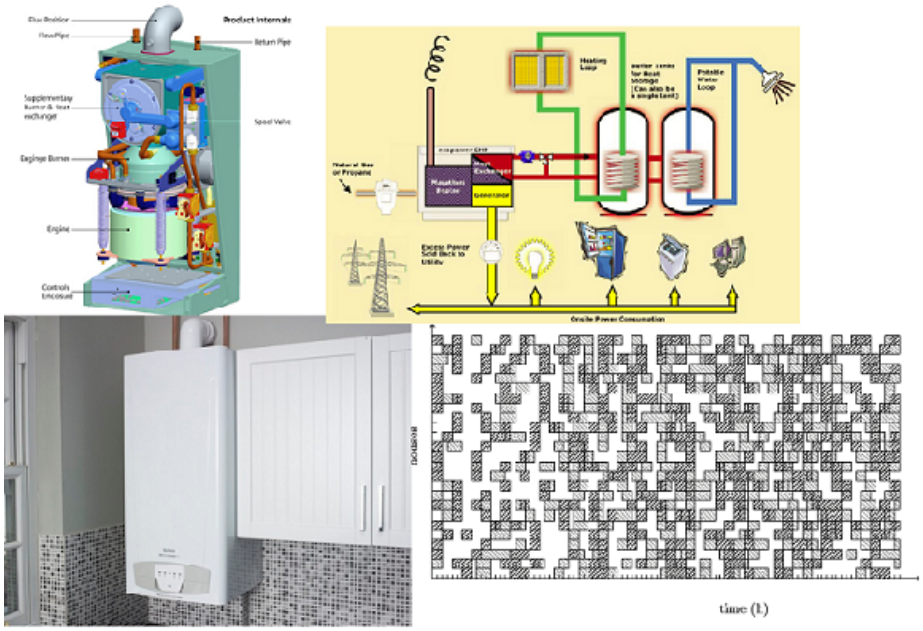
# An approximate dynamic programming approach to the micro-CHP scheduling problem

Maarten Vinke

August 24, 2012

Supervisors: dr. M.G.C. Bosman, prof. dr. J.L. Hurink




UNIVERSITEIT TWENTE.

# Abstract

Due to environmental issues such as the greenhouse effect, and the fact that the earth's oil and gas reserves are slowly depleting, the electricity supply chain is slowly transforming toward novel methods of energy generation. One of these methods consists of using micro-CHPs in households to satisfy part of the electricity demand. A micro-CHP is an installation that simultaneously generates heat and electricity, replacing the traditional boiler. In this setting, the electricity production is essentially a by-product of the heat production, so that there is no heat loss during the electricity production process. The micro-CHP comes with a heat buffer, in which hot water can be stored, so there is some flexibility in the time for this production.

Still, as electricity is dependent on the heat demand, the electricity generation from a single house can become erratic. Therefore, in this thesis we consider a group of houses, for which the goal is to obtain a more or less constant electricity production. To enforce this, we assume that there are fixed upper and lower bounds on the total electricity production. The goal in this thesis is to find a production schedule for the different micro-CHPs, so that both these total electricity bounds and the houses' individual heat demands are satisfied. Within these constraints, the objective is to maximize the revenue gained by selling this electricity, whereby these electricity prices are time-dependent, with peaks during the hours when electricity demand is higher.

This micro-CHP problem has already been investigated by Bosman [4], where multiple heuristics were used to find such a schedule, using a discretized time scale. In this thesis, we have attempted to solve the scheduling problem mentioned above using the technique of Approximate Dynamic Programming (ADP). For this the problem was first modelled as a Dynamic Program, which was too large to solve exactly.

After this technique is introduced by considering the taxicab problem, it is used on the actual micro-CHP problem. As a decision here consists of determining which micro-CHPs are turned on and off in the following time interval, often the number of possible decisions is too large to consider them all. Therefore, first the decision space is reduced by using a strict priority list. Then an approximation function for every state is defined, which uses a weighted sum of basis functions. These basis functions are numerical values based on certain features of a state. Then, the approximation function and the reduced decision space can be used to find to find paths through the state space, each resulting in a production schedule for the micro-CHPs. After such a schedule has been found, the values found in this schedule are used to update the weights in the approximation function, to increase the quality of the approximation. This is repeated for multiple iterations.

This algorithm is applied to a data set, after which the results were compared to those of Bosman [4]. The results are generally better than his results from the local search heuristic, and comparable with those of the column generation method. Only in the cases where the planning intervals were so small that the production behaviour of the micro-CHP had to be taken into account, the ADP algorithm did not perform as well.

# Dankwoord

Dit afstudeerwerk had ik uiteraard nooit alleen kunnen volbrengen. Daarom wil ik allereerst mijn beide begeleiders bedanken. Maurice Bosman heeft mij vooral geholpen met het invoeren van de data, het inwerken in het onderwerp en met mijn verslag. Uiteraard heb ik ook heel veel gehad aan de informatie en data uit zijn afstudeerscriptie, en de daaraan voorafgaande papers. Johann Hurink wil ik vooral bedanken voor de hulp met het schrijven van mijn verslag. Regelmatig heeft hij mijn verslag doorgeploegd, en meer dan eens stonden daarna op sommige pagina's meer aantekeningen dan originele tekst. Dit zag er altijd wat deprimerend uit, maar uiteindelijk is het mijn verslag wel enorm ten goede gekomen. Daarnaast wil ik ook Jan-Kees van Ommeren bedanken voor het plaatsnemen in mijn afstudeercommissie.

Verder wil ik de mensen binnen DMMP bedanken voor de gezellige sfeer waarin ik mijn onderzoek heb kunnen doen. De lunches en de koffiepauzes waren altijd gezellig, en maakten het werk een stuk minder zwaar en eentonig. Daarnaast wil ik ook mijn vrienden en familie bedanken voor de steun en de getoonde interesse, die ook regelmatig tot nieuwe inzichten hebben geleid. In het bijzonder moet ik hierbij mijn kamergenoot Mathijs ter Braak noemen, die mij altijd geduldig aanhoorde als ik weer eens vastliep, en mij ook regelmatig geholpen heeft met problemen met LaTeX.

# Contents

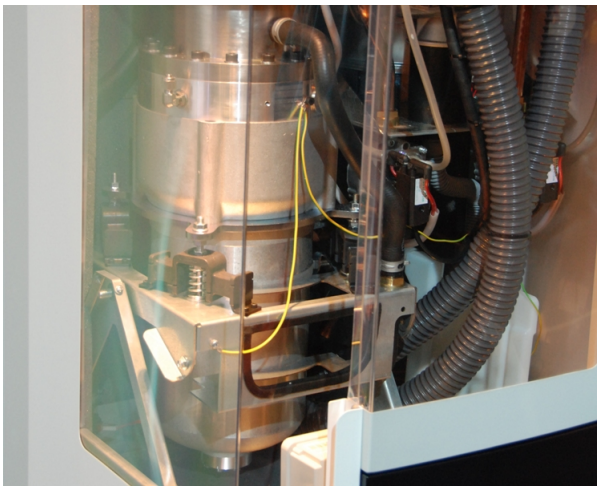


Figure 1: Example of a micro-CHP; the MTS Infinia micro-CHP

# 1 Introduction

Over the last years the demand for a more durable and efficient electricity generation is getting more and more attention. Partly because the demand for energy is increasing, and partly because the earth is running out of the traditional energy resources, such as oil and gas. In this thesis we look at one particular approach which can contribute to solving this problem, which is the micro-CHP. CHP here is an approximation for Combined Heat and Power. A micro-CHP is a special kind of boiler, that can be used within a household. As the abbreviation already suggests, the micro-CHP simultaneously produced heat and power, or electricity. The heat can be used for hot water and central heating (if necessary), while the generated electricity used within the household or added to the network. A picture of a micro-CHP is shown in Figure 1.

With this micro-CHP it is possible to get a more efficient production of electricity than with a traditional power plant, because there the heat loss is greatly reduced, as the heat is used within the household. This leads to an energy efficiency of about 95 %, where e.g. an open-cycle gas turbine has only an efficiency of 35-42 % [5]. The functioning of a micro-CHP is shown schematically in Figure 2. Note that this combined generation implies that there is only electricity production when there is a demand for heat.

Still, there are some difficulties with this arrangement, because if electricity is only produced when there is a heat demand, the total electricity production from houses equipped with a micro-CHP becomes erratic and difficult to predict. To still have a possibility of steering the electricity production, every micro-CHP has a heat buffer where hot water can be stored, so that the production does have some flexibility in time.

From this the question arises how this buffer should be utilized. More specifically,

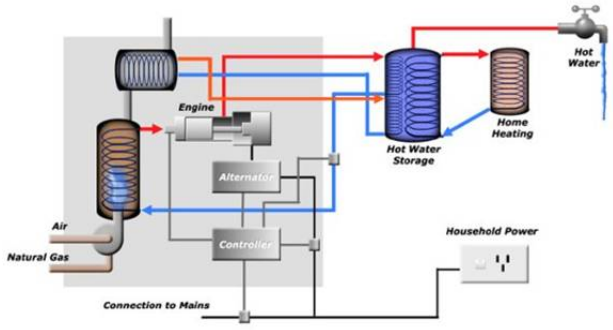



Figure 2: Schematic representation of a micro-CHP production process

if we consider a group of multiple houses equipped with a micro-CHP, we want to find a production schedule for the micro-CHPs. This schedule should ensure that the houses' heat demands are fulfilled, and the electricity production performs along the lines of a desired, more or less constant, schedule.

This problem has already been investigated by Bosman [4], where several heuristics were used to solve this scheduling problem. In one of these approaches, this problem was modelled as a very large Dynamic Programming instance. In this thesis an attempt is made to solve this Dynamic Programming problem with Approximate Dynamic Programming (ADP).

The structure of this thesis is as follows:

In Section 2 the micro-CHP problem is explained, and it is described how this problem can be modelled as a (very large) dynamic programming problem, as was done in [4]. In Section 3 ADP is introduced using an easier problem; the taxicab problem. This problem can be solved using ADP, and has the advantage that the approach here is a lot more intuitive than for the micro-CHP problem. This is the reason that we take a small detour by examining the taxicab problem, to later draw similarities that can be used for the micro-CHP problem. In Section 4 we then return to the micro-CHP problem, and describe how ADP can be implemented in this problem.

In Section 5 we shortly discuss alternative approaches to this problem, as presented in Bosman [4]. In section 6, we apply our approach to several data sets, and the results are presented an analyzed. We also compare our results with the ones found by the alternative approaches. In Section 7 a conclusion is drawn about the quality of the ADP approach and we provide some recommendations for further investigation.

# 2 The Micro-CHP problem

The micro-CHP is a boiler that produces both heat and electricity, that can be used within a house. In this thesis, we consider a situation with a group of houses that are all equipped with a micro-CHP. For every house, the heat necessary for heating the house and hot water has to come from the micro-CHP. We assume that the heat profile is known for every house, meaning that for different time intervals it is specified how much heat is required. Furthermore, every house has a heat buffer, in which it can store up to a certain amount of heat, in the form of hot water.

Of course, we will not know exactly how much heat the houses need, and when they need it. In this thesis, we assume that an estimation is known for this, based on previous heat data for the house. The deviations from this estimation are resolved by using a real-time control, for which also some additional space in the heat buffer is reserved. The real-time control will follow the original schedule as close as possible. This means that for our scheduling problem, we can assume the estimations of the heat demand to be fixed demands.

For this group of houses, it is assumed that a power company has set upper and lower bounds for the total production of electricity for different intervals. This is done to force the group to behave in a more predictable manner, which makes it easier for the company to deal with the total electricity produced by the group. A group of houses which behaves in such a way is called a virtual power plant (VPP).

The power company pays the group for the electricity produced, where the price of electricity is time-dependent.

The goal is now to maximize the revenue of the electricity sales of the group of houses, while respecting the upper and lower bound given by the power company, and while satisfying the heat demands of the different houses.

In doing this, the behaviour of the micro-CHP also has to be considered. First, there is only one level of production, so the micro-CHP either runs at full force or it is turned off; it can not run at half force. A typical electricity production scheme of a micro-CHP is shown in Figure 3. Here we can see that the electricity generation of the micro-CHP slowly builds up to a near-constant maximum production level, and that also some energy is generated during shutdown. After shutdown, the micro-CHP needs to cool down. Because of this, and to ensure that the micro-CHP runs are efficient, we infer that the micro-CHP has fixed start-up and shutdown periods. These cannot be interrupted, and are typically longer than the time needed to reach the maximum production or to stop producing.

In the next subsection this micro-CHP problem is translated into a mathematical model. After that, this model is transformed into a Dynamic Programming problem. This is first done for a single house, after which this is extended to multiple houses.

## 2.1 Modelling the micro-CHP

In this paragraph we translate the above sketched problem of a VPP into a mathematical model. First, we denote the number of houses by $N$. The total timespan for the planning of the micro-CHPs is split into $T$ intervals: $1, 2, ..., T$. This indicates that we use discrete time steps, whereby we infer that the on or off status of each micro-CHP

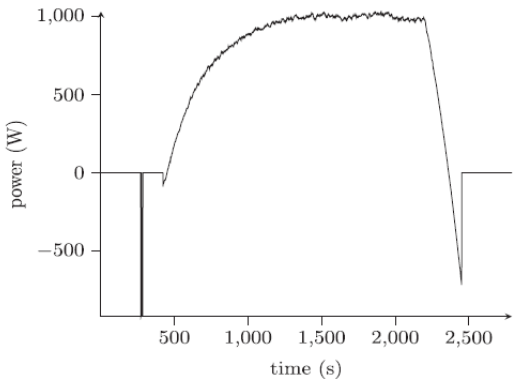



Figure 3: Generation during a micro-CHP run

within a time interval is constant. We hereby consider a micro-CHP to be turned off while it is shutting down, and during the start-up it is considered on.

### 2.1.1 Decision variables and streak values

To keep track of the status of the micro-CHPs, we introduce binary decision variables $x_{i,t}$. These are defined as follows:

$$x_{i,t} = \begin{cases} 0 \text{ if the micro-CHP in house } i \text{ is turned off during interval } t \\ 1 \text{ if the micro-CHP in house } i \text{ is turned on during interval } t. \end{cases} \tag{1}$$

As can be seen from Figure 3, the production level of a micro-CHP depends on how long the micro-CHP has been running, or how long it has been off (if it is still in the process of shutting down). This indicates that it is useful to keep track of this running time. For this we introduce variables $a_{i,t}$ with the following definition:

$$a_{i,t} = \begin{cases} \text{\# of intervals the micro-CHP has been turned on consecutively} & \text{if } x_{i,t} = 1 \\ \text{- \# of intervals the micro-CHP has been off consecutively} & \text{if } x_{i,t} = 0. \end{cases} \tag{2}$$

We refer to $a_{i,t}$ as the "streak value" of house $i$ in interval $t$. This streak value is taken at the end of the interval $t$. In this way by choosing $t = 0$, we can specify also the situation at the beginning of the planning horizon, meaning that $a_{i,0} \in \mathbb{Z}\backslash\{0\}$ is a model parameter that describes the initial streak value of the micro-CHP in house $i$. The streak values and the decision variables $x_{i,t}$ are linked by the following relation:

$$a_{i,t} = \begin{cases} \max(a_{i,t-1} + 1, 1) & \text{if } x_{i,t} = 1 \\ \min(-1, a_{i,t-1} - 1) & \text{if } x_{i,t} = 0. \end{cases} \tag{3}$$

This holds as the number of consecutive intervals is increased by 1 if the on/off status is the same as in the previous interval. When it is different, a new streak is started, starting from 1 or $-1$.

From (3) it follows that the $a_{i,t}$ are completely determined by the $x_{i,t}$ variables, for all $t > 0$, as $a_{i,0}$ is a fixed parameter. With these streak values we can also make sure that the start-up and shutdown periods are respected. Let the number of intervals during which the micro-CHP be denoted by $T_{on}$, and the number of intervals required for shutdown by $T_{off}$. With these notations we impose the following constraints on the model:

$$\begin{aligned} &x_{i,t} = 1 \text{ if } 1 \leq a_{i,t-1} < T_{on} \quad \forall i \in \{1,2,..,N\}, t \in \{1,2,...,T\} \\ &x_{i,t} = 0 \text{ if } -1 \geq a_{i,t-1} > -T_{off} \quad \forall i \in \{1,2,..,N\}, t \in \{1,2,...,T\}. \end{aligned} \tag{4}$$

### 2.1.2 The heat behaviour

We consider the heat behaviour in a single house. As mentioned in the introduction, the heat in each house is produced by the micro-CHP, and can be stored in a buffer. For this buffer there are upper and lower bounds for the amount of heat that can be stored. We assume that the lower bound is 0, and denote the upper bound of the heat buffer in house $i$ by $L_{max,i}$. To keep track of the amount of heat in the buffer of house $i$ at the end of interval $t$, we introduce the variable $L_{i,t}$, $t \in \{0,1,2,...,T\}$. This variable is defined as the amount of heat in house $i$ at the end of interval $t$, and is referred to as the "heat level" of house $i$ in interval $t$. We assume that the heat level at the beginning of the planning interval, $L_{0,t}$ is known. During interval $t$ within the planning horizon, the heat level in house $i$ is changed by the following factors:

- The house may use some up some heat for heating or hot water. As explained before, we assume that the heat demands are fixed, and use $D_{it}$ for the given amount of heat that house $i$ needs during interval $t$. The values $D_{it}$, $i \in \{1,2,...,N\}$, $t \in \{1,2,...,T\}$ are known model parameters.

- The micro-CHP can produce some heat during interval $t$, which is put into the buffer. As the production rate cannot be controlled, the heat production only depends on how long the micro-CHP has been on in interval $t$, and thus on the streak value $a_{i,t}$ of the micro-CHP. Therefore we can write the heat production in house $i$ during interval $t$ as $P(a_{i,t})$, representing the production of heat corresponding to a streak value of $a_{i,t}$.

- A third source of change in the heat level is heat loss; in every interval the heat buffer loses some heat to the outside world. In reality, the heat loss depends on the temperature difference between the boiler and the outside. This would imply that this loss is dependent on the heat level $L_{i,t}$, which would make the problem harder. However, typically the buffer is well isolated, so that the temperature inside the buffer is near-constant. Based on this, for our model we assume that the heat loss to the environment is independent of the heat level $L_{i,t}$, but may depend on the given installation of house $i$. These assumptions imply that for house $i$ there is a heat loss $HL_i$ in every interval.

