the accuracy drops when $S_{\max}$ drops lower than 4 since the true solution might not be selected.

Table 4-5: Accuracy of the GTM for various values of $S_{\max}$ with the models of Figure 4-5

	Accuracy from (4-3)
$S_{\max} = 6$	78.56%
$S_{\max} = 5$	81.46%
$S_{\max} = 4$	79.87%
$S_{\max} = 3$	73.15%
$S_{\max} = 2$	69.53%
$S_{\max} = 1$	66.54%

To get insight in the influence of C_{th} on the disaggregation results the dataset is changed. Without the change the solution space would be too big and memory problem would arise. Therefore, only for this part, the television and also the amplifier are completely omitted from the dataset, resulting in a synthetic dataset with only 4 appliances, see Figure 4-6. The disaggregation results for this dataset are shown in Table 4-6 for various values of C_{th} and $S_{\max}$. The influence of C_{th} is minimal and due to the memory issues when raising the value of C_{th}, it is not beneficial to raise the value of C_{th} .

$$\begin{bmatrix} 0 \\ 1500.0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 80.01 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 300.1 \\ 500.0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 100.0 \\ 500.1 \\ 1000.0 \end{bmatrix}$$

cooker charger microwave washing machine

Figure 4-6: Steady-state values of the appliances used for the results in Table 4-6.

Table 4-6: Accuracy of the GTM for various values of C_{th} and $S_{\max}$ for the synthetic dataset with 4 appliances

	$C_{\text{th}} = 0$	$C_{\text{th}} = 2$
$S_{\max} = 4$	92.79%	92.98%
$S_{\max} = 3$	92.79%	92.98%
$S_{\max} = 2$	84.55%	84.58%
$S_{\max} = 1$	80.28%	80.28%

4-4-3 Run time

Next the run time of the disaggregation algorithms is compared. All run time experiments in this thesis are performed on a laptop with an Intel® Core™ i7-2630QM CPU @ 2.00 GHz. The synthetic dataset with 6 appliances modeled as in Figure 4-3 is used. The run times corresponding to the accuracies in Table 4-3 are shown in Table 4-7. The bad time performance of the Viterbi algorithm on an FHMM is shown, since the disaggregation takes about 71 times the amount of time the Viterbi algorithm does for a super-state HMM. The hybrid method performs a lot better than the Viterbi for FHMMs, but is still about 43 times slower than the Viterbi for super-state HMMs.

A striking result is the performance of the sparse Viterbi algorithm. For both the super-state HMM and the FHMM the run time is slower in the case of the sparse Viterbi algorithm. Unlike the claims in [14] and [11] that the transition matrices are highly sparse and sparse matrix storage techniques are beneficial, the results for the synthetic dataset are different. The transition matrix of the super-state HMM takes 3.955 Mb and when this transition matrix is stored with sparse matrix storage technique is takes 3.961 Mb. Although the sparse Viterbi algorithm does omit the zeros in the transition matrix, it takes more time to look for the non-zero entries in the sparse transition matrix. The conclusion is that the sparse Viterbi algorithm is not able to improve the run time of the Viterbi algorithm.

Table 4-7: Run times of various Viterbi based disaggregation algorithms on the synthetic dataset

	Run time (s)
Viterbi super-state HMM	143.05
Viterbi FHMM	10169.69
hybrid Viterbi	6192.91
sparse Viterbi super-state HMM	195.45
sparse Viterbi FHMM	10338.72

The run times of the particle filter for both the super-state HMM and the FHMM are shown in Table 4-8. These run times are corresponding to the accuracies shown in Table 4-4. The particle filter for the super-state HMM is slightly faster than the particle filter for the FHMM, which can be explained by the fact that the particle filter for the FHMM needs to iterate over all appliance. However, the storage of the transition matrix of the super-state model will lead to memory overflow when the number of appliances grows. For 10 appliances with each having 3 states the transition matrix of the super-state model in double precision requires

$$\frac{(3^{10})^2 \cdot 8}{1024^3} = 25.98 \text{ Gb}$$

of memory. Therefore, the particle filter for the FHMM gets the preference over the particle filter for the super-state HMM.

Table 4-8: Run times of the particle filter, the run times shown are the means of 5 runs

Particles	Run time (s) super-state HMM	Run time (s) FHMM
720	788.15	822.47
1440	1553.4	1658.3
2880	3308.9	3984.3
4320	5683.3	6698.6
5760	7973.9	9217.2
7200	10466.0	11874.0

The comparison of the run times of the GTM with the other run times is not straightforward. The results of Table 4-5 are obtained on a high-performance cluster, which was necessary due to memory requirements. The high-performance cluster has a lower clock speed than the laptop the other algorithms were run on and also the influence of processes of other users would influence the run time of the disaggregation algorithms. This makes it not possible to quantitatively compare the run times of the disaggregation approaches in Table 4-5. However, it is possible to run the disaggregations from Table 4-6 on the laptop. The run time results are shown in Table 4-9. A striking result is that the run time for both C_{th} values is longer for $S_{\text{max}} = 3$ than for $S_{\text{max}} = 4$. Since the accuracy for both values of S_{max} is equal, see Table 4-6, it is unlikely that this is caused by the selection of a different feasible solution space. The longer run time could be explained in the way the solution space is found. Equation (3-12) gives 1 when $N = 4$ and $S_{\text{max}} = 4$, but 4 when $N = 4$ and $S_{\text{max}} = 3$. For $S_{\text{max}} = 3$ the algorithm has to change appliances in 4 different combinations, which takes more time than only change a single combination when $S_{\text{max}} = 4$. The run time difference between $C_{\text{th}} = 0$ and $C_{\text{th}} = 2$ is caused by the extra number of trajectories that need to be stored and computed for $C_{\text{th}} = 2$.