Now, if the streak value $a_{i,t}$ and the heat level at the previous interval $L_{i,t-1}$ are known, the heat level after interval $t$ $L_{i,t}$ can be found using the following relation:

$$L_{i,t} = L_{i,t-1} + P(a_{i,t}) - HL_i - D_{it} \quad \forall i \in \{1,2,..,N\}, t \in \{1,2,...,T\} \tag{5}$$

In this formula $HL_i$ and $D_{it}$ are fixed model parameters, while $P(a_{i,t})$ depends on the decision variables. To make sure that heat level does not exceed the maximum amount of heat that can be stored, and that there is always enough heat to meet the demands, the following constraints are imposed:

$$0 \leq L_{i,t} \leq L_{max,i} \quad \forall i \in \{1,2,...,N\}, t \in \{1,2,...,T\}. \tag{6}$$

### 2.1.3 Electricity bounds

The electricity production has to be taken into account as well. Similar to the heat production, the electricity generation $E_{i,t}$ in house $i$ during interval $t$ also only depends on the streak value $a_{i,t}$, and can be written as $E_{i,t} = E(a_{i,t})$. As mentioned in the introduction, there are fixed upper and lower bounds for the total electricity generation of the fleet in each interval $t$, which are denoted by $E_{min,t}$ and $E_{max,t}$. This leads to the following constraints for the total electricity generation:

$$E_{min,t} \leq \sum_{i=1}^{N} E(a_{i,t}) \leq E_{max,t} \quad \forall t \in \{1,2,...,T\}. \tag{7}$$

### 2.1.4 Objective

The constraints (3) to (7) restrict the possible choices for the decision variables. Within these constraints, the objective is to maximize the total revenue over all intervals gained from selling the electricity. If we denote the price of electricity during interval $t$ by $Pr(t)$, this revenue can be written as

$$\sum_{t=1}^{T} (Pr(t) \sum_{i=1}^{N} E(a_{i,t})) \tag{8}$$

### 2.1.5 The full model

Aligning all constraints and the objective, the problem now looks as follows:

$$\max \sum_{t=1}^{T} (Pr(t) \sum_{i=1}^{N} E(a_{i,t}))$$

under the constraints:

$$\begin{aligned}
x_{i,t} &\in \{0,1\} && \forall i,t \\
a_{i,t} &= \begin{cases} \max(a_{i,t-1}+1,1) \text{ if } x_{i,t}=1 \\ \min(-1,a_{i,t-1}-1) \text{ if } x_{i,t}=0 \end{cases} && \forall i,t \\
x_{i,t} &= 1 \text{ if } 1 \leq a_{i,t-1} < T_{on} && \forall i,t \\
x_{i,t} &= 0 \text{ if } -T_{off} < a_{i,t-1} \leq -1 && \forall i,t \\
L_{i,t} &= L_{i,t-1} + P_(a_{i,t}) - HL_{i,t} - D_{i,t} && \forall i,t \\
0 &\leq L_{i,t} \leq L_{max,i} && \forall i,t \\
E_{min,t} &\leq \sum_{i=1}^{N} E(a_{i,t}) \leq E_{max,t} && \forall t \qquad (9)
\end{aligned}$$

Written this way, this problem comes down to a constrained optimisation problem. In Bosman [4] this problem was proven to be NP-hard over the number of houses $N$ by reduction to 3-partition, which means there is in general no fast solution for larger instances this problem.

With some clever reformulations, this problem can also be written as an Integer Linear Programming problem, as has been done by Bosman et al. in [1]. However, in this thesis this problem is transformed to a Dynamic Programming instance, so that ADP can be applied to this problem.

## 2.2 DP model for a single house

To get an idea of how the micro-CHP problem can be solved using Dynamic Programming, we first consider this problem with only one house. The single house problem is a lot easier to grasp, and can easily be expanded to the problem for multiple houses. For convenience, we remove the $i$ from the subscript of the variables and parameters. Also, for now we disregard the global electricity constraint (7).

In applying Dynamic Programming to this model, we use the intervals $t$ for the phases of the problem. In every phase $t, t \in 0,1,...,T-1$ the value of $x_{t+1}$ is chosen. From the constraints it follows that $a_t$ and $L_t$ follow directly from the sequence of $x_t$, and so the choice of the $x_t$ defines the entire solution.

### 2.2.1 Re-expressing the heat level

Looking at the problem, we can see that in order to find the best decisions from time $t$ it is only necessary to know the heat level $L_t$ and streak value $a_t$ of the present time. This is because all future constraints and revenues can be respectively checked and found from this information. Therefore, an optimal decision and a maximum total revenue exist for a combination of these values.

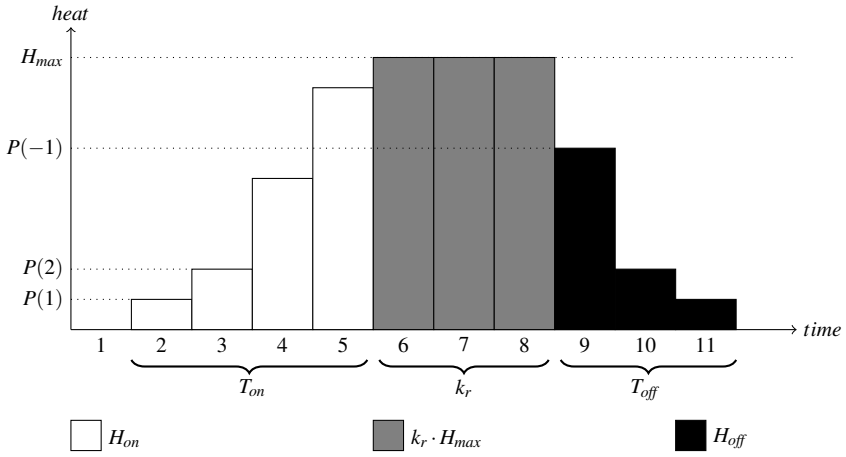



Figure 4: Example of a micro-CHP run

Yet, as $L_t$ is a continuous variable, we can not just write down all possible combinations of $a_t$ and $L_t$ to describe the states, as we need them for Dynamic Programming. Therefore we introduce two discrete variables to characterize the heat level at time $t$: $b_t$, the total number of time intervals the micro-CHP has been turned on up to time $t$, and $c_t$, the number of times the micro-CHP was switched off. Hereby an off-switch means that the micro-CHP was turned from *on* to *off*.

To clarify how $L_t$ can be found from these values, we look at the heat generated in a typical run $r$, as shown in Figure 4. First, in every run, the micro-CHP has a start-up period of $T_{on}$ intervals. Similarly, $T_{off}$ intervals are used to shut down the micro-CHP. We define $H_{on} := \sum_{i=1}^{T_{on}} P(i)$ as the total heat produced during a complete start-up, and $H_{off} := \sum_{i=-T_{off}}^{-1} P(i)$ as the heat produced while shutting down. In between these start-up and shutdown periods, the maximum heat production $H_{max}$ is produced in each time interval. If we denote by $k_r$ the number of intervals in run $r$ at which the micro-CHP remains switched on after the minimum on time, the total production during run $r$ of a micro-CHP can be written as $H_{on} + k_r \cdot H_{max} + H_{off}$.

As introduced above, we denote by $c_t$ the total number of off-switches up to time $t$, which also represents the number of runs that have finished within the planning interval. For now we assume that each of these off-switches corresponds to a completed run which is entirely inside the planning interval. To find the amount of *on*-intervals during these runs, we define a parameter $\tilde{b_t}$ describing the total on time during the first $c_t$ runs, which are already completed. $\tilde{b_t}$ can be found as follows:

$$\tilde{b_t} = \begin{cases} b_t - a_t & \text{if } a_t > 0 \\ b_t & \text{if } a_t < 0 \end{cases} \tag{10}$$

Using these notations and assumptions, the total amount of heat produced during

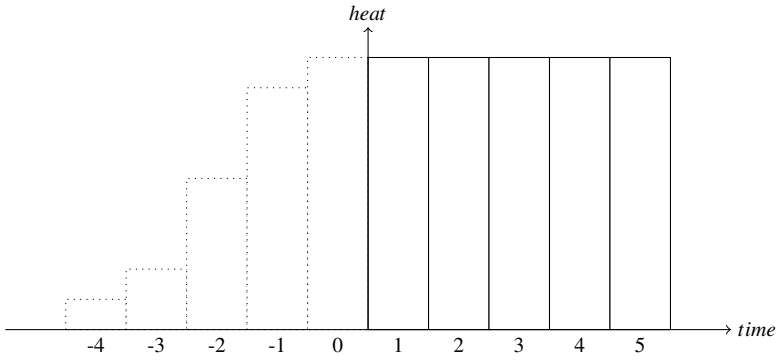



Figure 5: Example of the beginning phase of the planning horizon; $a_0 \geq T_{on}$

the first $c_t$ completed runs, $H_c$, equals:

$$\begin{aligned} H_c &= \sum_{r=1}^{c_t}(H_{on} + H_{off} + k_r \cdot H_{max}) \\ &= c_t \cdot (H_{on} + H_{off}) + H_{max} \sum_{r=1}^{c_t} k_r \\ &= c_t \cdot (H_{on} + H_{off}) + H_{max} \cdot (\tilde{b_t} - c_t T_{on}) \end{aligned} \tag{11}$$

We get the last equation from observing that the $\tilde{b_t}$ *on*-intervals are used for either start-up intervals or maximum production intervals. As the total number of intervals used for start-up equals $c_t T_{on}$, the number of maximum production intervals $\sum_{r=1}^{c_t} k_r$ equals $\tilde{b_t} - c_t T_{on}$.

However, we still need to deal with the assumptions we have made. First of all, we consider the assumption that each of the $c_t$ runs was entirely in the planning horizon. This is not always true for the first run, which may have started before time 0, nor for the last run, that may still be in the process of shutting down. Also, a new run could have started after run $c_t$. Because of this, we use correction factors $H_{start}$ to correct $H_c$ for the first run, and $H_{end}$ to correct for the last run. Hereby we assume that the micro-CHP was turned off during at least one interval after interval 0, which can also be written as $a_t < t$. For $a_t \geq t$, we perform a separate calculation. In finding $H_{start}$ we consider four different cases, based on the value of the streak value at the beginning of the planning horizon $a_0$:

- If $a_0 \geq T_{on}$, the micro-CHP was turned on initially, and the start-up of the first run was already completed during interval 1. Therefore the start-up intervals of the first run should not be considered in finding the total heat production. Instead, these intervals have to be counted as maximum production intervals. As the entire start-up happened before the first interval, the intervals of one start-up have to be replaced by maximum production intervals, as in Figure 5. In this case the correction factor equals $T_{on}H_{max} - H_{on}$.

- If $0 < a_0 < T_{on}$, as depicted in Figure 6, only the first $a_0$ start-up intervals were before the first interval. These intervals again have to be counted as maximum

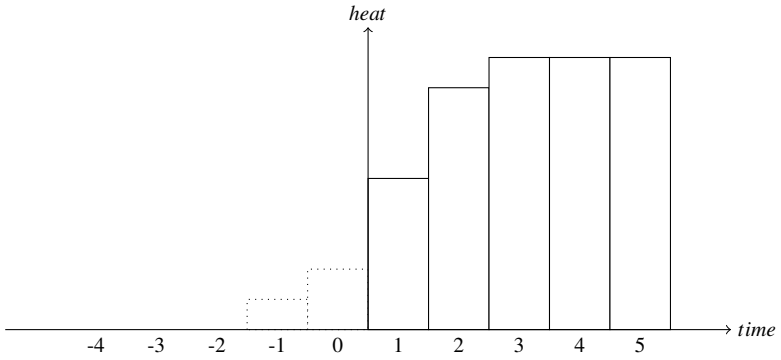



Figure 6: Example of the beginning phase of the planning horizon; $0 < a_0 < T_{on}$

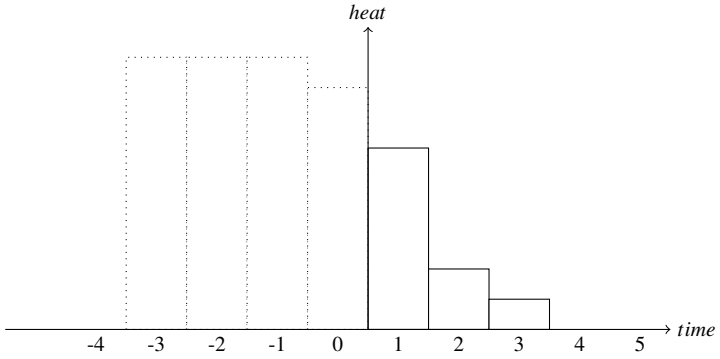



Figure 7: Example of the beginning phase of the planning horizon; $-T_{mathitoff} < a_0 < 0$

production intervals. As the heat produced before interval 1 equals $\sum_{i=1}^{a_0} P(i)$, the correction factor is equal to $a_0 H_{max} - \sum_{i=1}^{a_0} P(i)$.

- If $-T_{off} < a_{0,t} < 0$, the micro-CHP was switched off in interval 0, but was still producing some heat as it is shutting down (see Figure 7). Since the off-switch of that run occurred before interval 1 it is not included in $c_t$. Therefore, in this case an extra amount of heat of $\sum_{i=-T_{off}}^{a_0-1} P(i)$ needs to be added.

- Finally, if $a_0 < -T_{off}$, we have a situation similar to Figure 4, and no alterations on $H_c$ are necessary.

Combining these observations, we get:

$$H_{start} = \begin{cases} T_{on} H_{max} - H_{on} & \text{if } a_0 > T_{on} \\ a_0 H_{max} - \sum_{i=1}^{a_0} P(i) & \text{if } 1 \leq a_0 \leq T_{on} \\ \sum_{i=-T_{off}}^{a_0-1} P(i) & \text{if } -T_{off} \leq a_0 \leq 0 \\ 0 & \text{if } a_0 < -T_{off} \end{cases} \tag{12}$$

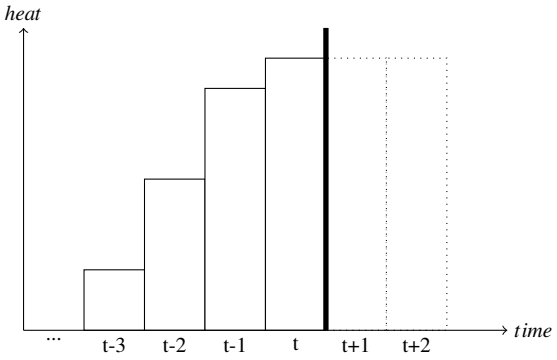



Figure 8: Example of the final phase of the planning horizon; $a_t > 0$

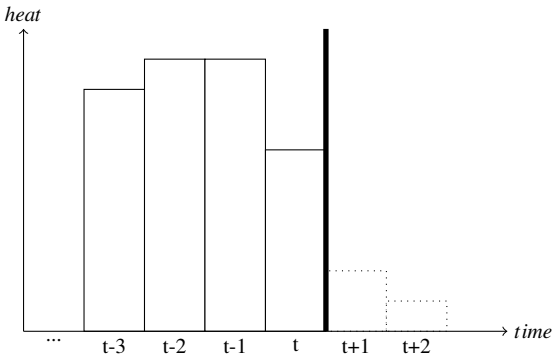



Figure 9: Example of the final phase of the planning horizon; $-T_{off} < a_t < 0$

To calculate $H_{end}(a_t)$ we consider different three different scenarios for the values of $a_t$:

- The situation where $a_t > 0$ is depicted in Figure 8. As the production intervals of the current run were not considered in $H_c$ by the definition of $\tilde{b_t}$ in (10), the correction consists of the heat produced during the last run: $\sum_{i=1}^{a_t} P(i)$. Note that this heat was all produced after interval 1, as we assumed that the micro-CHP has been off at some interval up to time $t$.
- If $T_{off} < a_t < 0$, the shutdown of the last run was not completed, as can be seen in Figure 9. To correct for that, an amount of $\sum_{i=-T_{off}}^{a_t-1} P(i)$ heat has to be subtracted from $H_c$.
- If $a_t < -T_{off}$, the last run has been completed, and no correction is necessary.

This results in the following expression for $H_{end}$:

$$H_{end} = \begin{cases} \sum_{i=1}^{a_t} P(i) & \text{if } a_t > 0 \\ -\sum_{i=-T_{off}}^{a_t-1} P(i) & \text{if } -T_{off} < a_t < 0 \\ 0 & \text{if } a_t < -T_{off} \end{cases} \tag{13}$$

With these correction factors, the total heat produced up to time $t$, $H_{total}$, can be found. For this, if $a_t < t$, we can use the sum of $H_c(t)$ and the correction factors we have just defined. If $a_t \geq t$ the micro-CHP has been on at all intervals, so the heat produced is just the part of the current run starting from interval 1, which equals $\sum_{i=a_0+1}^{a_t} P(i)$. Using this, we can now express $H_{total}$ as follows:

$$H_{total} = \begin{cases} \sum_{i=a_0+1}^{a_t} P(i) & \text{if } a_t \geq t \\ H_c(t) + H_{start} + H_{end} & \text{if } a_t < t \end{cases} \tag{14}$$

As all values used in the expression above can be derived from $a_t$, $b_t$ and $c_t$, $H_{total}$ is also a function of these three variables. As the heat losses $HL_{i,t}$ and the heat demands $D_{i,t}$ are model parameters, we can find the heat level at time $t$ from $a_t$, $b_t$ and $c_t$ as follows:

$$L_t := L_0 + H_{total} - \sum_{i=1}^{t} (HL_{i,t} + D_{i,t}) \tag{15}$$

### 2.2.2 States, decisions and transitions

As $L_t$ can be found from $a_t$, $b_t$ and $c_t$, and as $a_t$ and $L_t$ adequately describe the past decisions of the problem, we can now conclude that the three variables $a_t$, $b_t$ and $c_t$, completely describe the current position of the house, and the implications of possible future decisions from time $t$. We therefore characterize a state $\hat{s_t}$ at time $t$ by a combination $(a_t, b_t, c_t)_t$.

From every state $\hat{s_t}, t \in \{0, 1, 2, ..., T-1\}$, we define a decision $\hat{d}$, which can be either *on* or *off*. This decision describes whether the micro-CHP is on or off during the subsequent interval $t+1$.

Below we describe how the state changes when in state $\hat{s}_t$ decision $\hat{d}$ is made. This is done using a transition function $s_{t+1} = \hat{T}r(\hat{s}_t, \hat{d})$, where $\hat{s}_t = (a_t, b_t, c_t)_t$:

$$\hat{T}r((a_t, b_t, c_t)_t, on) = \begin{cases} (a_t + 1, b_t + 1, c_t)_{t+1} & \text{if } a_t > 0 \\ (1, b_t + 1, c_t, t + 1)_{t+1} & \text{if } a_t < 0 \end{cases} \tag{16}$$

$$\hat{T}r((a_t, b_t, c_t)_t, off) = \begin{cases} (-1, b_t, c_t + 1)_{t+1} & \text{if } a_t > 0 \\ (a_t - 1, b_t, c_t)_{t+1} & \text{if } a_t < 0 \end{cases} \tag{17}$$

These transitions are not always feasible, as some of the constraints (minimum run time, minimum off time (4) or the heat level constraint (5)) may not be satisfied. Yet, we formally allow these decisions, and deal with these infeasibilities in a different way.

Note that, from a given state, a set of future decisions always has the same pay-off structure, independent of how the state was reached. This means that $\hat{s}_t$ has its own future decision space, and so we can create a subproblem of maximizing the future revenue from state $\hat{s}_t$. This means that a state $\hat{s}_t$ also has an optimal decision $d_{opt}(\hat{s}_t)$.

### 2.2.3 The value function

We introduce for every obtainable state $\hat{s}_t$ a value function $\hat{V}(\hat{s}_t)$, which indicates the maximum revenue that can be obtained from state $s_t$. Obviously, $\hat{V}((a,b,c)_T) = 0$ for all states $(a,b,c)_T$ as no further revenue is obtained after interval $T$.

We now define $\hat{F}(\hat{s}_t, \hat{d})$ as the immediate revenue gained in interval $t+1$ after choosing decision $\hat{d}$ from state $\hat{s}_t$. If any of the constraints are violated during this interval, $\hat{F}(\hat{s}_t, \hat{d})$ takes the value $-\infty$. If the constraints are satisfied, $\hat{F}(\hat{s}_t, \hat{d})$ is the revenue obtained by selling the electricity that was generated in interval $t+1$, i.e. $E(a_{t+1}) \cdot Pr(t+1)$.

This enables us to write the following recursion for the value function $\hat{V}(\hat{s}_t)$:

$$\hat{V}(\hat{s}_t) = \max_{\hat{d} \in \{on, off\}} \hat{F}(\hat{s}_t, \hat{d}) + \hat{V}(\hat{T}r(\hat{s}_t, \hat{d})). \tag{18}$$

The idea behind this recursion is that the maximum revenue that can be obtained after decision $d$ is chosen is equal to the immediate revenue in the following interval, $F(s,d)$, plus the maximum revenue that can be obtained from the state reached, $\hat{V}(\hat{T}r(\hat{s}_t, \hat{d}))$. Taking the maximum over all possible decisions yields the maximum revenue that can be obtained from state $\hat{s}_t$, $\hat{V}(\hat{s}_t)$.

Using this, we can find the values in phase $t$ given the values in phase $t+1$. The set of possible states is finite, as we have $a_t \in \{-T, -T+1, ..., T\} \cup \{a_0 - T, a_0 - T + 1, ..., a_0 + T\}$, $b_t \in \{0, 1, ..., T\}$ and $c_t \in \{0, 1, ..., T\}$ for all intervals $t$ in the planning horizon. As we know the values in phase $T$, we have a starting point for this recursion, so we can track back the values in every phase, until the initial state $(a_0, 0, 0)_0$ is reached. This determines the optimal value, and the optimal path can then be found by moving forward in time, taking the optimal decision in every state. As infeasible paths have value $-\infty$, feasible paths always take priority over infeasible paths.

The complexity of this approach is of order $T^4$, as $a_t$, $b_t$, $c_t$ and $t$ all have order $T$ possibilities, so there are $O(T^4)$ states, and every state is visited only once.

## 2.3 The problem for multiple houses

In this subsection we expand the dynamic programming formulation in paragraph 2.2 for a situation with multiple houses. Hence, we now add an $i$ in the subscript of the parameters and variables which depend on the house. Formally, this means we return to the parameters in paragraph 2.1, and that $b_{i,t}$, $c_{i,t}$, $\hat{s}_{i,t}$ and $\hat{d}_i$ are denote the values of respectively $b_t$, $c_t$, $\hat{s}_t$ and $\hat{d}$ for house $i$.

### 2.3.1 New states, decisions and transitions

Because the electricity production constraint depends on all decisions in the different houses, we cannot just consider each house separately. Instead of this, we aggregate the states and decisions to and get states $s_t := (\hat{s_1}, \hat{s_2}, ...\hat{s_N}) \in S$ and decisions $d := (\hat{d_1}, \hat{d_2}, ..., \hat{d_N}) \in D, \hat{d_i} \in \{on, off\}$. The transition function $Tr(s_t, d)$ is given by:

$$Tr(s_t, d) = (\hat{Tr}(\hat{s}_{1,t}, \hat{d_1}), \hat{Tr}(\hat{s}_{2,t}, \hat{d_2}), ..., \hat{Tr}(\hat{s}_{N,t}, \hat{d_N})) \tag{19}$$

### 2.3.2 The new value function

With this new state definition a state still contains all information required for the future decisions and revenues. We can therefore use $V(s_t)$ to describe the maximum revenue from state $s_t$. In a similar way, we can define $F(s_t, d)$ to describe the total revenue earned in the subsequent period $t+1$, which is equal to the sum of the revenues of the houses $\sum_{i=1}^{N} \hat{F}(\hat{s}_{i,t}, \hat{d})$. Note that if a constraint is violated the total revenue will still be equal to $-\infty$. However, in the situation with multiple houses we should also consider the electricity constraint (7), which was disregarded in the situation with a single house. We once again infer a revenue of $-\infty$ if this constraint is violated. This results in the following formula for $F(s_t, d)$:

$$F(s_t, d) = \begin{cases} \sum_{i=1}^{N} \hat{F}(\hat{s}_{i,t}, \hat{d}) & \text{if } E_{min,t+1} \leq \sum_{i=1}^{N} E(a_{i,t+1}) \leq E_{max,t+1} \\ -\infty & \text{otherwise} \end{cases} \tag{20}$$

With these two definitions the recursive value function can be written as follows:

$$V(s_t) = \max_{d}(F(s_t, d) + V(Tr(s_t, d)) \tag{21}$$

Again, this can be solved by tracking back in time. However, where the number of states in the single house problem was of order $T^4$, here we have an aggregated state of $N$ such states, which are independent. The new order of states is therefore $T^{4N}$, which becomes a very large number for the values of $T$ and $N$ we wish to examine. For example, if $T = 24$ and $N = 25$, which would make a relatively small instance, $T^{4N}$ approximates $1.08 \cdot 10^{138}$. It is therefore clear that the general problem of this form is too large for any computer or database to solve. Therefore, we look at an approximation technique for such large DP's: Approximate Dynamic Programming.

In order to get some feeling with this technique, in the next section we first look at a simpler problem where ADP can be used, namely the taxicab problem. In Section 4 we then return to the micro-CHP problem.

# 3 The taxicab problem

Dynamic Programming (DP) is a technique used for solving decision problems, where multiple decisions have to be taken in sequence. Typical is that there are multiple paths to arrive at a certain decision epoch, and the optimal strategy from that point does not depend on previous decisions.

This method usually works fairly good when it is applicable, and can be used in e.g. path-finding algorithms and inventory management problems. However, for some problems the number of states becomes too large. For example, consider an inventory which can keep up to 9 units of 100 different products. Then there are $10^{100}$ different inventory positions (as of each product any number between 0 and 9 units may be available) for each time unit. If all these positions are possible at a certain time, at least $10^{100}$ subproblems have to be investigated, which is of course impossible. As we have seen in the previous section, the micro-CHP scheduling problem is another example.

To still find a solution to these types of problems, albeit not an optimal one, we can use Approximate Dynamic Programming (ADP). ADP seeks to only consider a small but relevant subset of the state space, for which estimates of the states' values are used. Then this estimated value function is used to find a series of good (not necessarily optimal) decisions. After that the value function is updated using the actual values found in this decision path. This is repeated until a certain stopping criterion is met.

In order to get more feeling of how ADP works, we first look at a simple problem known as the taxicab problem. In this problem we have to find the shortest path for a taxicab through an orthogonal grid. This problem can easily be modelled as a Dynamic Programming problem, which can be solved using ADP. In this section we look at both methods and make a comparison.

In this section we first introduce Dynamic Programming, and then turn to the taxicab problem. There we first introduce the problem, then provide the ADP approach, and finally come to some conclusions and recommendations for the main problem of this thesis, the micro-CHP problem.

## 3.1 Introduction to Dynamic Programming

As stated in the beginning of this chapter, Dynamic Programming is used in sequential decision problems. A decision epoch which contains all relevant information for the future decision and pay-out structure is called a state $s$. Note that for a state only the future is relevant, so it does not matter how the state was reached. Typically, in Dynamic Programming the same state can be reached from different paths.

In every state a decision $d$ has to be taken, after which a transition takes place to a new state $Tr(s,d)$. The set of possible decisions in state $s$ is denoted as $D(s)$. During the transition, a revenue $F(s,d)$ can be obtained. The objective of the problem is to maximize the total revenue.

In DP, the states are grouped in phases, such that after every decision from a state one arrives at a state in a later phase, and the total number of phases is finite. From this it follows that the number of decisions taken also is finite.

One of the ways to solve a DP instance is to define a value $V(s)$ for every state $s$, which is the value obtained from following the optimal strategy starting from that state.

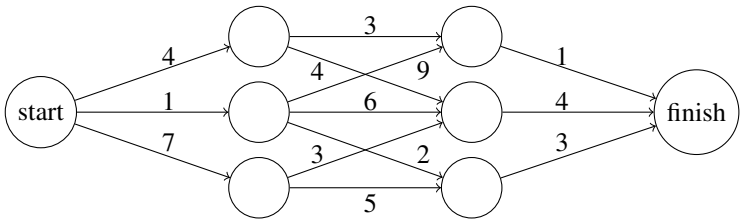



Figure 10: Example of a DP-instance

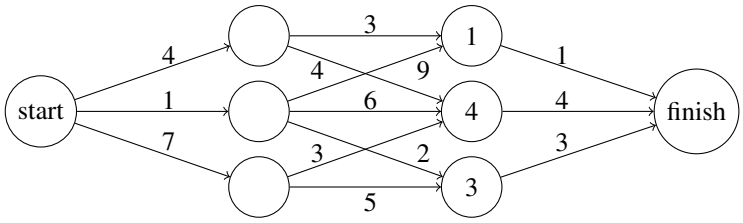



Figure 11: Example of a DP-instance; step 1

This value is trivial for the states in the final phase, as no decisions have to be taken from there. Then the values in the final phase can be used to find the values in the previous phase, by calculating the revenues of the different decisions. As the revenue after choosing decision $d$ is equal to $F(s,d)+V(Tr(s,d))$, we find the following recursion for every state:

$$V(s) = \max_{d \in D(s)} F(s,d) + V(Tr(s,d)) \tag{22}$$

Using this recursion, we can track back along the phases, until the starting point of the problem is reached. If we then look up the optimal decision in every state along the optimal path, starting at the initial phase, we can find the optimal solution and the optimal value of the problem.

As an example, consider the shortest path problem in Figure 10. The goal here is to find the shortest path in this directed graph from start to finish, where the lengths of the edges are given. We can see that this is a DP instance with four phases, sorted vertically, as every edge is directed to the right.

For the points neighbouring to the final node, i.e. points from which the final node can be reached, we can see that there is only one path to the exit, with distances 1, 4 and 3 respectively. These numbers can be filled in as values $V(s)$ of the corresponding states $s$, as is done in Figure 11.

Now we can consider the states in the second phase. For this, we can use the recursion (22), but as this is a minimization problem, we have to take the minimum, i.e.:

$$V(s) = \min_{d \in D(s)} F(s,d) + V(Tr(s,d)) \tag{23}$$

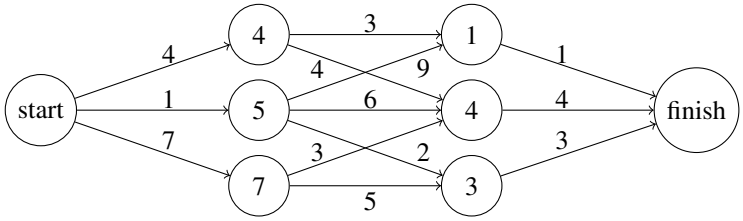



Figure 12: Example of a DP-instance; step 2

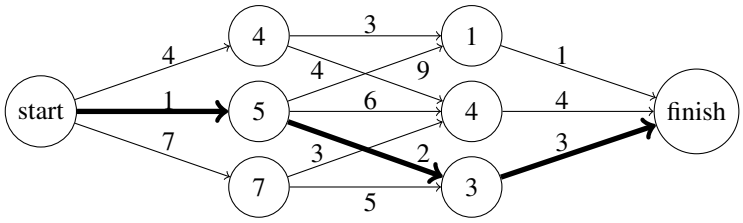



Figure 13: Example of a DP-instance; optimal solution

For the top-most state in the phase, we can see there are two possible decisions: the top one corresponds to a path length of 3 leading to a state of value 1, and the second decision has length 4 and leads to a state with value 4. As the revenues in this situation are the path lengths, we find that the value of the examined state is equal to $\min(3+1, 4+4) = \min(4, 8) = 4$, which corresponds to the top-most decision.

Doing the same for the middle state, we find a value of $\min(9+1, 6+4, 2+3) = \min(10, 10, 5) = 5$, and for the bottom state we have $\min(3+4, 5+4) = \min(7, 9) = 7$. Filling in these values leads to Figure 12.

Now the value of the initial state can be found, which is equal to $\min(4+4, 1+5, 7+7) = \min(8, 6, 14) = 6$. Therefore, the shortest path has length 6, and by tracking back through the optimal solution and looking which decision corresponded to the minimum path length, we can see the optimal path of the original problem is the path shown in Figure 13. This value could also have been found by starting from the "start" node, and keeping track of the minimal cost to reach each state. This is known as "forward Dynamic Programming", while the technique we discussed is called "backward Dynamic Programming". Our ADP approach is most similar to backward Dynamic Programming, which is why it was discussed here.

## 3.2 The taxicab problem

In the taxicab problem we consider a taxicab that is located somewhere on a two-dimensional grid, and has to travel to another given location of the grid, the exit. The goal is then to reach the exit in as few steps as possible, whereby a step is defined as a move of one grid point (left, right, up or down). In addition, some points of the grid are inaccessible for the taxicab. The grid is assumed to be bounded and rectangular with

height $n$ and width $m$.

We call the position the taxicab starts $s_0$ and the exit, the place the taxicab has to go to, $s_e$. We define a path as a sequence of adjacent accessible grid points $[s_0, s_1, ..., s_e]$. Hence, a path is a way for the taxicab to reach the exit. The number of grid positions in such a path minus one is called the path length. We can see that the number of steps required for the taxicab is equal to the path length.

We can write this problem as a Dynamic Programming problem as follows: let the state space consist of the accessible points on the grid. For each state $s$ let $V(s)$ be equal to the minimum distance which is needed to get from point $s$ to the exit. Of course, for the exit this distance is 0. For the points adjacent to the exit, the distance is 1, and for the points bordering those points the distance is 2 (except for the exit itself of course). Continuing this process results in algorithm 1 to solve the taxicab problem with DP:

**Algorithm 1** Solve the taxicab problem using DP

Set $V(s_e) = 0$, set $n := 0$ and $V(s) = m \cdot n \; \forall s \neq s_e$.
Set list $L := [s_e]$
**while** $V(s_0) = m \cdot n$ **do**
  set $n := n + 1$
  **for all** states $s$ in L (i.e. all states that satisfy $V(s) = n - 1$) **do**
    set $V(s') := n$ for all neighbours $s'$ of $s$ for which $V(s') > n$
    add the states $s'$ to $L^*$
  **end for**
  $L := L^*$; $L^* := \emptyset$
**end while**

One can easily verify that in step $n$ this algorithm finds all the points at distance $n$ from the exit point, so when the starting point is reached, the shortest distance from start to exit is known. The complexity of this approach is of $O(m \cdot n)$, as each point is handled at most once. The number of neighbours is at most 4 for every state, so the complexity of handling a node is constant. This algorithm solves the problem quite satisfactorily, but if $m$ and $n$ become large, it may not be too useful to search in every possible direction, but to instead search more towards the direction where the exit is situated.

## 3.3 ADP for the taxicab problem

In this subsection we attempt to solve the taxicab problem with Approximate Dynamic Programming. The aim here is to find a method that searches for a path to the exit, by repeatedly selecting steps that they are more likely to bring the taxicab closer to the exit. If the grid is not too complex, i.e. there are only few or no infeasible points, the shortest path has length $O(\max(m,n))$, as the Manhattan distance between the starting point and the exit is at most $m+n$. The ADP path is found step by step, where executing a step takes a constant time. Therefore, if the length of the path found is of the same order as the shortest path, we can see that the ADP algorithm finds this path in $O(\max(m,n))$ time.

After a path has been found, using an algorithm discussed later in this paragraph, the information on this path is used to improve on this path. We do this by looking back along the path to see which improvements were found. If we do this $k$ times (where $k \ll min(m,n)$), this method should finish in $k \cdot O(max(m,n)) \ll O(m \cdot n)$ time. We can see that this method indeed promises to give results faster, at the cost of losing certainty of finding an optimal solution.

In the next subsection we first explain how the value function works, and is updated once a path has been found. Then it is described how a path is found, whereby the value function is one of the factors used to determine the path.

### 3.3.1 Updating the value function

The base of the search for a path in this approach is a probability distribution over the possible directions for a given state. The idea is that we give 'better' directions a higher probability. This probability distribution depends partly on known value estimations found from previous paths. The concrete way how a given path influences these decisions is presented in the following. In this subsection we assume that a path from start to exit has been found, and describe how the value function is updated.

Instead of the actual distance to the exit (which is unknown here), we use the minimal distance we have encountered in previous paths, $\tilde{V}(s)$. Therefore $\tilde{V}(s)$ is always an upper bound for the value $V(s)$ of state $s$.