Table 4-9: Run times in seconds of the GTM for various values of C_{th} and S_{max} for the synthetic dataset with 4 appliances

	$C_{\text{th}} = 0$	$C_{\text{th}} = 2$
$S_{\text{max}} = 4$	42.77	69.31
$S_{\text{max}} = 3$	54.27	79.09
$S_{\text{max}} = 2$	39.62	52.22
$S_{\text{max}} = 1$	12.35	12.08

At this point the run times of all disaggregation algorithms can be compared. The Viterbi algorithm for super-state HMMs has the best performance by far, but it will not be able to deal with a regular household of appliances since that would required to much memory to store the transition matrix. The particle filter for the FHMM is only slower than the Viterbi for the FHMM when 7200 particles are used. From Section 4-4-2 it is known that the accuracy with 2880 particles is almost the same as with 7200 particles, but with 2880 particles the particle filter is about 2.5 faster than the Viterbi for FHMMs. The conclusion is that the Viterbi algorithm is outperformed by the particle filter.

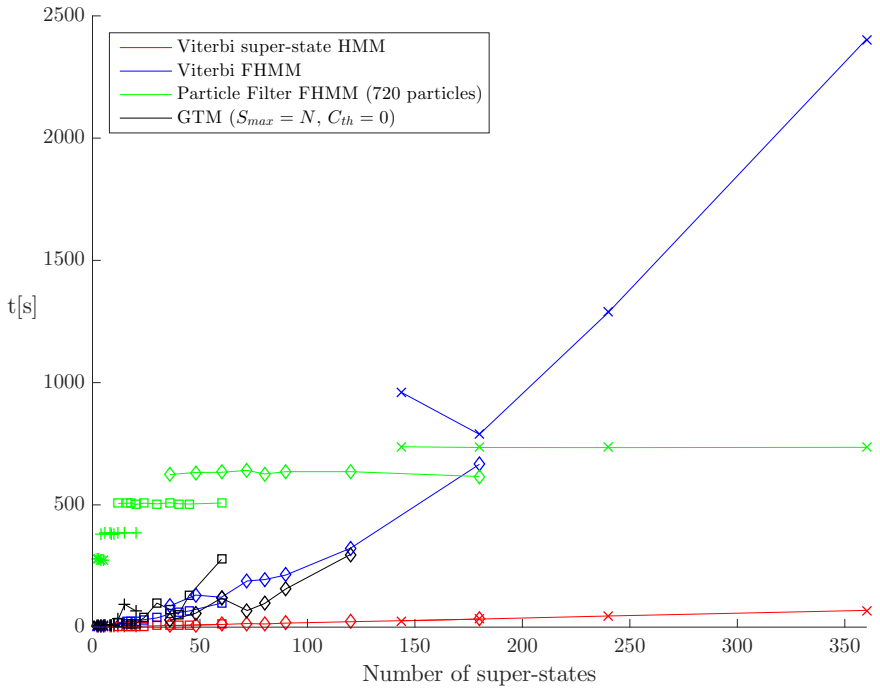
For the GTM the results are promising, but the results are obtained on a dataset with only 4 appliances consisting of $2 \cdot 2 \cdot 3 \cdot 4 = 48$ super-states. It is not possible to say anything useful

in terms of run time at this point in comparison with the Viterbi algorithm or the particle filter. A quantitative run time comparison can only be done when the number of appliances and the number of super-states is equal for the various algorithms. For that reason the run time of the four most important algorithms is compared for various numbers of appliances and various numbers of super-states. Any total number of super-states that can be formed by the product of the individual states of the appliances from Figure 4-3 is evaluated. For example if only 1 appliance is considered there is the possibility to have 2, 3, 4, or 5 super-states based on the numbers of states in Figure 4-3. The power signals of the appliances not under consideration are subtracted from the aggregated power signal and the disaggregation is done with the resulting signal. The run times results are shown in Figure 4-7. When the number of super-states is formed by 1 appliance the $*$ sign is used, the $+$ sign for 2 appliances, the $\square$ sign for 3 appliances, the $\lozenge$ sign for 4 appliances, and the $\times$ sign for 5 appliances. 6 appliances resulted in only a single realization of the number of super-states (720 super-states) and the results did not give any additional information for the run time comparison. Therefore, the result for 6 appliances is not shown in Figure 4-7.

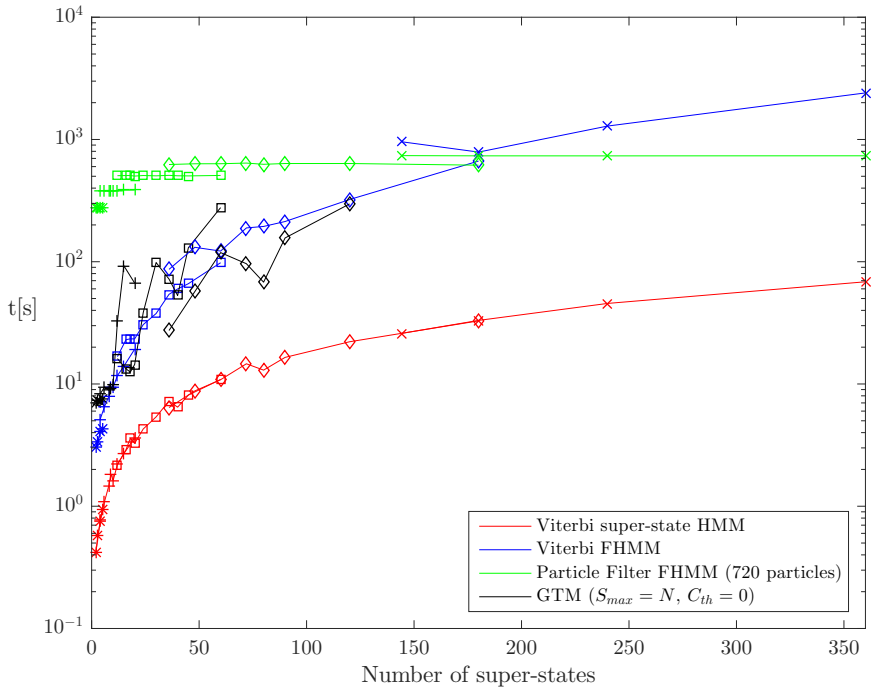
From Figure 4-7 it is easily seen that the Viterbi algorithm for the super-state HMM is the fastest algorithm and that the Viterbi algorithm for the FHMM is the slowest algorithm when the number of super-states grows. However, like said before, the Viterbi algorithm for the super-state HMM will eventually lead to memory issues; hence it is not practicable in real-life situations.