Since the minimum distance can never be more than the number of grid points on the grid, we initialize $\tilde{V}(s)$ as follows:

$$\tilde{V}(s) = \begin{cases} 0 & \text{if } s = s_e \\ m \cdot n & \text{if } s \neq s_e \end{cases} \tag{24}$$

Now, once a path to the exit $(s_0, s_1, ..., s_{k-1}, s_k)$, where $s_k = s_e$, has been found, we track back along this path. In every state $s_i$, we then update the values $\tilde{V}(s)$ for a state in the path $s_i$ by taking the minimum of the current minimum path length $\tilde{V}(s)$ and the path created by first moving to state $s_{i+1}$ and then taking the shortest path from there. As this path has length has $\tilde{V}(s_{i+1}) + 1$, the value function can be updated by setting:

$$\tilde{V}(s_i) := min(\tilde{V}(s_{i+1}) + 1, \tilde{V}(s_i)) \tag{25}$$

for states $\{s_{k-1}, s_{k-2}, ..., s_2, s_1\}$.

### 3.3.2 Finding the path

In this subsection it is explained how a new path is found. We first define the set of possible directions $\bar{D} := \{up, down, left, right\}$. Before taking a step, for every possible direction $d \in \bar{D}$ a probability to walk in that direction is determined. Then, a realization of this probability distribution determines the direction of the next step in the path. In the implemented algorithm we have chosen to let the probability depend on the following measures:

- the Manhattan distance to the exit: The smaller it is, the more likely it should be to take a step in this direction.

- not going back: if the taxicab goes back to the grid point it just came from, it is likely that no new information is obtained. Therefore, the probability of returning to the previous point is made smaller.
- forward consistency: To prevent small cycles, which do not add a lot of information, we give a bonus to the probability that the direction is in the same direction as the previous step.
- the minimum distance found so far (i.e. the value $\tilde{V}(s_i)$ for the different points $s_i$ adjacent to the current position of the taxicab): Paths with a lower value have been found to result in shorter paths, so these states are more likely to be used in a path of shorter length.

We now translate these measures into numbers, depending respectively on parameters $A$, $B$, $C$ and $D$, which are fixed model parameters. These parameters describe the importance of each measure, and are used as follows:

$$A(s,d) = \begin{cases} A \text{ if the Manhattan distance to the exit is decreased} \\ 0 \text{ otherwise} \end{cases} \tag{26}$$

$$B(s,d) = \begin{cases} B \text{ if } d \text{ is in the opposite direction of the previous step} \\ 0 \text{ otherwise} \end{cases} \tag{27}$$

$$C(s,d) = \begin{cases} C \text{ if } d \text{ follows the same direction as the previous step} \\ 0 \text{ otherwise} \end{cases} \tag{28}$$

$$D(s,d) = \begin{cases} D \text{ if } \tilde{V}(Tr(s,d)) \text{ is minimal for all } d \in \bar{D} \text{ and } \tilde{V}((Tr(s,d)) < m \cdot n \\ 0 \text{ otherwise} \end{cases} \tag{29}$$

Note that the last condition in $D(s,d)$ makes sure that $D(s,d) = 0$ if the value of all neighbouring states of $s$ equals the initial value, as that indicates there is no information about the distance to the exit. A positive value for $D(s,d)$ in this situation would therefore only increase the randomness of the algorithm, which is not desired. For the other variables it is simply checked whether the desired feature is present. If it is, the corresponding variable gets value $A$, $B$, $C$ or $D$.

These variables are used to define the probability score. This is a nonnegative number that is calculated for every allowed direction, and is used to find the actual probability of walking in a certain direction. For the probability score $PS(d)$ we have chosen the following formula for every possible direction $d \in \bar{D}$:

$$PS(s,d) = \begin{cases} K + A(s,d) - B(s,d) + C(s,d) + D(s,d) & \text{if } d \text{ leads to an allowed point} \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

In this formula, $K$ is a parameter that can be used to increase or decrease the randomness of the decisions. If $K$ is small compared to $A$, $B$, $C$ and $D$, the factors almost completely decide which direction is taken. As $K$ takes larger values, the directions are chosen more randomly.

In choosing these parameters, we should always ensure that $PS(s,d) \geq 0$ for all $d \in \bar{D}$ and that $\sum_{d \in \bar{D}} PS(s,d) > 0$. This makes sure that we can indeed rescale the scores to probabilities. If $d$ is a direction that leads to an inaccessible point, or crosses the edge of the grid, we set $PS(d) = 0$.

From these probability scores we now define the probability of going to direction $d$, $P(s,d)$, as follows:

$$P(s,d) := \frac{PS(s,d)}{\sum_{i \in \bar{D}} PS(s,i)} \tag{31}$$

This gives a probability distribution over the different possible directions $d \in \bar{D}$, from which we can take a realization to choose a direction for the next step. This probability distribution is largely determined by the parameters $A$, $B$, $C$, $D$ and $K$. We could split these parameters into 3 groups; the global steering variables $A$ and $D$, where $A$ is a parameter that steers the path to a closer Manhattan distance to the exit, and $D$ looks at the results from previous paths. $B$ and $C$ can be considered consistency parameters, that ensure the path does not contain too many turns ($B$) and loops ($C$). $K$ is a parameter that determines the randomness of the algorithm, or the influence of the steering of the other variables.

This method leads to an iteration of the grid as given by algorithm 2. During such an iteration also the value function $\tilde{V}(s)$ is updated.

**Algorithm 2** Calibration of $\tilde{V}(s)$ with a sample path

$s \leftarrow s_0$, path $p \leftarrow [s_0]$
**while** $s_e \notin p$ **do**
  Find $P(s,d)$ for all directions $d$ from state $s$ as described in (31)
  Generate a direction $d$ from this probability distribution
  Add $Tr(s,d)$ to the path $p$: $p \leftarrow [p, Tr(s,d)]$
  $s \leftarrow Tr(s,d)$
**end while**
Update $V(s)$ as described in (25)

This algorithm is repeated $k$ times in order to find a better estimate for the value function.

## 3.4 Results and analysis

In order to see if this algorithm works, and how we should choose the parameters, this algorithm is applied to different grids. We have chosen the default values of this algorithm at $A = 1$, $B = 0.9$, $C = 0.3$, $D = 1$ and $K = 1$.

For every grid, we vary $A$, $B$, $C$, $D$ and $K$ individually, where the remaining parameters keep their default values. We look at three different grids and compare the results. As not all comparison criteria are equally effective on every grid, these criteria differ per grid.

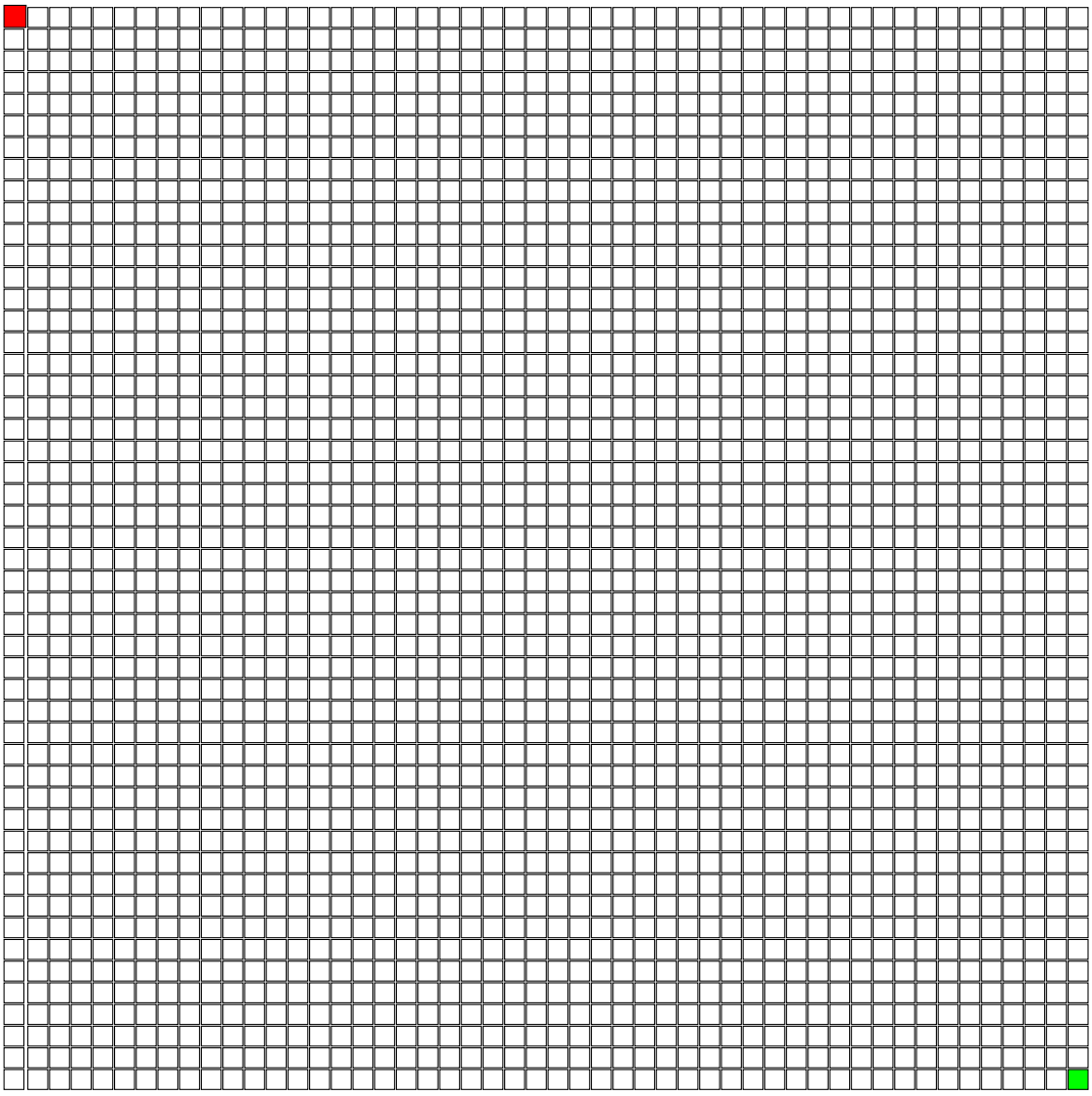


Figure 14: Grid 1

| A = | # paths | time (ms) | B = | # paths | time (ms) | C = | # paths | time (ms) |
|---|---|---|---|---|---|---|---|---|
| 0 | 18.41 | 29.57 | 0 | 16.22 | 6.99 | 0 | 18.70 | 6.45 |
| 0.25 | 21.62 | 11.54 | 0.1 | 15.95 | 6.88 | 0.1 | 18.69 | 6.26 |
| 0.5 | 21.05 | 8.39 | 0.2 | 16.68 | 5.88 | 0.2 | 18.65 | 6.04 |
| 0.75 | 19.91 | 6.89 | 0.3 | 17.26 | 6.66 | 0.3 | 18.78 | 6.21 |
| 1 | 18.61 | 6.00 | 0.4 | 16.85 | 6.16 | 0.4 | 18.82 | 6.07 |
| 1.25 | 17.34 | 4.62 | 0.5 | 17.32 | 6.20 | 0.5 | 18.49 | 6.02 |
| 1.5 | 15.65 | 4.08 | 0.6 | 17.50 | 6.00 | 0.6 | 18.70 | 6.27 |
| 1.75 | 14.56 | 3.73 | 0.7 | 18.08 | 5.84 | 0.7 | 18.63 | 6.29 |
| 2 | 13.57 | 3.37 | 0.8 | 18.64 | 5.99 | 0.8 | 18.05 | 6.03 |
| 2.25 | 12.57 | 2.96 | 0.9 | 18.60 | 6.05 | 0.9 | 17.69 | 5.84 |
| 2.5 | 11.61 | 2.67 | 1 | 19.11 | 5.96 | 1 | 17.97 | 5.89 |
| D = | # paths | time (ms) | K = | # paths | time (ms) | | | |
| 0 | 23.58 | 9.33 | 0.9 | 17.80 | 5.12 | | | |
| 0.25 | 21.84 | 7.80 | 1.1 | 19.54 | 6.09 | | | |
| 0.5 | 21.04 | 6.89 | 1.3 | 20.14 | 7.07 | | | |
| 0.75 | 19.61 | 6.33 | 1.5 | 20.90 | 8.45 | | | |
| 1 | 18.35 | 5.94 | 1.7 | 21.06 | 9.54 | | | |
| 1.25 | 17.56 | 5.41 | 1.9 | 21.33 | 10.56 | | | |
| 1.5 | 17.00 | 5.20 | 2.1 | 21.76 | 11.70 | | | |
| 1.75 | 16.39 | 5.13 | 2.3 | 21.40 | 12.31 | | | |
| 2 | 15.66 | 4.70 | 2.5 | 21.88 | 14.85 | | | |
| 2.25 | 15.04 | 4.43 | 2.7 | 22.17 | 16.92 | | | |
| 2.5 | 14.93 | 4.72 | 2.9 | 21.58 | 17.28 | | | |

Table 1: Results for grid 1

### 3.4.1 Grid 1

First of all, we consider a 50 x 50 grid with no obstacles, where the start (red) is the top left corner, and the exit (green) in the bottom right corner, as shown in Figure 14. The presented algorithm was applied to this grid. We ran this algorithm until the optimal solution of 98 steps was found, and then we looked at which iteration this optimum was found, and the time it took to find this optimal solution. As the directions are chosen randomly in this algorithm, a different run can have a different result. Therefore, we decided to restart the algorithm a total of 1,000 times for each combination of values. The average values found in these runs are shown in Table 1.

Examining the effect of a change in the parameter $A$, we see that larger values lead to a decrease of the time needed to find the optimal solution. The reason for this is fairly obvious, as in this empty grid every step toward the exit is an optimal step. So if $A$ increases, the probability of taking an optimal step increases, and an optimal solution is found faster. The number of paths needed also decreases as $A$ is increased, with the exception of small values of $A$ ($0 \leq A \leq 0.5$). This can be explained by the fact that a smaller $A$ leads to longer paths, which contain more information. Hence, as expected, a larger $A$ makes the algorithm faster and in general it takes less paths to find the optimal solution.

For $B$, we see that if $B$ becomes larger, the number of paths increases while the time needed to find the optimal solution decreases. This indicates that for higher values of $B$ the paths are shorter, but because the longer, 'worse' paths contain more information, less of these paths are required. But taking the time as an indication of the quality of the algorithm, we see that larger values for $B$ produces better results for this grid.

For $C$ we see that both the number of the iterations the optimal solution was found in and the running time do not show much difference, but both seem to show a decreasing trend. This indicates that the optimal solution is computed slightly faster and uses less paths as $C$ takes larger values.

If $D$ is increased the required number of paths decreases, as well as the running time. As $D$ increases, the focus is more about improving the previous paths than it is to find more or less new paths. As in this grid most non-optimal paths can be made optimal by finding a few shortcuts, improving previous paths is quite a good strategy, which explains this result.

For $K$ we picked a minimum value of 0.9, because a lower K could cause probabilities $P(s,d)$ to become negative. Here we see that the more random the algorithm becomes, the slower it finds the optimal paths. This is because the grid contains no inaccessible squares, and so the steering variables $A$, $B$, $C$ and $D$ all have a positive effect on the algorithm, because of the considerations described above. Therefore, for this grid we can conclude that the lower the random factor is, the better the algorithm works.

### 3.4.2 Grid 2

For the second grid (see Figure 15) we added a large block in the middle, which can only be passed by finding the narrow bridge over it. Again, the optimal solution is 98, which can be achieved by e.g. first walking to the right top corner, and then going

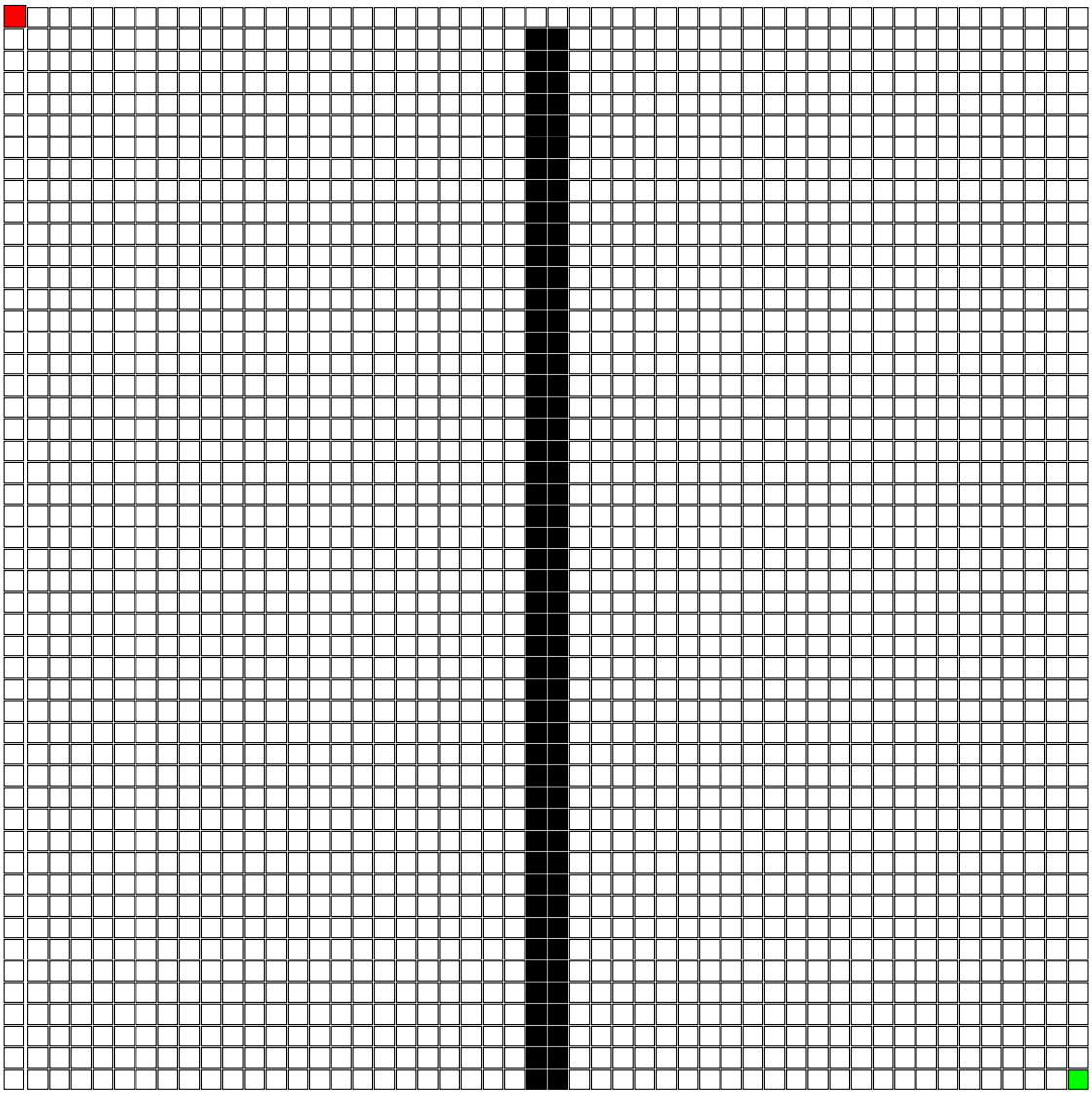


Figure 15: Grid 2

| A = | sols | avg sol | best sol | B = | sols | avg sol | best sol | C = | sols | avg sol | best sol | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 200 | 98 | 98 | 0 | 67 | 116.1 | 98 | 0 | 67 | 121.2 | 102 | |
| 0.25 | 200 | 98 | 98 | 0.1 | 80 | 116.8 | 100 | 0.1 | 72 | 117.8 | 100 | |
| 0.5 | 183 | 100.1 | 98 | 0.2 | 87 | 114.1 | 100 | 0.2 | 89 | 121.7 | 98 | |
| 0.75 | 133 | 107.3 | 98 | 0.3 | 76 | 116.6 | 100 | 0.3 | 79 | 119.3 | 100 | |
| 1 | 88 | 117.7 | 100 | 0.4 | 83 | 117.2 | 98 | 0.4 | 91 | 116.7 | 100 | |
| 1.25 | 53 | 123.8 | 100 | 0.5 | 81 | 116.8 | 100 | 0.5 | 102 | 118.6 | 98 | |
| 1.5 | 41 | 124.6 | 108 | 0.6 | 80 | 116.3 | 98 | 0.6 | 112 | 117.4 | 98 | |
| 1.75 | 13 | 123.5 | 112 | 0.7 | 87 | 120.0 | 98 | 0.7 | 115 | 114.9 | 98 | |
| 2 | 12 | 127.5 | 110 | 0.8 | 92 | 119.8 | 100 | 0.8 | 116 | 114.7 | 98 | |
| 2.25 | 19 | 121.1 | 112 | 0.9 | 88 | 118.3 | 98 | 0.9 | 131 | 113.8 | 98 | |
| 2.5 | 13 | 123.1 | 108 | 1 | 77 | 123.1 | 100 | 1 | 147 | 113.8 | 98 | |
| D = | sols | avg sol | best sol | K = | sols | avg sol | best sol | | | | | |
| 0 | 87 | 136.2 | 112 | 0.9 | 80 | 119.5 | 98 | | | | | |
| 0.25 | 68 | 132.6 | 108 | 1.1 | 110 | 116.0 | 98 | | | | | |
| 0.5 | 80 | 128.2 | 104 | 1.3 | 122 | 115.2 | 98 | | | | | |
| 0.75 | 91 | 126.8 | 102 | 1.5 | 139 | 112.7 | 98 | | | | | |
| 1 | 80 | 118.6 | 100 | 1.7 | 157 | 109.1 | 98 | | | | | |
| 1.25 | 79 | 114.1 | 98 | 1.9 | 174 | 108.7 | 98 | | | | | |
| 1.5 | 84 | 110.1 | 98 | 2.1 | 175 | 107.0 | 98 | | | | | |
| 1.75 | 92 | 107.6 | 98 | 2.3 | 186 | 106.5 | 98 | | | | | |
| 2 | 80 | 106.8 | 98 | 2.5 | 184 | 103.6 | 98 | | | | | |
| 2.25 | 83 | 105.6 | 98 | 2.7 | 191 | 102.3 | 98 | | | | | |
| 2.5 | 86 | 105.8 | 98 | 2.9 | 199 | 99.1 | 98 | | | | | |

Table 2: Results for grid 2

down. We have run the algorithm on this grid as well, but only used 200 runs, as more time was needed for this grid. Here it was found that here the optimum solution was not always reached. In fact, often no solution was found at all in 200 iterations, after which the algorithm was stopped. This is because the algorithm has to find the narrow bridge over the block, for which the variable $A$ turned out to have very bad influence, as the algorithm is pulled the path on the region left to the bottom of the block, as the minimum distance to the exit reaches a local minimum there. For this reason the average solution after 200 iterations in 200 runs (avg sol) is shown. The average is taken only over the values where a solution was found, runs without a solution were not considered in finding the 'avg sol'-value. We also show the best solution found (bestsol), and the amount of runs in which in a solution was found at all (sol). The result of the simulation can be found in Table 2.

We see that the algorithm for this grid runs better for a smaller value of $A$. This is, as mentioned before, due to the fact that initially the path is pulled towards the region left to the bottom of the block, as there is a local minimum value for the distance there. As the path keeps being pulled there, it becomes very hard to leave that region.

Looking at the performance of the algorithm when $B$ is changed, we see that the algorithm produces quite consistent results, so it seems that this variable doesn't have a lot of effect in this case.

For parameter $C$, we can observe that the algorithm seems to get better when $C$ is increased. This can easily be seen from the number of times a solution is found, and also from the average final solution. This can be explained not only by less cycles in the algorithm, but also because a straight line is needed in large part of the optimal solution.

For $D$, we see that the algorithm gets better as $D$ becomes larger. An explanation for this is that once a path to the exit has been found, the path is pulled towards this path, and more and better solutions can be found from there. Also, as D takes larger values, the presence of the "bad" variable A becomes less important.

Looking at $K$, we can observe that the algorithm gets better as the randomness factor increases. When the algorithm is more random, there is less pull to the region to the left to the bottom of the block, and so the algorithm has a better chance of finding the bridge over the block. This causes more solutions to be found, and the final solution to be better.

### 3.4.3 Grid 3

For the third grid we inserted some blocks of unaccessible squares on the grid, leading to the grid in Figure 16. It can be easily verified that there is a solution of length 98, which uses a long horizontal line in the middle of the grid.

In this grid we are interested in both the ability of the algorithm to find this path, and the time it takes to complete the runs. Therefore we once again used 1,000 runs with 200 iterations for every instance, and set out the average solution found, and the average time needed for a run. Note that in the first grid we considered the time needed to find the optimal solution, but here the time required for all 200 iterations is considered, as the optimal solution is not always found. The results can be found in Table 3.

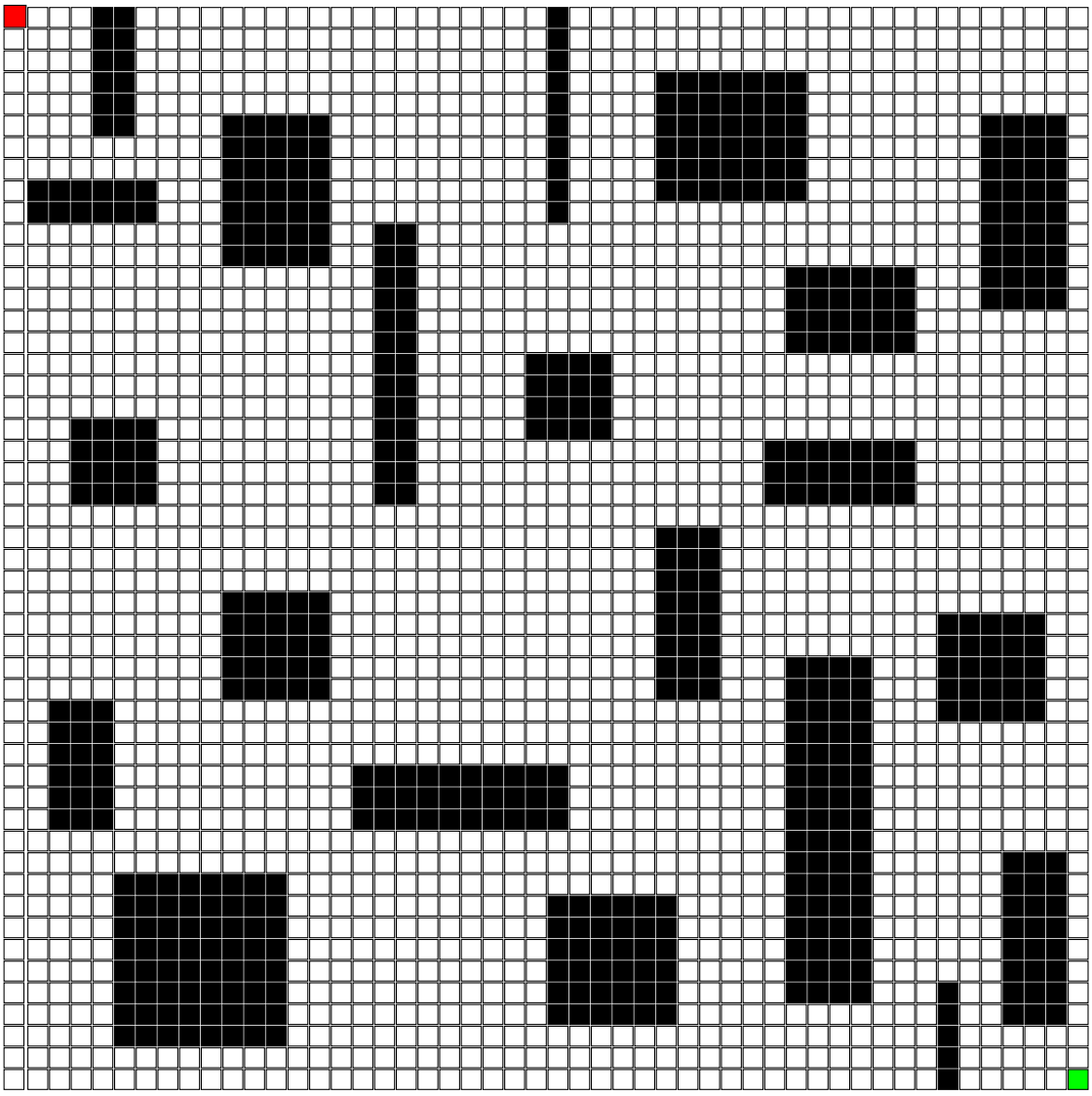


Figure 16: Grid 3

Looking at the different values of $A$, we see that the solution gets worse while $A$ takes larger values. The reason for this is that a larger value of $A$ creates a diagonal pull to the finish, where the optimal solution requires partly a horizontal line in the middle. As $A$ gets larger, it also becomes harder to get back to this horizontal line once it drops below. We also see that the algorithm initially become faster as $A$ becomes larger, but if it gets too large, the taxicab tends to get caught behind the little vertical block near the exit, and so the algorithm becomes slower for higher values of $A$.

As $B$ takes larger values, we see that the time it takes to solve the problem decreases, while the average solution seems to get a little bit worse. This is explained by the fact that decreasing the probability of going back makes it harder to return to the short horizontal path, but does create shorter paths, so it takes less time.

Increasing $C$ leads to better solutions, and also the time needed to find the solution decreases. This is because the optimal solution uses a long horizontal line, which is of course easier discovered when the probability of continuing in the same direction is increases. Also, a higher $C$ leads to more consistent paths, so once the path is going in the right direction, it is more likely to keep going there. Of course, this also holds when the path is going in the wrong direction, but apparently this is outweighed by the positive effects, causing the solving speed to increase as well.

When $D$ is increased we see that the quality of the result of the algorithm becomes worse. This is not too surprising, as the first time it is likely that the algorithm does not find an optimal algorithm, and after that it is drawn to that path, which makes it harder to deviate and find the optimal path. A larger $D$ does lead to faster solving times, because the algorithm is more likely to follow the previous path and thus typically has no problem to find the exit.

For $K$, a higher $K$ implies more randomness, which increases the probability of finding the optimal solution. It also gives the best result over all changes we tried. This is explained by the fact that a higher $K$ causes longer runs, which contain more information. Also, with a higher $K$ the path is not too much disturbed by the diagonal pull direction of $A$, and the pull towards (non-optimal) previous paths of $D$. Of course, a higher $K$ also increases the average number of steps needed to find the exit, which results in a longer running time.

## 3.5 Conclusion

The approximate dynamic program presented in this section seems to provide reasonable solutions for solving the taxicab problem. The program was run under different parameters for several different grids, and for each grid the optimal solution was found for at least some parameter configurations. Different parameters often provided different results, which were all fairly intuitive and could easily be explained, and often even predicted. So we have indeed managed to solve some instances of the taxicab problem with Approximate Dynamic Programming.

However, the grids chosen are so small that they could all be solved optimally, and almost instantaneously, using Dynamic Programming. However, the algorithm described and the results presented in this section give a good idea of how ADP works. Also, we already saw some issues that are also encountered in solving the micro-CHP problem, such as the balance between exploring the state space and exploiting the es-

| A = | avg sol | time (ms) | B = | avg sol | time (ms) | C = | avg sol | time (ms) |
|---|---|---|---|---|---|---|---|---|
| 0 | 100.4 | 114.4 | 0 | 101.59 | 84.2 | 0 | 101.82 | 64.0 |
| 0.25 | 100.9 | 78.1 | 0.1 | 101.58 | 79.0 | 0.1 | 101.79 | 63.4 |
| 0.5 | 101.36 | 70.3 | 0.2 | 101.61 | 76.5 | 0.2 | 101.70 | 63.4 |
| 0.75 | 101.63 | 66.9 | 0.3 | 101.63 | 74.1 | 0.3 | 101.64 | 63.2 |
| 1 | 101.69 | 64.6 | 0.4 | 101.57 | 71.6 | 0.4 | 101.62 | 63.1 |
| 1.25 | 101.77 | 64.0 | 0.5 | 101.65 | 69.1 | 0.5 | 101.60 | 63.0 |
| 1.5 | 101.85 | 64.8 | 0.6 | 101.67 | 66.7 | 0.6 | 101.49 | 63.2 |
| 1.75 | 101.81 | 66.4 | 0.7 | 101.63 | 64.3 | 0.7 | 101.48 | 62.6 |
| 2 | 101.82 | 69.3 | 0.8 | 101.70 | 62.3 | 0.8 | 101.34 | 62.9 |
| 2.25 | 101.88 | 73.1 | 0.9 | 101.66 | 59.3 | 0.9 | 101.29 | 62.8 |
| 2.5 | 101.85 | 77.6 | 1 | 101.63 | 57.1 | 1 | 101.20 | 62.3 |

| D = | avg sol | time (ms) | K = | avg sol | time (ms) | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 100.63 | 141.0 | 0.9 | 101.71 | 60.7 | | | |
| 0.25 | 101.09 | 97.0 | 1.1 | 101.63 | 74.5 | | | |
| 0.5 | 101.36 | 78.7 | 1.3 | 101.49 | 75.6 | | | |
| 0.75 | 101.57 | 68.5 | 1.5 | 101.17 | 83.9 | | | |
| 1 | 101.73 | 62.3 | 1.7 | 101.00 | 92.6 | | | |
| 1.25 | 101.73 | 58.2 | 1.9 | 100.83 | 102.1 | | | |
| 1.5 | 101.78 | 55.6 | 2.1 | 100.50 | 111.2 | | | |
| 1.75 | 101.80 | 52.8 | 2.3 | 100.24 | 120.8 | | | |
| 2 | 101.80 | 51.1 | 2.5 | 99.84 | 130.4 | | | |
| 2.25 | 101.78 | 49.5 | 2.7 | 99.62 | 140.1 | | | |
| 2.5 | 101.83 | 48.6 | 2.9 | 99.34 | 150.7 | | | |

Table 3: Results for grid 3

timated values. We already saw that too much exploration takes more time and leads to less useful states being visited, while too much exploitation leads to too much repetition and worse solutions as we saw in grids 2 and 3. This is a dilemma already described in literature, see for example Powell [2] and James [3].

We also saw that multiple runs of the same algorithm (but with a different random seed) can lead to different results. Therefore, it seems to be a good idea to run the algorithm multiple times and take the best results, if it involves randomness. This often works better than trying new iterations from the previous values. We also have seen that the best choice of the parameter settings depends on the grid. So if it is necessary to estimate some parameters in the model, we need to be careful.

Based on these observations and experiences with us as we turn to applying ADP to the micro-CHP problem in the next section.

# 4 ADP Approach

In this section we present an Approximate Dynamic Programming approach to the micro-CHP problem presented in Section 2. This approach is slightly different from the approach to the taxicab problem. This is because in the micro-CHP problem there are too many states to store values for all of them. Keeping track of just the values of the states that are visited is also difficult, not only because this number can become too large as well, but also because it is difficult to look up the stored values, if not all are stored.

Therefore, instead of keeping track of values for all states, as we did for the taxicab problem, in this section an approximation of the value function is considered. In this approximation, we use features of the current state, where a feature uses some characteristics of a state, indicating something about its quality. Basis functions are used to transform these features into numerical values. To obtain an approximation for the value function of a state, we use a weighted sum of these basis functions. The weights used to scale the basis functions depend on the phase, so that every phase $t$ has its own weights.

Although this method is quite different from the ADP approach in the previous section, there are a lot of similarities between the two applications of ADP, so that the experience from the previous section can still be used.

In this section first the decision space is reduced, as there are simply too many decisions from a state to consider all of them. Then the structure of the chosen value function is specified. In the final subsection we explain how the parameters of this value function are updated, using the values of the path found.

## 4.1 Reducing the decision space

In this subsection we will describe the way in which the decision space is reduced in our ADP approach. Currently, the decision vector $d$ consists of $N$ binary decision variables, where $N$ is the number of houses. This means that for a typical instance of 100 houses at each state we have $2^{100} \approx 10^{30}$ possible decision vectors. This is of course too many to consider them all. To deal with this problem, we consider only a subset of the possible decisions.

To do this, we introduce in each state a strict priority list for the houses. For this the houses are ordered on a measure of suitability to be *on* in the next interval. Once this list is determined, the only allowed decisions are to take the *on* decision for the first houses on the priority list, and the *off* decision is made for the remaining houses. Hereby the only decision that can be made is how many houses are turned on. In other words, we determine $n$, $0 \leq n \leq N$, after which the *on* decision is taken for the first $n$ houses on the priority list, and the other houses are turned *off*. As in this situation the only decision left is to determine $n$, the number of decisions is decreased from $2^N$ to $N+1$. Of course, we also lose a lot of flexibility here.