The particle filter with 720 particles is the slowest algorithm when the number of super-states is low, but from about 180 super-states the particle filter becomes faster than the Viterbi algorithm for the FHMM. Another striking result is that the run time of the particle filter is only determined by the number of appliances and not by the number of super-states; for every number of appliances the run time stays approximately the same for every number of super-states in Figure 4-7. Although the run time is constant with an increasing number of super-states, the accuracy of the disaggregation might drop.

The GTM with $S_{\max} = N$ and $C_{\text{th}} = 0$ is not able to disaggregate a power signal with more than 4 appliances and 120 super-states. Despite this deficiency the GTM cannot be ruled out as practical NILM disaggregator. The performance of the GTM highly depends on the composition of the appliances. If the feasible solution space for the GTM consists of more than one possible solution for too many consecutive time steps and the trajectory register cannot be reduced, eventually memory overflow will occur or large memory usage will lead to exceptional long run times. Our dataset is created in such a way that most of the time steps multiple solutions are possible and for that reason memory cannot be reduced that often. For a real-world dataset this might be different and the GTM might perform well in that case. From what can be seen in Figure 4-7 the GTM is faster than the particle filter in the cases where the disaggregation can be done with the GTM and there is no clear pattern in the graph of the GTM as there is for the other algorithms.



(a) Normal plot



(b) Logarithmic plot

Figure 4-7: Run time of various disaggregation algorithms versus number of super-states. The $*$ sign is used for a 1 appliance dataset, the $+$ sign for 2 appliances, the $\square$ sign for 3 appliances, the $\Diamond$ sign for 4 appliances, and the $\times$ sign for 5 appliances.

4-5 Improving Non-Intrusive Load Monitoring (NILM) with time information

In Section 3-2 it is suggested to use time information as an improvement for the disaggregation algorithms. The likeliness of an appliance being turned on in a specific time frame could be used to create time-dependent transition matrices for the HMM. In case of this thesis the 6 appliances from the previous sections generate a dataset, but this time the probability of an appliance being in use is dependent on the time of the day, see Table 4-10. The appliances have the same power levels as earlier in this chapter, accept for the microwave which is split into two separated appliances. The reason for that is that the amount of power the microwave is used on is dependent on the time of day. From this table it can be seen that for example the charger is used with a probability of 0.7 during night time and with a probability of 0.1 for each of the other day parts.

Table 4-10: Probalities of an appliance being turned on in a certain time span

	00:00 - 06:00	06:00 - 12:00	12:00 - 18:00	18:00 - 24:00
Cooker	0.01	0.14	0.65	0.2
Charger	0.7	0.1	0.1	0.1
Television	0.05	0.1	0.25	0.6
Microwave (300W)	0.01	0.5	0.45	0.04
Microwave (500W)	0.01	0.04	0.45	0.5
Washing machine	0.6	0.2	0.05	0.15
Amplifier	0.01	0.33	0.33	0.33

2592000 samples of training data, which is equal to 30 days, were created. Every appliance (considering the microwave as two different appliances) is being used 10 times a day, where the probabilities from Table 4-10 determine in which day part the appliance operates. Based on this training data the appliances are trained with iterative K-means for each day part; hence every appliance has a transition matrix for every day part. Next, test data is created for 86400 samples, which is equal to 1 day. This test data is divided into the 4 day parts and every part is disaggregated by the super-state HMM Viterbi algorithm with the trained transition matrices corresponding to that day part. The reason that only a single experiment with the super-state HMM Viterbi algorithm is done, is that the super-state HMM Viterbi algorithm was the most accurate and fastest algorithm in the sections above. Further the Viterbi algorithm is deterministic, i.e. the result is always the same for a certain dataset. Any improvements by using time information in combination with this dataset should be noticed with this single experiment. The accuracy given by (4-3) is 99.34%; only noise is not disaggregated.

For comparison the appliances are also trained on the same training dataset with a single transition matrix for each appliance. These trained models were used by the disaggregation of the complete test dataset with the super-state Viterbi algorithm. The result is again an accuracy of 99.34%, concluding that the use of time information does not give any improvements for this dataset.

4-6 Conclusions

Synthetic training and test datasets with 6 appliances and a total of 720 super-states is generated. Normally distributed noise with a maximum of approximately 5% of the steady-state value is added to the steady-state values.

For training HMMs the iterative K-means algorithm and the expectation maximization algorithm performed equally in terms of accuracy, but the iterative K-means algorithm was faster for every appliance.

The Viterbi algorithm and particle filter were tested for both super-state HMMs and for FHMMs. In terms of accuracy the Viterbi algorithm is better, but it will be intractable in terms of memory requirements or run time when the number of appliances grows. The sparse Viterbi algorithm does not improve these issues for the Viterbi algorithm. The Global Transition Minimization is less accurate than the particle filter for FHMM, but when the GTM is able to get disaggregation results, it is always faster than the particle filter. However, the GTM has memory issues for the synthetic dataset when the number of super-states is over 120. Based on the synthetic dataset the particle filter is the best disaggregation algorithm, but in Chapter 5 real-world data will be used for further testing.

The use of time information is tested by modeling every appliance for every part of the day separately. This, however, did not result in a higher accuracy for the synthetic dataset.

Chapter 5

Real-life case study

In Chapter 4 synthetic data was used to test disaggregation algorithms. In this chapter the Dutch Residential Energy Dataset (DRED) [15] is used for testing. Also the influence of reactive power as extra data input will be considered.

5-1 Setup and model training

The DRED is the first open-access dataset with energy consumption data of a household from the Netherlands. For a period of 6 months electricity, occupancy data, and ambient parameters have been measured in a household. For 12 appliances the power consumption is measured at a rate of 1 Hz and the aggregated power consumption is measured by a smart meter, also at a rate of 1 Hz. The data is not preprocessed and therefore there is an offset between the aggregated power signal and the summation of the power signals of the individual appliances.

The first 135 days of the DRED, which corresponds to 11664000 samples for each appliance, are selected as training data. The next 15 days of the dataset, which corresponds to 1296000 samples, are selected as test data. For all 12 appliances Hidden Markov Models (HMMs) are trained with K-means, where the value of K is prespecified. The trained appliances for K reaching from 2 to 5 are tested for accuracy individually, i.e. according to (4-4). The Viterbi algorithm for super-state HMMs is used as disaggragator on the test data of individual appliances, since it is the most accurate and fastest algorithm discussed. Hence, a trained Hidden Markov Model (HMM) and the 15 days of test data are given as input to the Viterbi algorithm. With the disaggregation result the accuracy can be computed according to (4-4). The results for various values of K are shown in Table 5-1. The percentages in Table 5-1 show how well the appliances can be modeled by a certain number of states. Where the fridge can be modeled with 2 states and with an accuracy of more than 95%, the unknown appliance and the sockets need at least 5 states to get an accuracy just above 80%. Based on the accuracies in Table 5-1 a suitable number of states to model an appliance can be selected. Since a total accuracy, see (4-3), of at least 80% is desired, the accuracies of the individual

appliances should not be much lower than 80% and it should be kept in mind that the total accuracy can never be higher than the highest accuracy of an individual appliance.

Table 5-1: Accuracies according to (4-4) where the appliances are trained with K-means on the training dataset and tested with Viterbi for super-state HMMs with the test dataset

	$K=2$	$K=3$	$K=4$	$K=5$
television	73.09%	88.02%	98.36%	98.99%
fan	87.96%	93.5%	95.94%	96.77%
fridge	95.01%	96.18%	97.42%	97.76%
laptop computer	49.33%	82.87%	87.6%	89.25%
electric heating element	92.98%	95.25%	95.29%	96.25%
oven	88.65%	92.1%	96.37%	97.21%
unknown	-7.51%	52.97%	61.97%	80.89%
washing machine	56.61%	90.36%	90.84%	93.18%
microwave	82.67%	89.19%	94.56%	95.9%
toaster	56.56%	84.53%	91.96%	95.06%
sockets	77.01%	72.23%	78.4%	81.86%
cooker	94.54%	97.81%	98.65%	98.98%

5-2 Disaggregation

In Chapter 4 it was concluded that the Viterbi algorithm for a Factorial Hidden Markov Model (FHMM) is outperformed by other algorithms and that the performance of the particle filter for super-state HMMs is comparable with the particle filter for Factorial Hidden Markov Models (FHMMs). For that reason only the particle filter for FHMMs and the Global Transition Minimization (GTM) will be considered here. As a first experiment all appliances from Table 5-1 with an accuracy of at least 90% are selected. This means that the television is modeled with 4 states, the fan with 3 states, the fridge with 2 states, etc. The laptop computer, the unknown device, and the sockets are not considered in this experiment since they need more than 5 states to get an accuracy of at least 90%. Hence, 9 appliances with a total of 13824 super-states will be considered.

Originally the aggregated power signal is noisy and preprocessing is required, but since not all appliances are considered the aggregated power signal cannot be used. For this experiment the aggregated power signal is the summation of the individual appliance power signals:

$$y_t = \sum_{n=1}^{N} y_t^{(n)} \tag{5-1}$$

Hence, the aggregated power signal is an artificial signal created by the summation of the power signals of the 9 selected appliances only. The complete aggregated test signal consists of 15 days of data. Testing with the particle filter and the GTM with this dataset would take various days. However, it is possible to construct the transition matrix for the Viterbi algorithm for super-state HMMs, but it costs far too much memory to store the most likely state vector ϕ for every time step. For that reason a section of 3 days, from the beginning of day 4 till the end of day 6, is selected from the 15 days test set. During these days all 9

appliances are turned on. The Viterbi algorithm can still not be run due to memory issues, but the particle filter and the GTM are able to give results. To keep the run time relatively short, the particle filter is run with 720 particles. The GTM is run with $C_{\text{th}} = 0$ and $S_{\text{max}} = 3$, also to keep the run time short and with $S_{\text{max}} = 3$ three appliances can change state every time step. It is a reasonable assumption that no more than 3 appliances change state during the same second. The results are shown in Table 5-2. For both algorithms the accuracy is much lower than the minimal required 80%. A run for the GTM with $S_{\text{max}} = 4$ does not give improvements compared to $S_{\text{max}} = 3$ and the run time is about 4 times longer than the particle filter. The conclusion is that the GTM is outperformed in terms of accuracy and run time by the particle filter.