To make the priority list, we define a value $R(i)$ for every house $i$ and the priority list is then achieved by sorting the houses from low to high values of $R(i)$. This is done in the following way. First, it is checked whether the house can be turned on in the next interval without a constraint being violated. If this is not the case, this house is

of course very unsuited to be *on* in the next interval, so we impose $R(i) := \infty$. For the remaining houses, $R(i)$ is defined as the number of consecutive intervals that the *off*-decision can be taken, without a constraint being violated. Note, that if the *off*-decision cannot be taken in the next interval, so the only feasible decision is to turn the house on, we get $R(i) = 0$, putting these houses in the front of the list. This is a desirable property, as these houses are of course the best suited to be turned on.

For the other houses, the idea is that a higher value of $R(i)$ leaves more options in the next intervals. If house $i$ has $R(i) = 1$, and we decide to turn it off in the coming interval, we already know that it has to be turned on in the next interval. If we turn off a house for which $R(i) = 2$, we will likely have both the *on* and the *off* decision available in the following interval (unless the off time constraint 4 prevents that). For a house with $R(i) = 3$ it takes even longer to get a forced decision if it remains turned off, so that house receives less priority to be turned on.

We define $\tilde{D}$ as the set of decision vectors that can be obtained in this manner for $n \in \{0, 1, ..., N\}$. Note that we consider all possible values within this set in taking these decisions. This includes the decisions leading to an infeasibility, e.g. switching on a house $i$ for which $R(i) = \infty$ or switching off a house for which $R(i) = 0$. We do this because in some cases, all decisions are infeasible, or lead to an infeasibility in the future, and we still need to make a decision in these cases.

## 4.2 The value function

In this subsection we describe how the value function is initiated and updated. As we have mentioned before, the number of visited states typically becomes too large to keep track of values for all of them, as we have done in solving the taxicab problem. Therefore, we estimate this value function by considering $F$ different features that apply to any state $s_t \in S$. Then, features $f \in \{1, 2, ..., F\}$ are mapped to a number by a basis function $\phi_f(s_t)$. A weighted sum of these basis functions is used to find an estimate of the value.

In this weighted sum, the basis functions are multiplied by the phase-dependent weight of the feature $\theta_{f,t}$. These weights are updated after a decision from a state in phase $t$ has been chosen, which means that all weights are updated once during each iteration. The set of weights at a certain time is denoted by $\theta_t := \{\theta_{1,t}, \theta_{2,t}, ..., \theta_{F,t}\}$.

In formula form, the approximate value function $\tilde{V}(s_t, \theta_t)$ looks as follows:

$$\tilde{V}(s_t, \theta_t) = \sum_{f \in F} \theta_{f,t} \phi_f(s_t) \tag{32}$$

If these weights and basis functions work well, the approximate value of the reachable states can be calculated, and these values can be used to find the best decision, according to this approximation.

The approximate value function we use here has the same interpretation as $V(s_t)$ in Section 2: the maximum revenue from all future electricity sales being in state $s_t$ at time $t$. For the final interval we infer $\tilde{V}(s_T, \theta_T) := 0$, as after the final time interval no more electricity is sold.

However, we chose to create a different penalty for violation of the constraints from the one chosen in Section 2. Instead of obtaining a revenue of $-\infty$ when a constraint

is violated, an amount $P$ is subtracted from the total revenue if the global electricity constraint (7) is violated. If another constraint, e.g. the heat level constraint (6), is violated, an amount of $10P$ is subtracted. $P$ will be chosen in the order of the maximum total revenue that can be obtained by all houses in one time interval. This ensures that a feasible solution is always preferred over an infeasible solution.

This change is made because the obtained revenues of a found solution are used to update the weights $\theta_{f,t}$, and in this update values of $-\infty$ are very impractical. This becomes clear in the description of the method of updating the weights. Furthermore, we have chosen a smaller penalty when the global electricity constraint (7) is violated, so that in case of a choice of constraint violations, violation of constraint (7) takes preference. This will lead to an easier comparison between different infeasible runs, as often the same constraint is violated in these cases.

### 4.2.1 Value function decomposition

In this paragraph, the parameters and basis functions we have chosen to represent the value function are presented.

We have chosen to introduce four basis functions, so that the value function has the following form:

$$\tilde{V}(s_t, \theta_t) = \theta_{1,t}\phi_1(s_t) + \theta_{2,t}\phi_2(s_t) + \theta_{3,t}\phi_3(s_t) + \theta_{4,t}\phi_4(s_t) \tag{33}$$

In the following each of these basis functions are defined and explained.

- $\phi_1(s_t)$ is an estimate of the total maximum sales revenue of electricity from state $s_t$. To obtain this, we drop the global electricity constraint, after which we look at each house $i$ individually, and simplify the situation of the house. We assume that a house only produces when its micro-CHP is switched on, and that it produces the maximum electricity $H_{max}$ in these intervals. This implies that to find the current heat level, the streak value $a_{i,t}$ and the number of on/off switches $c_{i,t}$ are not relevant anymore; only the total running time $b_{i,t}$ matters. Finally, all constraints are discarded as well, except for the heat level constraints (6) for the final interval $T$, for all houses $i$. In this heavily simplified model, the number intervals where the micro-CHP of house $i$ has to be on is fixed for every house $i$, as this is the maximum number of *on*-intervals such that the heat level after interval $T$ is within the heat bounds. If we denote this number of intervals by $w$, the maximum revenue can be obtained by turning the micro-CHP on during the $w$ intervals in $\{t+1, ..., T\}$ where the electricity price is the highest. This is done for all houses $i$, and the sum of these revenues is used as an estimate of the total sales revenue from state $s_t$.

- $\phi_2(s_t)$ consists of the minimum total amount of penalties received for constraint violations during interval $t+1$. These penalties are the revenues of $-P$ and $-10P$ that are obtained if a constraint is violated. Note that the value of this basis function is found before the decision vector is chosen.

First we check all constraints but the global electricity constraint (7). This is done by looking for houses for which both the *on* and *off* decisions are infeasible. A penalty of $10P$ is obtained for every house for which this holds.

To account for the global electricity constraint (7) we first find the $R(i)$-value for all houses, as we did in finding the optimal decisions. Then, we can find the minimum total electricity generation in the next interval by choosing the *off*-decision for all houses for which $R(i) > 0$, and the *on*-decision for the remaining houses. Obviously, this is the minimum electricity production that can be obtained in a feasible way. If this minimum value is higher than the upper electricity bound $E_{max}$, a penalty of $P$ is unavoidable, and will be considered in $\phi_2(s_t)$. Similarly, the maximum total electricity production is found by determining the electricity production obtained when all houses are assumed *on* when $R(i) < \infty$, and this value is compared to $E_{min}$. Again, a penalty of $P$ is obtained if this value is lower.

- To calculate $\phi_3(s_t)$ we first determine the number of houses for which both the *on* and the *off* decisions are feasible. To obtain $\phi_3(s_t)$, this number is downscaled by multiplying this number with $k_3$, a fixed model parameter, which is done to make this basis function fit better to the size of the other basis functions. The idea here is that more houses with two possible decisions leads to more flexibility, which can be a useful property.

- With $\phi_4(s_t)$, we want to ensure that the average remaining electricity production is in line with $E_{min}$ and $E_{max}$. Otherwise, it could happen that a large part of the production takes place in the first part of the day, producing around $E_{max}$ in every interval, leaving not enough room for electricity production later, so that $E_{min}$ can not be reached.

  To prevent this, we first find the average electricity production per time interval from time $t$ on, $E_{avg}$, and compare that value to the minimum and maximum production in each interval. To calculate $E_{max}$, we first determine for every house $i$ an estimate of the final electricity level after interval $T$, $L_{final,i}$. This is a model parameter that lies between the heat bounds 0 and $L_{max,i}$, which tries to make an accurate prediction of the electricity level.

  Then we determine for every house $i$ the number $Z(i)$, which indicates how much heat has to be produced to reach this level, also considering the heat demands and heat losses. This is summed over all the houses, to find an estimate of the remaining heat production. To convert this estimate into an estimate of the electricity production, we estimate that the ratio between electricity and heat production is approximately the ratio of their maxima. Therefore, we define the number $\eta$ as $\frac{H_{max}}{E_{max}}$, and divide by this number to convert the heat demand to electricity production. $E_{avg}$ can now be found by dividing this number by the number of remaining intervals $T-t+1$:

$$E_{avg} := \frac{\sum_{i=1}^{N} Z(i)}{(T-t+1)\eta} \tag{34}$$

$E_{avg}$ is compared with the upper and lower bounds for the total electricity production. These differences are converted to a fraction of the maximum heat production by dividing by $NH_{max}$ to obtain the relative difference from the bounds. To do this, we define $d_u$ and $d_l$ as follows:

$$d_u := \frac{\eta(E_{max} - E_{avg})}{NH_{max}}, \quad d_l := \frac{\eta(E_{avg} - E_{min})}{NH_{max}} \tag{35}$$

These values $d_u$ and $d_l$ now provide a relative measure of how close the remaining electricity production is to the boundaries. The smaller these values, the closer they are. The following formula is used for the basis function, where 1 represents the indicator function:

$$\phi_4(s_t) := -k_4 \cdot N((0.1 - d_l)^2 \cdot 1_{d_l<0.1} + (0.1 - d_u)^2 \cdot 1_{d_u<0.1}) \tag{36}$$

The negative sign in this formula indicates that this number is a penalty. This penalty gets larger as the minimum of $d_l$ and $d_u$ becomes smaller. As with $\phi_3(s_t)$, we multiplied this number with a constant, in this case $k_4 \cdot N$, to get this basis function into the right order of magnitude.

Summarizing, the first basis function gives an upper bound for the total revenue of the electricity sales, the second looks for unavoidable penalties in the current interval, the third looks at local flexibility, and the fourth looks at global consistency and flexibility.

## 4.3 Finding a path

Using the decision space as defined in Section 4.1, at every state but the states in the final interval, (i.e. $s_t$, $t < T$), we have $N+1$ possible decisions. To find the value corresponding to a decision $d$ in state $s_t$, we observe that this consists of two parts: the revenue in the next interval $t+1$, which is equal to $F(s_t, d)$ as defined in Section 2.3, and the value of $Tr(s,d)$, the state reached after taking decision $d$, which can be estimated by $\tilde{V}(Tr(s,d), \theta_{t+1})$. The 'optimal' decision $d_{opt}(s_t)$, given the approximate value function and the weights $\theta_t$, is then

$$d_{opt}(s_t) = \arg\max_{d \in \tilde{D}} C(s_t, d) + \tilde{V}(Tr(s_t, d), \theta_t) \tag{37}$$

If the weights $\theta_{f,t}$ are defined, we can find a decision strategy that always chooses decision $d_{opt}(s_t)$ from state $s_t$. This is the strategy we use in our ADP approach. An interesting aspect of this strategy is that infeasibilities are prevented quite powerfully. If there is a series of decisions in $\tilde{D}$ that does not cause an infeasibility in the next two intervals, the algorithm will find it. This is because $C(s_t, d)$ prevents infeasibilities in the current interval, and $\tilde{V}(Tr(s_t, d), \theta_t)$ contains $\phi_2(s_t)$, which prevents infeasibilities in the interval $t+1$, if this is possible.

Starting from the initial state, this strategy can be used to find paths through the state space. We will use the actual revenues obtained on those paths to rescale the

weights $\theta_{f,t}$. For the rescaling of $\theta_t$ there are several possibilities, which include using a Kalman filter or a gradient method [2]. In this thesis the gradient method is selected, as this is easier to implement and requires less calculations.

To explain how this gradient method works in this problem, we first define $\hat{v_t}(s_t)$, which is another estimate of the value of state $s_t$. This estimate is equal to the value corresponding to taking the optimal decision $d_{opt}$ in state $s_t$, and can be found as follows:

$$\hat{v_t}(s_t) := max_{d \in \bar{D}}\, C(s_t, d) + \tilde{V}(Tr(s_t, d), \theta_{t+1}) \tag{38}$$

We consider $\hat{v_t}(s_t)$ to be a more accurate estimate, as it contains the actual revenue of the next interval, and the value function estimate considers one less interval. Therefore, to update the weights $\theta_{f,t}$ we wish to move the estimate $V(s_t, \theta_t)$ toward $\hat{v_t}(s_t)$. For this we use the following formula to find the new weight $\theta^p_{f,t}$ for basis function $i$ at time $t$ in path $p$, where $\theta^{p-1}_{f,t}$ represent the old weights, and $\theta^{p-1}_t$ the vector of old weights $(\theta^{p-1}_{1,t}, \theta^{p-1}_{2,t}, ..., \theta^{p-1}_{F,t})$:

$$\theta^p_{f,t} = \theta^{p-1}_{f,t} - \alpha_p(\tilde{V}(s_t, \theta^{p-1}_t) - \hat{v_t}(s_t)) \cdot \nabla_{\theta_t}(\tilde{V}(s_t, \theta^{p-1}_t)). \tag{39}$$

In this formula, $\alpha_p$ is a small number that may depend on the iteration $p$. The weight change is proportional to the gap between the two values $\tilde{V}(s_t, \theta^{p-1}_t) - \hat{v_t}(s_t)$. This ensures that the bigger the gap is, the more the weights are changed, and also that the change is made in the right direction. The gradient ensures that the weight change is proportional to the derivative of the value function, so that weights that have more effect are changed the most.

Note that there is no relation between the size of the weights and the size of the value functions and the gradients. Therefore, we should choose $\alpha_p$ in such a way that the product $(\tilde{V}(s_t, \theta^{p-1}_t) - \hat{v_t}(s_t)) \cdot \nabla_{\theta_t}(\tilde{V_t}(s_t, \theta^p_t))$ is scaled into the same order of magnitude of $\theta^{p-1}_{f,t}$.

Also, we want to have some convergence in the value function, so that the value of being in a certain state does not deviate too much in different runs, even if $\tilde{V_t}(s_t, \theta_t)$ does not converge to $\hat{v_t}(s_t)$ for all states $s_t$ in phase $t$. To obtain this convergence, we let the weights $\theta_{i,t}$ converge by letting the update factor $\alpha_p$ approach 0 over time. We do this by setting $\alpha_p := \frac{1}{p} \cdot \alpha$, where $\alpha$ is a model parameter that depends on the instance. We have chosen the multiplication with $\frac{1}{p}$ because this approaches 0 as $p \to \infty$, so that $\theta^p_{f,t}$ converges, and the value function $\tilde{V}(s_t, \theta_t)$ with it. Also, the sum of the harmonic series diverges, so that $\sum_{p=1}^{\infty} a_p = \infty$. With this property in mind, we expect that this method of decreasing $\alpha_p$ does not prevent the $\theta^p_{f,t}$ from converging to good values.

As we can observe from considering the value function (33) that $\nabla_{\theta_t}(\tilde{V_t}(s_t, \theta^{p-1}_t))$ is simply a vector of $\phi_f(s_t)$, we can rewrite equation (39) to:

$$\theta^p_{f,t} = \theta^{p-1}_{f,t} - \frac{1}{p}\alpha(\tilde{V}(s_t, \theta^{p-1}_t) - \hat{v_t}(s_t)) \cdot \phi_f(s_t) \tag{40}$$

Now we have enough information to describe an iteration in the micro-CHP approach, as is done in Algorithm 3.

**Algorithm 3** An iteration of our ADP approach to the micro-CHP problem

**for all** $p$ from 1 to $P$ **do**
  Set $t := 0$ and current state $s := s_0$
  **while** $t \leq T$ **do**
    Find $R(i)$ for all houses $i$ and sort the houses to obtain $\tilde{D}$.
    Find $d_{opt}$ as in (37) and $\hat{v}_t(s_t)$ as in (38).
    Update $\theta_{f,t}$ using (40).
    Set $s := Tr(s, d_{opt})$ and $t := t + 1$.
  **end while**
**end for**

The entire ADP approach now consists a series of $\bar{P}$ iterations, during which the best value and solution are remembered.

# 5 Alternative approaches

As mentioned before, in Bosman [4] a number of alternative methods have been presented. In this section we will present a short overview. First of all, two exact methods were examined. These were the ILP approach, and the exact DP approach. Also two other heuristics are presented: DP-based local search and a column generation method.

## 5.1 ILP approach

In the ILP approach, the constrained optimisation problem (9) mentioned in Section 2, is transformed into an ILP. In this method the $a_{i,t}$ variables were not used; instead the variables $start_{i,t}$ and $stop_{i,t}$ are introduced, which are binary variables indicating if a run of the micro-CHP in house $i$ was started or stopped in interval $t$. Using these variables, the heat and electricity generation can also be found in a linear setting, so that an ILP is found. This ILP was entered in the CPLEX solver, and was able to solve most of the smaller instances tested.

These instances contained 24 planning intervals up to 10 houses, for which different scenarios for the electricity bounds were tested. We also used these instances to test our micro-CHP approach to, and they are explained in more detail in the Results section. With the ILP approach, as the number of houses grew, the computational time increased quite dramatically. In the cases up to 6 houses, all tested instances were solved within 5 minutes, but from 7 houses on, this could go well over an hour, and in some of the cases with 9 and 10 houses, the solver terminated and the optimality of the solution could not be determined. In 3 cases, a better solution was found in the exact Dynamic Programming method described below.

## 5.2 Solving the Dynamic Program

Another exact method that was attempted was to simply solve the Dynamic Program, which, as mentioned before, is in general only possible for small instances. It was tested for the same small instances as the ILP approach, and it turned out that it was possible to solve cases up to 10 houses with this method. This was done using a SQL database to store the values. Both the time needed to find the solution and the memory needed to store the values increased exponentially in this method. In the cases with 10 houses, the database needed 15,1 GB of memory, and the running time exceeded 10 days. However, the optimal solution was found for all tested cases.

## 5.3 DP-based Local Search

As the exact methods could not be made to work for larger instances, the focus was shifted to heuristics. One of the tested heuristics was a local search method that consists of two steps. The first step uses the single house DP (as explained in paragraph 2.2), where for every house the optimal schedule is found. As the complexity for the single house DP is only $T^4$, this is possible.

In the second step these optimal schedules are combined, and the electricity prices are altered to give the houses more incentive to perform within the electricity bounds.