Since the particle filter still runs faster than real time, the number of particles can be increased in order to improve the accuracy. The number of super-state is 13824, hence it is not surprising that the accuracy is not that high with only 720 particles. 10000 particles is about the maximum to let the algorithm run in real time. Since the algorithm runs almost at real time, it was not able to disaggregated 3 complete days. The first 12 hours of the considered 3 days are disaggregated with the particle filter with 10000 particles. An accuracy of 33.24% is achieved and with a run time of 42451 seconds. The conclusion is that the particle filter is also not able to perform an accurate disaggregation. It is surprising that the particle filter with 10000 particles is far less accurate than the particle filter with 720 particles. When the particle filter with 720 particles is used on the 12 hour dataset an accuracy of 77.14% is achieved. This is a striking result, since with almost 14 times less particles and a run time of 2190 seconds, which is about 19 times faster, an accuracy that is more than twice as high is achieved. Remember that for the synthetic dataset there is a general trend that the accuracy is higher when the number of particles is higher, see Table 4-4. For this part of the DRED this is not the case. An explanation for the decreasing accuracy with the increasing number of particles could possibly be found in the higher number of state combinations that is considered. There might be state combinations that are unlikely to be reached with 720 particles, but that will be considered when 10000 particles are used. When the weights of these particles is also high, an unlikely state transition might happen, i.e. given the current measurement it is likely, but given the last state it is unlikely.

It is beyond the time frame of this thesis to perform additional experiments on the complete dataset or with the real aggregated signal, instead of the summation of the appliances. Further, it may be assumed that the extra states and noise would lead to even worse results than the results in the experiment above. In a following research the relation between the number of super-states and the number of particles of the particle filter might be considered. Probably there is an optimal number of particles, such that the accuracy is maximized.

Table 5-2: Disaggregation results for the particle filter and the GTM on 3 days of the DRED

	Accuracy from (4-3)	Run time (s)
particle filter FHMM, 720 particles	64.60%	12948
GTM, $C_{\text{th}} = 0$ and $S_{\text{max}} = 3$	29.42%	11946
GTM, $C_{\text{th}} = 0$ and $S_{\text{max}} = 4$	29.24%	50026

5-3 Using reactive power as extra input

Active and reactive power measurements of a Miele washing machine and a Miele clothes dryer were obtained; both with a frequency of 1 Hz, see Figure 5-2. A scatter plot, where the reactive power is plotted against the active power, is shown for both appliances in Figure 5-1. The data points of both appliances in Figure 5-1 are not completely clustered separately. This cannot be fixed with Principal Component Analysis (PCA), but the reactive power component can be combined with the active power component. The result of the PCA is shown in Figure 5-3, where the values on first principal component are plotted against the values on the second principal component.

With the active and reactive power data from Figure 5-2 30 days of training data and 10 days of test data is created where the washing machine and clothes dryer are used both 6 times a day. In this way three data streams are generated: active power data, reactive power data, and data along the first principal component of all the data. On every data stream the appliances are trained with K-means where $K = 5$. The Viterbi algorithm for super-state HMMs is used as disaggregator. The accuracy of the disaggregation for the different data streams is shown in Table 5-3. The disaggregation accuracy for the first principal component is lower than when only active or reactive power is used; concluding that the use of reactive power data does not improve disaggregation results in this case.

Table 5-3: Accuracy results of the test dataset for active power, reactive power, and the first principal component data

	Accuracy from (4-3)
active power	96.96%
reactive power	96.62%
first principal component	95.25%

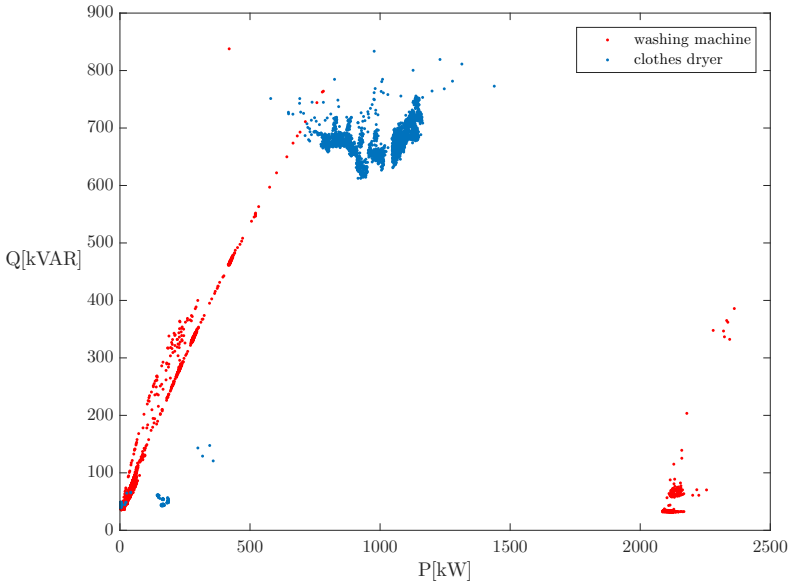
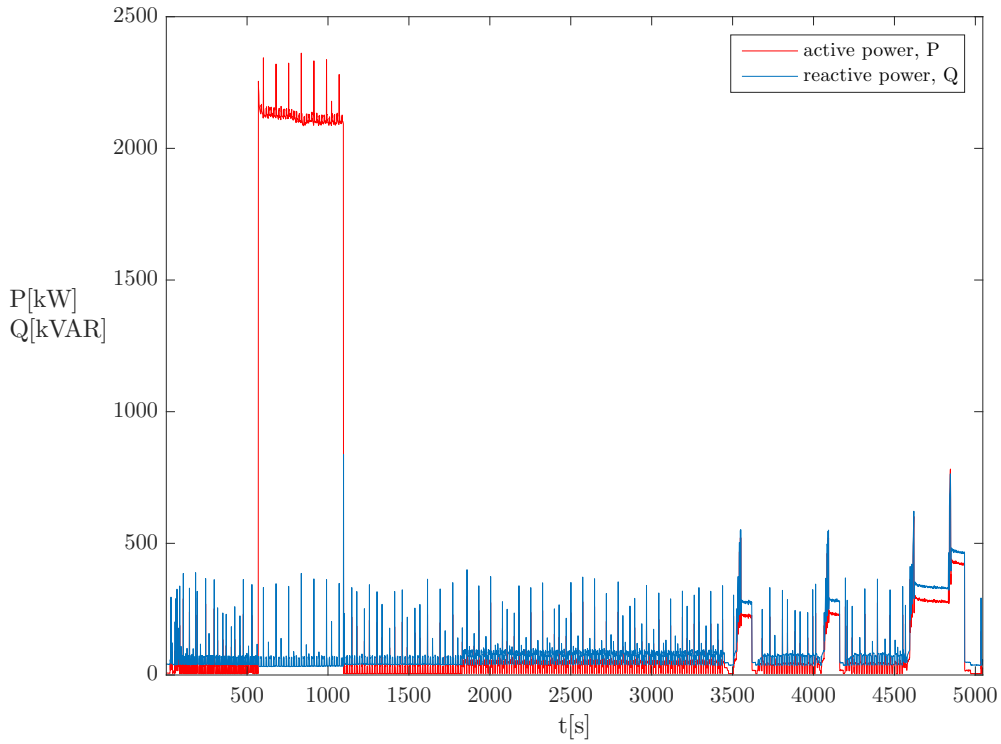
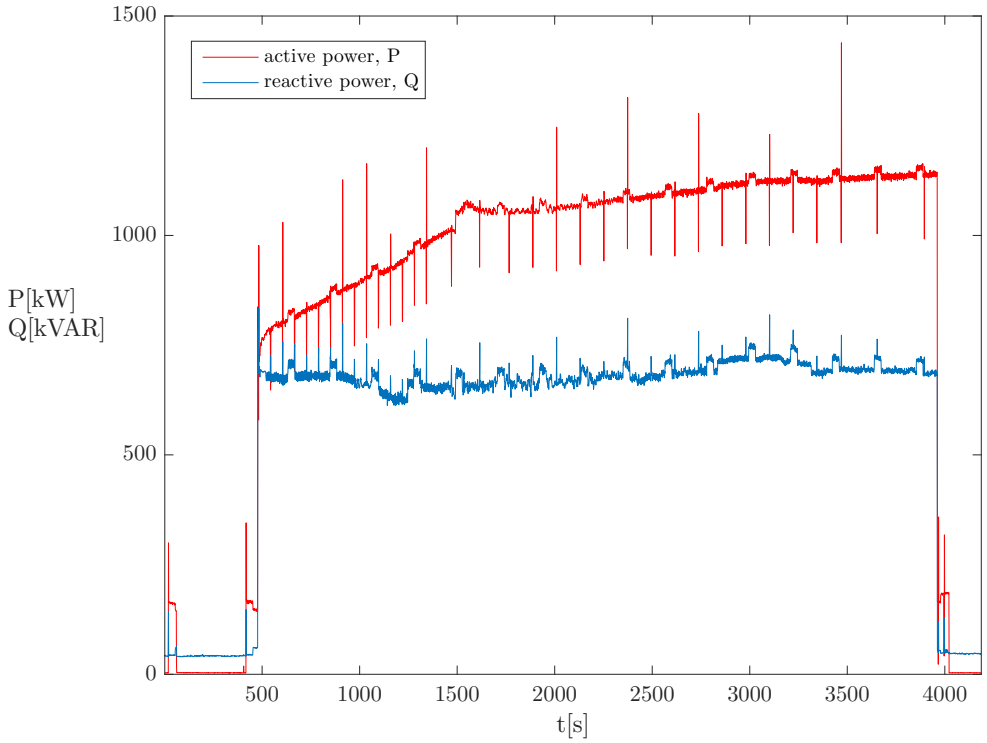


Figure 5-1: Scatter plot; reactive power vs. active power



(a) washing machine



(b) clothes dryer

Figure 5-2: Active and reactive power of a Miele washing machine and a Miele clothes dryer

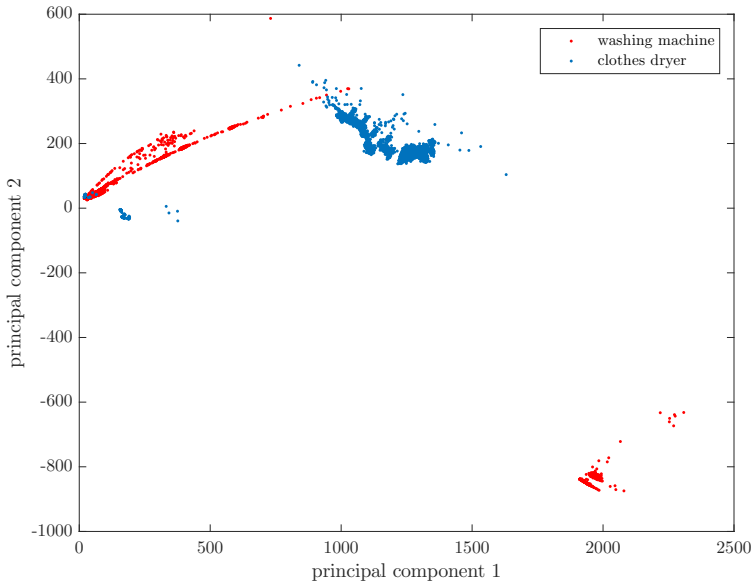


Figure 5-3: Scatter plot; the first principal component vs. the second principal component