This is done by multiplying the electricity prices in interval $t$ by a factor $\alpha$, $(0 < \alpha < 1)$, if too much electricity was produced, and a factor $2 - \alpha$ if too little was produced. This was repeated until a maximum of iterations was reached, or until a feasible schedule was found.

This method was also tested for the smaller instances, and also some larger instances. More specifics of these instances, as well as the results of the tests (with $\alpha = 0.9$, which turned out to be the best tested value) can be found in the Results section.

## 5.4 Column Generation

Also, a column generation method was tested. This method also exists of two parts. First, for every house $i$, a set of feasible production schedules $F_i$ is chosen for the micro-CHP in house $i$. Then, in the main problem, an optimal solution is generated whereby only schedules in $F_i$ are allowed for house $i$. If the combined schedule is infeasible, because the global electricity constraint is violated, for all houses $i$ a subproblem is solved, that finds a good candidate for new schedules to be added in $F_i$.

In these subproblems, the goal is to find a feasible schedule for house $i$, whereby we want to produce less electricity in intervals where there is an excess production in the global schedule, and produce more if the global electricity production is insufficient. The schedule that fits this requirement the best is then added to $F_i$. This is done for all houses $i \in \{1,2,...,N\}$. This sequence of main and sub problems is repeated until no further improvement is found.

In the subproblem, two methods have been tested to find the best schedule. One method consisted of looking at the actual difference in electricity production, while the other only considered the on/off-status of the micro-CHP. The latter method was significantly faster, and also slightly better for the large instances, which is why we use those results to compare the results of the ADP approach to. The results for this method for both the smaller and the larger instances are shown in the Results section.

# 6 Results

In this section the results of applying Approximate Dynamic Programming are presented. For these results we used the parameters that were introduced by Bosman et al. in [4]. The algorithm presented in the previous section is applied to the data set. Anywhere in the results where a schedule is found for $n$ houses, the first $n$ houses of this set are intended.

The data set contains 200 houses with hourly heat demands between 500 $Wh$ and 4000 $Wh$ spread over the day, with peaks during the morning and the evening hours. The capacity of the buffer $L_{i,max}$ equals 10 $kWh$, and the maximum heat production $H_{max}$ is $8000W/h$. The prices of electricity varied between $19,01$ €/MWh to $500,00$ €/MWh. The maximum electricity production is $1000W/h$, so that the maximum revenue per interval per house equals €$0,50$. The penalty $P$ for violation of the electricity bounds is set at €$1 \cdot N$, so that a feasible solution is always preferred over an infeasible solution.

In our ADP approach, we have chosen $k_3 = 0.001$ and $k_4 = 0.1$, and initialized $\theta^0_{f,t} := 1$ for all $f \in \{1,2,...,F\}, t \in \{1,2,...,T\}$.

## 6.1 Small instances

First of all, we considered several small instances, where we let the number of houses $N$ vary between 1 and 10. For the minimum and maximum electricity production several fixed percentages of the maximum electricity production were considered, see Table 4. In the tenth variant, the bounds were set to the tightest bounds in the cases 1-9 for which a feasible solution was found. The other parameters were kept constant for the different instances.

For each of these small instances we set the number of iterations to $\bar{P} = 200$, and we had $T_{on} = T_{off} = 1$, so the streak length does not restrain the set of feasible decisions. The total amount of intervals $T$ is 24, with the interpretation that the total time equals one day, so each interval represents an hour. For every instance, we have manually found a good value for $\alpha$, which, as it turned out, was very important for the algorithm. The importance of $\alpha$ and how we have selected it are described to more detail in the next paragraph.

The revenues corresponding to the solutions found are presented in Table 5. These results represent the total revenue per house in €. We also show the running times and the best choices found for $\alpha$.Hereby, we multiplied these with 100, to get them into a more convenient range. Also, the optimal values found in the exact DP approach by Bosman in [4] are presented. As can be seen, the algorithm seems to generally perform a little worse than the optimal solution, and in a total of 7 cases it does not find a feasible solution when there does exist one.

We also show the results generated by the heuristics used in Bosman [4]. The results for the DP-based local search method are shown in Table 6, and the results for the column generation method are given in Table 7. As we can see, both heuristics by Bosman almost always give better results in variants 1 and 2, but in the other variants the values are generally better. Looking at the errors, the ADP algorithm had a total of

| case | $\frac{E_{min}}{NH_{max}} \cdot 100\%$ | $\frac{E_{max}}{NH_{max}} \cdot 100\%$ |
|---|---|---|
| 1 | 0 | 100 |
| 2 | 0 | 90 |
| 3 | 0 | 80 |
| 4 | 0 | 70 |
| 5 | 0 | 60 |
| 6 | 0 | 50 |
| 7 | 0 | 40 |
| 8 | 10 | 100 |
| 9 | 20 | 100 |
| 10 | $20^*$ | $40^*$ |

Table 4: Electricity bounds variants

7 cases where no feasible solution was found, where the local search method and the column generation had respectively 12 and 15 instances with errors.

This seems very promising indeed, especially as we are most interested in the variants where bounds are tight, which is not the case in the first two variants. We should however note that the column generation method appears to get better as the number of houses increases. Looking only at the cases from 8 houses up, the column generation method shows 1 error-free run more, and the size of the errors is quite a lot smaller. It therefore could be that this method works better for larger cases, which are the cases we eventually want to consider. We do still see a large difference in the revenues; if we compare the instances with more than 8 houses where both heuristics find a feasible solution, the average difference in value is $0,100$.

Also note that the other heuristics both generate their results a lot faster than the ADP approach; the local search algorithm found all results within $0,3$ seconds, and the column generation method rarely exceeds 2 seconds to solve an instance. In the ADP approach, for the larger instances about 6 seconds were needed, which should perhaps be multiplied by a factor 7 or 8 to account for the number of attempts needed to optimize the value for $\alpha$.

## 6.2 The choice of $\alpha$

In finding the results in the previous paragraph, we have mentioned that we had to find the optimal value for $\alpha$. In this paragraph we zoom in on how this value was found, and show the effect on different values for $\alpha$ on the runs. In the pictures below the instance with $N = 9$ houses and the sixth case for the electricity bounds is used. The best value found was 0.974, which was found when $\alpha = 4$.

In Figure 17 we have shown the changes in value during the 200 iterations when $\alpha = 1$. We can see that there is only one shift in value, after 20 iterations. As all other values are exactly equal, we can safely assume that only two different paths were tested during this run. This seems very little, and the reason for this is that the weight changes, which are scaled by $\alpha$, were apparently too low to lead to other paths being

| N | variant | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $\alpha(\cdot 100)$ | 350,0 | 350,0 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 350,0 |
| | value | 1,126 | 1,071 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 1,071 |
| | time (s) | 0,343 | 0,345 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 0,345 |
| | optimal | 1,147 | 1,092 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 1,092 |
| 2 | $\alpha(\cdot 100)$ | 90,0 | 80,0 | 120,0 | 120,0 | 120,0 | 120,0 | xxx | xxx | xxx | 120,0 |
| | value | 1,225 | 1,197 | 0,996 | 0,996 | 0,996 | 0,993 | xxx | xxx | xxx | 0,993 |
| | time (s) | 0,683 | 0,620 | 0,681 | 0,637 | 0,672 | 0,622 | xxx | xxx | xxx | 0,622 |
| | optimal | 1,236 | 1,208 | 1,016 | 1,016 | 1,016 | 1,016 | xxx | xxx | xxx | 1,016 |
| 3 | $\alpha(\cdot 100)$ | 35,0 | 35,0 | 35,0 | 35,0 | 30,0 | xxx | xxx | xxx | xxx | 30,0 |
| | value | 1,189 | 1,189 | 1,102 | 1,102 | 0,905 | xxx | xxx | xxx | xxx | 0,905 |
| | time (s) | 0,978 | 0,960 | 0,957 | 0,947 | 0,970 | xxx | xxx | xxx | xxx | 0,970 |
| | optimal | 1,197 | 1,197 | 1,106 | 1,106 | 1,002 | xxx | xxx | xxx | xxx | 1,002 |
| 4 | $\alpha(\cdot 100)$ | 15,0 | 15,0 | 15,0 | 15,0 | 20,0 | 20,0 | xxx | 15,0 | 15,0 | 15,0 |
| | value | 1,173 | 1,096 | 1,072 | 1,107 | 0,985 | 0,985 | xxx | $1,042^{[1]}$ | $1,042^{[2]}$ | $0,971^{[3]}$ |
| | time (s) | 1,383 | 1,124 | 1,122 | 1,356 | 1,350 | 1,350 | xxx | 1,397 | 1,402 | 1,460 |
| | optimal | 1,183 | 1,164 | 1,128 | 1,114 | 1,021 | 1,021 | xxx | 1,009 | 1,009 | 0,949 |
| 5 | $\alpha(\cdot 100)$ | 10,0 | 10,0 | 15,0 | 15,0 | 10,0 | xxx | xxx | 10,0 | xxx | 15,0 |
| | value | 1,159 | 1,119 | 1,114 | 1,052 | 1,052 | xxx | xxx | 1,117 | xxx | 1,004 |
| | time (s) | 1,902 | 1,881 | 1,854 | 1,746 | 1,854 | xxx | xxx | 1,892 | xxx | 1,864 |
| | optimal | 1,164 | 1,149 | 1,120 | 1,060 | 1,060 | xxx | xxx | 1,118 | xxx | 1,023 |
| 6 | $\alpha(\cdot 100)$ | 6,0 | 6,0 | 6,0 | 6,0 | 6,0 | 6,0 | xxx | 6,0 | xxx | 5,0 |
| | value | 1,159 | 1,126 | 1,126 | 1,080 | 1,006 | 1,005 | xxx | 1,138 | xxx | 1,004 |
| | time (s) | 1,945 | 2,053 | 2,031 | 2,047 | 2,061 | 2,043 | xxx | 2,044 | xxx | 2,041 |
| | optimal | 1,163 | 1,150 | 1,130 | 1,092 | 1,048 | 1,027 | xxx | 1,139 | xxx | 1,021 |
| 7 | $\alpha(\cdot 100)$ | 4,5 | 4,5 | 4,5 | 3,0 | 5,0 | 4,0 | 5,0 | 5,0 | xxx | 4,0 |
| | value | 1,132 | 1,132 | 1,132 | 1,101 | 1,059 | 0,948 | 0,914 | 1,131 | xxx | 0,891 |
| | time (s) | 3,074 | 2,872 | 2,762 | 3,033 | 2,775 | 2,240 | 2,582 | 3,099 | xxx | 3,108 |
| | optimal | 1,156 | 1,145 | 1,137 | 1,109 | 1,069 | 0,972 | 0,925 | 1,150 | xxx | 0,924 |
| 8 | $\alpha(\cdot 100)$ | 3,5 | 3,5 | 3,5 | 5,0 | 4,0 | 5,0 | 3,0 | 4,0 | 4,0 | 5,0 |
| | value | 1,134 | 1,134 | 1,109 | 1,082 | 1,070 | 1,024 | 0,887 | 1,132 | $1,078^{[2]}$ | $0,894^{[2]}$ |
| | time (s) | 3,700 | 3,838 | 3,8 | 3,963 | 3,713 | 3,713 | 3,536 | 3,561 | 3.804 | 3,835 |
| | optimal | 1,156 | 1,145 | 1,130 | 1,114 | 1,080 | 1,032 | 0,919 | 1,152 | 1,069 | 0,902 |
| 9 | $\alpha(\cdot 100)$ | 3,0 | 3,0 | 3,0 | 3,0 | 4,5 | 4,0 | xxx | 3,0 | 3,0 | 3,0 |
| | value | 1,133 | 1,133 | 1,111 | 1,088 | 1,062 | 0,974 | xxx | 1,130 | $1,109^{[4]}$ | 0,948 |
| | time (s) | 4,427 | 4,449 | 4,522 | 4,666 | 4,452 | 4,6 | xxx | 4,671 | 4,602 | 4,698 |
| | optimal | 1,153 | 1,143 | 1,121 | 1,098 | 1,072 | 0,999 | xxx | 1,150 | 1,113 | 0,976 |
| 10 | $\alpha(\cdot 100)$ | 3,0 | 3,0 | 1,5 | 2,0 | 3,0 | 4,0 | 4,0 | 2,0 | 3,0 | 3,0 |
| | value | 1,150 | 1,150 | 1,129 | 1,108 | 1,090 | 1,024 | 0,940 | 1,151 | $0,979^{[5]}$ | 0,927 |
| | time (s) | 5,529 | 5,606 | 5,060 | 5,412 | 5,384 | 5,047 | 4,937 | 5,189 | 5,088 | 5,816 |
| | optimal | 1,176 | 1,162 | 1,143 | 1,122 | 1,095 | 1,044 | 0,956 | 1,173 | 1,028 | 0,945 |

[1] [2] [3] [4] [5]: no feasible solution found: error value respectively 700, 1500, 750, 1550, 3750

Table 5: Results for small instances

| # houses | variant | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | value | 1,147 | 1,092 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 1,092 |
| | time (s) | 0,015 | 0,015 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 0,016 |
| | error | 0 | 0 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 0 |
| 2 | value | 1,236 | 1,208 | 0,937 | 0,937 | 0,937 | 1,016 | xxx | xxx | xxx | 1,016 |
| | time (s) | 0,015 | 0,015 | 0,078 | 0,078 | 0,078 | 0,016 | xxx | xxx | xxx | 0,016 |
| | error | 0 | 0 | 200 | 400 | 600 | 0 | xxx | xxx | xxx | 0 |
| 3 | value | 1,197 | 1,197 | 1,097 | 1,097 | 0,863 | xxx | xxx | xxx | xxx | 0,863 |
| | time (s) | 0,015 | 0,015 | 0,016 | 0,016 | 0,031 | xxx | xxx | xxx | xxx | 0,031 |
| | error | 0 | 0 | 0 | 0 | 0 | xxx | xxx | xxx | xxx | 0 |
| 4 | value | 1,183 | 1,068 | 1,050 | 1,103 | 0,939 | 0,794 | xxx | 0,931 | 0,931 | 0,822 |
| | time (s) | 0,015 | 0,015 | 0,016 | 0,015 | 0,078 | 0,047 | xxx | 0,125 | 0,141 | 0,141 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | 1250 | 2850 | 1500 |
| 5 | value | 1,164 | 1,083 | 1,063 | 1,054 | 1,054 | xxx | xxx | 0,978 | xxx | 0,856 |
| | time (s) | 0,015 | 0,016 | 0,016 | 0,047 | 0,031 | xxx | xxx | 0,172 | xxx | 0,062 |
| | error | 0 | 0 | 0 | 0 | 0 | xxx | xxx | 400 | xxx | 0 |
| 6 | value | 1,163 | 1,096 | 1,096 | 1,092 | 0,967 | 0,940 | xxx | 0,925 | xxx | 0,942 |
| | time (s) | 0,015 | 0,015 | 0,016 | 0,031 | 0,047 | 0,063 | xxx | 0,203 | xxx | 0,141 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | 500 | xxx | 0 |
| 7 | value | 1,156 | 1,137 | 1,137 | 1,079 | 1,068 | 0,904 | 0,893 | 1,093 | xxx | 0,839 |
| | time (s) | 0,015 | 0,016 | 0,016 | 0,016 | 0,031 | 0,078 | 0,219 | 0,266 | xxx | 0,993 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | 900 | 0 | xxx | 150 |
| 8 | value | 1,156 | 1,138 | 1,087 | 1,087 | 1,073 | 0,875 | 0,838 | 1,151 | 0,986 | 0,881 |
| | time (s) | 0,016 | 0,016 | 0,016 | 0,016 | 0,031 | 0,063 | 0,218 | 0,016 | 0,266 | 0,219 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1050 | 7600 |
| 9 | value | 1,153 | 1,137 | 1,092 | 1,092 | 1,057 | 0,842 | xxx | 1,148 | 0,960 | 0,843 |
| | time (s) | 0,016 | 0,031 | 0,016 | 0,031 | 0,031 | 0,078 | xxx | 0,015 | 0,250 | 0,188 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | 0 | 650 | 0 |
| 10 | value | 1,176 | 1,161 | 1,098 | 1,098 | 1,094 | 0,963 | 0,871 | 1,170 | 0,968 | 0,849 |
| | time (s) | 0,016 | 0,016 | 0,016 | 0,015 | 0,031 | 0,109 | 0,109 | 0,031 | 0,297 | 0,297 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2100 | 6850 |

Table 6: Results for local search method in Bosman [4]; small instances

| # houses | variant | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | value | 1,147 | 1,050 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 1,050 |
| | time (s) | 0,00 | 0,00 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 0,00 |
| | error | 0 | 0 | xxx | xxx | xxx | xxx | xxx | xxx | xxx | 0 |
| 2 | value | 1,236 | 0,890 | 0,995 | 0,995 | 0,995 | 0,995 | xxx | xxx | xxx | 0,995 |
| | time (s) | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0 | xxx | xxx | xxx | 0 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | xxx | xxx | 0 |
| 3 | value | 1,197 | 1,197 | 0,899 | 0,899 | 1,017 | xxx | xxx | xxx | xxx | 1,017 |
| | time (s) | 0,00 | 0,00 | 0,00 | 0,00 | 0,25 | xxx | xxx | xxx | xxx | 0,25 |
| | error | 0 | 0 | 0 | 0 | 150 | xxx | xxx | xxx | xxx | 150 |
| 4 | value | 1,183 | 0,892 | 1,090 | 1,000 | 0,883 | 0,984 | xxx | 0,825 | 0,914 | 0,946 |
| | time (s) | 0,00 | 0,00 | 0,25 | 0,26 | 0,26 | 0,26 | xxx | 0,53 | 0,53 | 0,52 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | 700 | 1500 | 1450 |
| 5 | value | 1,164 | 0,863 | 0,902 | 0,941 | 0,941 | xxx | xxx | 1,056 | xxx | 0,942 |
| | time (s) | 0,27 | 0,26 | 0,26 | 0,25 | 0,25 | xxx | xxx | 0,52 | xxx | 0,78 |
| | error | 0 | 0 | 0 | 0 | 0 | xxx | xxx | 400 | xxx | 0 |
| 6 | value | 1,163 | 0,999 | 0,938 | 1,041 | 0,917 | 0,868 | xxx | 0,941 | xxx | 0,911 |
| | time (s) | 0,53 | 0,26 | 0,52 | 0,26 | 0,52 | 0,26 | xxx | 0,53 | xxx | 0,8 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | 500 | xxx | 450 |
| 7 | value | 1,156 | 1,080 | 0,964 | 0,841 | 1,015 | 0,915 | 0,920 | 0,929 | xxx | 0,927 |
| | time (s) | 0,25 | 0,52 | 0,53 | 0,53 | 0,52 | 0,80 | 1,56 | 0,26 | xxx | 1,04 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | xxx | 250 |
| 8 | value | 1,154 | 1,041 | 0,938 | 0,831 | 0,957 | 0,894 | 0,876 | 0,945 | 0,977 | 0,878 |
| | time (s) | 0,25 | 0,53 | 0,53 | 0,51 | 0,53 | 0,53 | 1,33 | 0,51 | 1,06 | 3,98 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 550 |
| 9 | value | 1,151 | 1,018 | 1,050 | 0,915 | 0,931 | 0,909 | xxx | 1,003 | 1,046 | 0,905 |
| | time (s) | 0,26 | 0,53 | 0,52 | 0,51 | 0,53 | 0,53 | xxx | 0,51 | 0,78 | 1,33 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | xxx | 0 | 0 | 0 |
| 10 | value | 1,174 | 1,010 | 1,010 | 0,832 | 0,900 | 0,930 | 0,911 | 1,107 | 0,886 | 0,876 |
| | time (s) | 0,26 | 0,53 | 0,53 | 0,53 | 0,80 | 0,53 | 1,34 | 0,52 | 6,88 | 4,54 |
| | error | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 50 | 50 |

Table 7: Results for column generation method in Bosman [4]; small instances

found. The value this run converged to, which is around 0.971, is also a little lower than the optimum value.