5-4 Conclusions

The Dutch Residential Energy Dataset (DRED) is an open-access dataset from the Netherlands consisting of a household with 12 appliances. This dataset is used to test the performance of the particle filter and the GTM on a real-life dataset. The first 135 days of data are used to train the appliances. Only 9 appliances achieve an accuracy of at least 90% when modeled with 5 states or less. These 9 appliances are used to disaggregate 3 days of data with the particle filter and the GTM. The particle filter with 720 particles had a better performance in terms of accuracy (64.60%) and in terms of run time (12948 seconds) than the GTM; hence, the GTM is outperformed by the particle filter. An attempt to improve the accuracy of the particle filter by increasing the number of particles to 10000 has been done. Since the particle filter with 10000 particles runs almost in real time, only the first 12 hours of the 3 days of data are disaggregated. The result is an accuracy of 33.24%. When the particle filter with 720 particles is used on this same 12 dataset an accuracy of 77.14% is achieved, concluding that for a real-world dataset an increase in the number of particles does not always lead to a higher accuracy.

The use of reactive power as extra input is investigated by using active and reactive power data from a washing machine and a clothes dryer. The conclusion is that the use of reactive power does not lead to a higher accuracy for this washing machine and this clothes dryer. The accuracy is even slightly smaller than when using only active power data. The use of reactive power as extra input does not seem to be a way for improving the disaggregation accuracy.

The reason for the overall poor performance of the particle filter and the GTM on the DRED can be explained by the fact that the appliances trained for disaggregation were modeled with an accuracy of just above 90%. In the synthetic case the appliances were modeled with an accuracy over 99%. Improving the training accuracy would require more states, which results in a more complex disaggregation problem. At this point it cannot be con-

cluded that any disaggregation method in this thesis can lead to a practical solution for the NILM problem. In Chapter 6 improvements and new research directions will be discussed.

Chapter 6

Discussion, conclusion, and future work

6-1 Discussion

This thesis basically treated three algorithms: the Viterbi algorithm, the particle filter, and the Global Transition Minimization (GTM). The Viterbi algorithm could be applied on a super-state Hidden Markov Model (HMM) or on a Factorial Hidden Markov Model (FHMM). Although the Viterbi algorithm achieved the highest accuracy on the synthetic dataset, it is not an algorithm that can be applied on real-world data: in the case of the super-state HMM the Viterbi algorithm requires too much memory and in the case of the FHMM the Viterbi algorithm requires too much computation time. Implementing the sparse Viterbi algorithm does not lead to a better performance. At last the Viterbi algorithm is not able to run in real time due to the backtracking step.

The results of the particle filter on the synthetic dataset are good in terms of accuracy and from about 180 super-states the particle filter is faster than the Viterbi algorithm applied on an FHMM. Further the run time of the particle filter scales about linearly with the number of appliances. On the real-world Dutch Residential Energy Dataset (DRED) the performance of the particle filter is poor. Although appliances are modeled with an accuracy of at least 90%, the particle filter reaches no more than 65% accuracy and increasing the number of particles such that it can run just in real time will not improve the results.

The performance of the GTM on the synthetic dataset is quite variable. When the GTM does not run into memory problems, the accuracy results are acceptable (above 80% for 6 appliances) and the run time is shorter than the run time of the particle filter. On the other hand the GTM was not able to disaggregate a problem with more than 120 super-states. On the DRED there are no memory issues, but the accuracy is very low and the run time is longer than the run time of the particle filter.

As concluded in Chapter 5 the poor performance of the particle filter and the GTM on the DRED can be explained as follows. First of all, the appliances trained for disaggregation were modeled with an accuracy of just above 90%. In the synthetic case the appliances were

modeled with an accuracy over 99%. The real appliances of the DRED are hard to model with a number of discrete states. There are no instantaneous jumps between states, but it might take multiple seconds before a state transition is made. The intermediate data points are easily assigned to another appliance. Especially in the case of the GTM, which is based on the thought that it is unlikely that a lot of appliances change state between two consecutive time steps, it is a real problem, since the true appliance might not be selected into the solution space while wrong appliances will. Second, the experiment on the DRED included 9 appliances and a total of 13824 super-states where the synthetic experiment only included 6 appliances and 720 super-states. To differentiate between 13824 super-states takes more time and is more complex than to differentiate between 720 super-states. This will lead to less accurate results for the DRED. Further, only 9 appliances were included in the experiment in Section 5-2. A real household would typically include at least two times that number of appliances leading to even longer run times and less accuracy.

From above it can be concluded that it is difficult, if not impossible, to model real-world appliances with steady-state values for accurate disaggregation results. For that reason it might be useful to drop the first criterion from Section 1-2, i.e. that the signatures of the appliances may only be based on the active power sampled at 1 Hz. The consequence is that non-steady-state methods are qualified again, but the conventional smart meters cannot be used for these methods. Another solution might be a different modeling framework than the HMMs, where the appliances can be modeled more accurately and the disaggregation problem is still tractable.

6-2 Conclusion

The problem statement given at the beginning of this thesis is: Which methods might contribute to a practical solution of the Non-Intrusive Load Monitoring (NILM) problem? Because of the first criteria that the sampling frequency should not be higher than 1 Hz only steady-state algorithms are qualified as practical. The Viterbi algorithm for super-state HMMs is not practical, because of its memory issues for a reasonable number of appliances in a household. The Viterbi algorithm for FHMM is not practical either, because the run time will be far too long for a reasonable number of appliances. The GTM is shown to be too inaccurate when it comes to a real-world problem. Only the particle filter gives promising accuracy and run-time results for a real-world problem.

The conclusion that can be made is that of all the algorithms discussed, the particle filter is the only disaggregation algorithm that might lead to a practical solution for the NILM problem. However, this result is obtained from a single real-world dataset with only 9 appliances. Future work is required to increase the accuracy of the particle filter and the number of appliances it can work with.

6-3 Future work

In Chapter 5 it became clear that the particle filter is more accurate with 720 particles than with 10000 particle when part of the real-world DRED was disaggregated. For a short-

term research the optimal number of particles, given the number of super-states, could be researched such that the disaggregation accuracy is maximized.

When more time is available, new modeling methods that replace the HMM framework could be researched. These models should be able to describe the non-instantaneous state transitions, such that real-world appliances can be modeled with low-frequency data. A modeling framework that is a suitable candidate should combine the information of the steady-state power levels with the non-instantaneous state transitions. The duration of the state transitions might be an important signature for the differentiation of appliances. With the new modeling framework possibly new disaggregation algorithms need to be designed.

In the long term, disaggregation methods where the low-frequency requirement on the measurement data is dropped could be researched. Another possibility is to adopt a research in the field of pattern recognition. Based on measurement patterns and a database of prespecified patterns a machine learning algorithm should recognize appliances from the aggregated data stream. A combination of pattern recognition and optimization might improve the disaggregation results even when only low-frequency data is considered.

Bibliography

[1] https://www.forbes.com/sites/energyinnovation/2017/05/10/americas-renewable-electricity-forecast-grows-to-2050-even-under-trump/#5dcafe5516e4.

[2] https://ec.europa.eu/energy/en/topics/markets-and-consumers/smart-grids-and-meters.

[3] COMMISSION STAFF WORKING DOCUMENT Country fiches for electricity smart metering Accompanying the document Report from the Commission Benchmarking smart metering deployment in the EU-27 with a focus on electricity /* SWD/2014/0188 final */.

[4] R. Bonfigli, S. Squartini, M. Fagiani, and F. Piazza. Unsupervised algorithms for non-intrusive load monitoring: An up-to-date overview. In *2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC)*, pages 1175–1180, June 2015.

[5] D. Egarter, V. P. Bhuvana, and W. Elmenreich. PALDi: Online Load Disaggregation via Particle Filtering. *IEEE Transactions on Instrumentation and Measurement*, 64(2):467–477, Feb 2015.

[6] D. Egarter and W. Elmenreich. Autonomous load disaggregation approach based on active power measurements. *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 293–298, March 2015.

[7] G. W. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, Dec 1992.

[8] W. Kong, Z. Y. Dong, and D. J. Hill. A Hierarchical Hidden Markov Model Framework for Home Appliance Modelling. *IEEE Transactions on Smart Grid*, 2016.

[9] W. Kong, Z. Y. Dong, D. J. Hill, F. Luo, and Y. Xu. Improving Nonintrusive Load Monitoring Efficiency via a Hybrid Programing Method. *IEEE Transactions on Industrial Informatics*, 12(6):2148–2157, Dec 2016.

[10] W. Kong, Z. Y. Dong, J. Ma, D. Hill, J. Zhao, and F. Luo. An Extensible Approach for Non-Intrusive Load Disaggregation with Smart Meter Data. *IEEE Transactions on Smart Grid*, 2016.

[11] S. Makonin, F. Popowich, I. V. Bajić, B. Gill, and L. Bartram. Exploiting HMM Sparsity to Perform Online Real-Time Nonintrusive Load Monitoring. *IEEE Transactions on Smart Grid*, 7(6):2575–2585, Nov 2016.

[12] Stephen Makonin and Fred Popowich. Nonintrusive load monitoring (NILM) performance evaluation. *Energy Efficiency*, 8(4):809–814, Jul 2015.

[13] Masako Matsumoto, Yu Fujimoto, and Yasuhiro Hayashi. Energy Disaggregation Based on Semi-Binary NMF. *Machine Learning and Data Mining in Pattern Recognition: 12th International Conference, MLDM 2016, New York, NY, USA, July 16-21, 2016, Proceedings*, pages 401–414, 2016.

[14] J. A. Mueller and J. W. Kimball. Accurate Energy Use Estimation for Nonintrusive Load Monitoring in Systems of Known Devices. *IEEE Transactions on Smart Grid*, 2016.

[15] Akshay S.N. Uttama Nambi, Antonio Reyes Lua, and Venkatesha R. Prasad. LocED: Location-aware Energy Disaggregation Framework. *Proceedings of the 2Nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, pages 45–54, 2015.

[16] M. Weiss, A. Helfenstein, F. Mattern, and T. Staake. Leveraging smart meter data to recognize home appliances. *2012 IEEE International Conference on Pervasive Computing and Communications*, pages 190–197, March 2012.

[17] W.H.C. Janssen. Energy disaggregation: Nonintrusive load monitoring; The search for practical methods. 2017.

[18] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

Glossary

List of Acronyms

NILM	Non-Intrusive Load Monitoring
HMM	Hidden Markov Model
HMMs	Hidden Markov Models
NMF	Non-negative Matrix Factorization
FHMM	Factorial Hidden Markov Model
FHMMs	Factorial Hidden Markov Models
PCA	Principal Component Analysis
SVD	Singular Value Decomposition
GTM	Global Transition Minimization
DRED	Dutch Residential Energy Dataset