When $\alpha$ is set to a higher value, such as $\alpha = 7$ in Figure 18, we can observe a lot of jumps initially, not always finding better solutions. In fact, between run 7 and 13 all runs were infeasible, as an electricity constraint was violated. These are depicted as values of 0.91. Often, the weights are changed severely altered when an infeasibility is found, because there is a large difference between the estimated and the realized value. Once an infeasibility is encountered during an iteration, it becomes much more difficult to find good solutions. Here it can be seen as well that the algorithm settles for a value of 0.949, severely lower than the highest value found, which was 0.965 in run 4.

It seems to be obvious that the optimal value for $\alpha$ can be found in between these values of 0.1 and 0.7, so that the weight changes is not so large as to cause infeasibilities, and not so small so that the weights are barely changed. In 19 the values for $\alpha = 4$, the best found value for $\alpha$, are shown. It can be seen that the highest value found is about 0.974, reached in the 6th iteration. After 13 runs the value does not change anymore, and the value appears to have reached convergence at a value of 0.949. It happens quite often that the algorithm does not stabilize at the highest value found.

This example is very typical for the results in the previous paragraph; for too small values of $\alpha$ the algorithm found only few improvements over the initial value, and for too large values infeasible solutions were found and the algorithm does not recover from that.

Therefore, we can conclude that the choice of $\alpha$ is important. In the results presented in this section, every instance was run several times, during which the choice of $\alpha$ was changed accordingly. This was repeated until a reasonable choice for $\alpha$ was found. In doing this we only considered choices for $\alpha$ of the type $x \cdot 10^y$ where $x \in \{1, 1.2, 1.5, 1.7, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7, 8, 9\}$ and $y \in \mathbb{Z}$ and optimized within this range. For the results presented in this section, this was done manually. This was partly due to time constraints, and partly because it is difficult to formalize this process algorithmically. The typical process consisted of starting with choices of $\alpha$ around an optimum for a similar case, until 'too many' infeasible runs were found, or the value strayed 'too far' from the highest value found, followed by searching near the choices leading to the highest values.

On average about 7 or 8 choices were tested for $\alpha$ in every instance. In a large fraction (around 40 %) of the cases tested, the maximum value was found in the first run, making the choice of $\alpha$ irrelevant.

## 6.3 Large instances

We also tested some large instances with 25, 50, 75 and 100 houses. These instances were also tested with different numbers of time intervals; 24, 48 and 96. In all these cases the same day is considered, but in the 24 intervals case each time interval represents one hour, with 48 intervals half an hour, and with 96 intervals every interval represents 15 minutes. In the first two cases, the minimum run time $T_{on}$ and off time $T_{off}$ were 1 interval, so the minimum and maximum run times do not restrict the feasible decision space. With 96 intervals, $T_{on} = T_{off} = 2$, as the minimum running time and the minimum off time were defined to be 30 minutes.

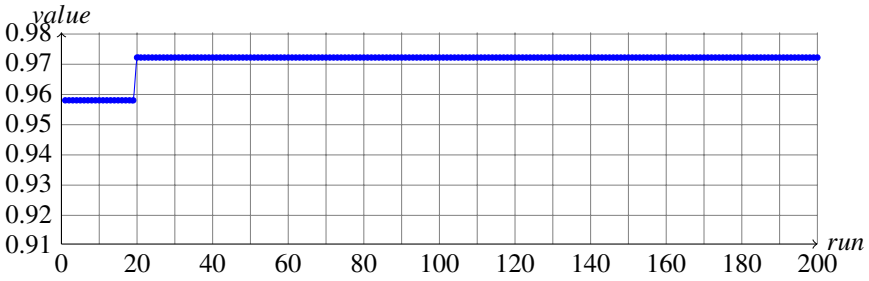



Figure 17: Value changes during run where $\alpha = 1$

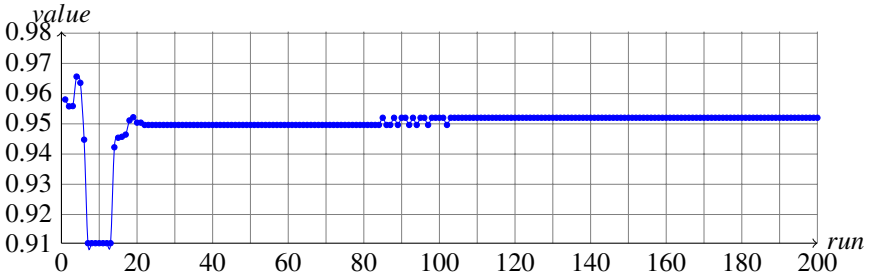



Figure 18: Value changes during run where $\alpha = 7$

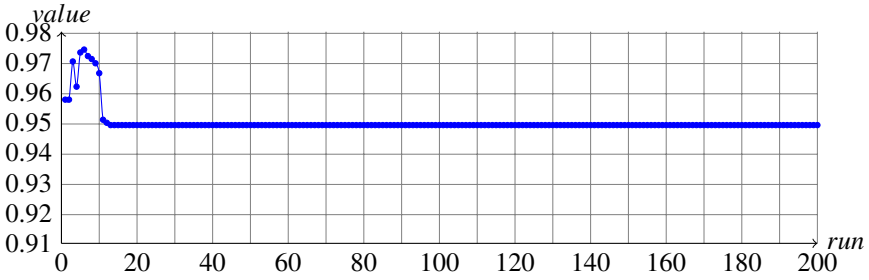



Figure 19: Value changes during run where $\alpha = 4$

| N: | Elec bounds: | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
|---|---|---|---|---|---|
| 25 | $\alpha \cdot (100)$ | 0,2 | 0,35 | 0,35 | 0,35 |
| | value | 1,09992 | 1,03603 | 0,94359 | 0,90164 |
| | error | 0 | 1950 | 4450 | 5700 |
| | run time | 6,923 | 7,535 | 6,737 | 6,203 |
| 50 | $\alpha(\cdot 10^{18})$ | 0,15 | 0,15 | 0,15 | 0,15 |
| | value | 1,098180 | 1,036690 | 0,981824 | 0,914132 |
| | error | 0 | 4000 | 9000 | 16950 |
| | run time | 20,049 | 23,492 | 25,354 | 20,562 |
| 75 | $\alpha(\cdot 10^{18})$ | 0,05 | 0,05 | 0,05 | 0,03 |
| | value | 1,10203 | 1,03613 | 0,96678 | 0,90454 |
| | error | 0 | 5950 | 12250 | 44650 |
| | run time | 45,183 | 42,294 | 41,121 | 40,923 |
| 100 | $\alpha(\cdot 10^{18})$ | 0,015 | 0,01 | 0,01 | 0,03 |
| | value | 1,09378 | 1,04354 | 0,96971 | 0,89921 |
| | error | 0 | 8000 | 18000 | 34750 |
| | run time | 65,771 | 67,471 | 72,644 | 66,239 |

Table 8: Large instances; 24 intervals

In these cases we only let the algorithm for $\bar{P} = 50$ iterations, as each run took a lot longer than it did with the smaller instances. The results the algorithm gave for these instances are shown in Tables 8, 9 and 10. We can see that no errors occurred in the cases with 48 intervals, while there is a large error in the case with 96 intervals.

This seems strange, as the solutions found for 48 intervals are also solutions for the case with 96 intervals. However, apparently it is much harder to find these error-free solutions in the case with 96 intervals. This is likely due to the fact that with 96 intervals, the minimum off time and the minimum on time are 2 intervals, where they were 1 interval in the situation with 48 intervals. Therefore, at a typical state, the number of feasible decisions is smaller, because some of the houses have been turned on the previous interval, and so their state must remain unchanged. This could make it it more difficult to find a feasible schedule for the following two intervals, and as a consequence the error is higher.

To get more insight in the influence of the different data sets to this algorithm, we have written the average value for every parameter in Table 11. This means that for the first entry in the table, we took the average value (and error and runtime) for all results where 24 intervals were used.

From this table we can see that the running times depend mostly on the number of houses and the numbers of intervals, and both increase faster than linear. For the intervals, this can be explained by the fact that in every interval, finding $\phi_1(s_t)$ requires a sum over every interval to be taken, and since this has to be done in every interval, this gives us a $T^2$-term in the computation time. For the houses we note that in reducing the decision space the houses are sorted on their $R(i)$-value, which takes $N^2$ time, as the sorting algorithm is not most efficient.

We also see that the value decreases as the electricity bounds get tighter. Obviously,

| N: | Elec bounds: | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
|---|---|---|---|---|---|
| 25 | $\alpha(\cdot 100)$ | 0,5 | 0,7 | 0,7 | 0,2 |
| | value | 1,25431 | 1,10719 | 0,96935 | 0,87922 |
| | error | 0 | 0 | 0 | 0 |
| | run time | 19,101 | 21,143 | 17,015 | 16,969 |
| 50 | $\alpha(\cdot 100)$ | 0,1 | 0,1 | 0,1 | 0,15 |
| | value | 1,25406 | 1,08606 | 0,94669 | 0,90723 |
| | error | 0 | 0 | 0 | 0 |
| | run time | 46,004 | 48,704 | 48,471 | 49,250 |
| 75 | $\alpha(\cdot 100)$ | 0,07 | 0,07 | 0,07 | 0,07 |
| | value | 1,25230 | 1,10215 | 0,99048 | 0,89273 |
| | error | 0 | 0 | 0 | 0 |
| | run time | 95,661 | 96,506 | 102,711 | 95,581 |
| 100 | $\alpha(\cdot 100)$ | 0,03 | 0,03 | 0,03 | 0,03 |
| | value | 1,23980 | 1,09577 | 0,96083 | 0,89756 |
| | error | 0 | 0 | 0 | 0 |
| | run time | 171,697 | 163,489 | 172,661 | 165,142 |

Table 9: Large instances; 48 intervals

| N: | Elec bounds: | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
|---|---|---|---|---|---|
| 25 | $\alpha(\cdot 100)$ | 0,35 | 0,1 | 0,1 | 0,1 |
| | value | 1,22444 | 1,04120 | 0,92303 | 0,84896 |
| | error | 0 | 0 | 1900 | 14100 |
| | run time | 38,444 | 41,589 | 39,513 | 46,328 |
| 50 | $\alpha(\cdot 100)$ | 0,1 | 0,1 | 0,1 | 0,1 |
| | value | 1,22704 | 1,05646 | 0,92474 | 0,85964 |
| | error | 0 | 0 | 6600 | 11950 |
| | run time | 132,257 | 136,001 | 130,54 | 143,61 |
| 75 | $\alpha(\cdot 100)$ | 0,01 | 0,01 | 0,01 | 0,01 |
| | value | 1,21518 | 1,05564 | 0,90533 | 0,89114 |
| | error | 0 | 0 | 12100 | 84425 |
| | run time | 278,415 | 275,164 | 276,683 | 261,058 |
| 100 | $\alpha(\cdot 100)$ | 0,005 | 0,005 | 0,005 | 0,005 |
| | value | 1,23483 | 1,0578 | 0,92361 | 0,89593 |
| | error | 0 | 1650 | 16600 | 103100 |
| | run time | 542,231 | 485,194 | 464,913 | 468,786 |

Table 10: Large instances; 96 intervals

| # of intervals: | 24 | 48 | 96 | |
|---|---|---|---|---|
| value | 1,002 | 1,052 | 1,018 | |
| error | 10353 | 0 | 15777 | |
| run time (s) | 35 | 83 | 235 | |
| Elec bounds | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
| value | 1,191 | 1,063 | 0,950 | 0,891 |
| error | 0 | 1796 | 6742 | 26302 |
| run time (s) | 122 | 117 | 117 | 115 |
| # of houses | 25 | 50 | 75 | 100 |
| value | 1,019 | 1,024 | 1,026 | 1,026 |
| error | 2342 | 4042 | 13281 | 15175 |
| run time (s) | 22 | 69 | 138 | 242 |

Table 11: Average results with one parameter fixed

| # of intervals: | 24 | 48 | 96 | |
|---|---|---|---|---|
| value | 0,953 | 1,023 | 1,103 | |
| error | 27163 | 39252 | 74162 | |
| run time (s) | 1 | 243 | 7053 | |
| Elec schedule | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
| value | 1,165 | 0,971 | 0,984 | 0,984 |
| error | 0 | 9340 | 65654 | 112440 |
| run time (s) | 859 | 2951 | 2962 | 2958 |
| # of houses | 25 | 50 | 75 | 100 |
| value | 1,007 | 1,026 | 1,040 | 1,031 |
| error | 20588 | 36063 | 59734 | 71050 |
| run time (s) | 1048 | 1982 | 2869 | 3831 |

Table 12: Results for local search method by Bosman; large instances

tighter bounds lead to lower production when the price is highest, so the value is lower there. Also, the error increases in this case as these tighter bounds are more difficult to satisfy. Beside this, we can also observe an increase in the error as the number of houses becomes larger. With more houses, of course larger errors are possible.

The results of these methods are provided in Table 12 and Table 13. These results are aligned in a similar way as in Table 11.

Here we can see that the local search method has significantly higher errors than those obtained by the ADP approach. If we compare the values found, we see that they appear to be close: for the case with 96 houses and with tighter electricity bounds, the local search method performs better, while the ADP approach performs better in the other cases. On average, the local search method has slightly higher values (1,026 vs 1,024), but this is clearly offset by the different in error sizes.

Comparing the results to the column generation method, we see that the values for the ADP approach are higher across the table. The average value found in the column generation method is 0,897, 0,127 lower than in the ADP approach. However, the error

| # of intervals: | 24 | 48 | 96 | |
|---|---|---|---|---|
| value | 0,969 | 0,859 | 0,864 | |
| error | 7141 | 19 | 224 | |
| run time (s) | 39 | 181 | 277 | |
| Elec schedule | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
| value | 0,846 | 0,986 | 0,899 | 0,86 |
| error | 0 | 1283 | 3367 | 5194 |
| run time (s) | 57 | 77 | 116 | 412 |
| # of houses | 25 | 50 | 75 | 100 |
| value | 0,898 | 0,884 | 0,901 | 0,907 |
| error | 950 | 1919 | 3059 | 3917 |
| run time (s) | 105 | 128 | 192 | 237 |

Table 13: Results for column generation method by Bosman; large instances

is typically smaller in the column generation method. We should note here that most of the values in the original table are 'contaminated' in a way with the values of the case with 96 intervals, where the errors found are 224 and 15777 for respectively the column generation method and the ADP approach.

Looking closer at the cases for different numbers of houses, we see that in the column generation method the errors are smaller in the cases with 24 intervals. However, in the instances with 48 intervals some (small) errors were found, which was not the case in the ADP-approach. The time needed was also significantly smaller in the column generation method.

## 6.4 Basis functions

To check the usefulness of the basis functions we selected, we also decided to run the algorithm without each of the basis functions. In every run a different basis function was left out. The case with 24 intervals and 25 houses was used for the test, where the other parameters were set equal to the ones in the previous run. The results of this test are shown in Table 14. As our algorithm only gives penalties when a constraint is violated, and no regard is given to the size of the violation, constraint violations are more important to our algorithm than error sizes. Usually, this is no real issue, as the goal is to find error-free intervals. However, for this part it is relevant, so we also show the number of intervals in which constraints were violated in the optimal solution as '# of violated intervals'.

Here we can see that the first two basis functions indeed seem to be important; without these we can observe lower values and higher error rates. This indicates that these two basis functions significantly contribute to the algorithm.

The last two basis functions generate about equal results, and even a higher value can be found when the third basis function is left out in the $20-40\%$ situation. This may be caused by the fact that these basis functions are restraints that are supposed to lead to more options, where a lower score in the next interval is accepted in order to obtain higher flexibility later. In some cases, this higher flexibility will not be relevant.

| Without bf: | Elec bounds: | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
|---|---|---|---|---|---|
| none | $\alpha(\cdot 100)$ | 0,2 | 0,35 | 0,35 | 0,35 |
| | value | 1,09992 | 1,03603 | 0,94359 | 0,90164 |
| | error | 0 | 1950 | 4450 | 5700 |
| | run time | 6,923 | 7,535 | 6,737 | 6,203 |
| | # of violated intervals | 0 | 1 | 1 | 1 |
| 1 | $\alpha(\cdot 100)$ | 0 | 0 | 0 | 0 |
| | value | 0,907178 | 0,903136 | 0,908267 | 0,913903 |
| | error | 0 | 1050 | 5500 | 11050 |
| | run time | 4,321 | 4,649 | 4,617 | 4,789 |
| | # of violated intervals | 0 | 1 | 2 | 3 |
| 2 | $\alpha(\cdot 100)$ | 0,2 | 0,45 | 0,25 | 0,35 |
| | value | 1,09992 | 1,03657 | 0,94418 | 0,90879 |
| | error | 0 | 1050 | 4450 | 11550 |
| | run time | 4,165 | 4,119 | 4,134 | 4,04 |
| | # of violated intervals | 0 | 1 | 1 | 2 |
| 3 | $\alpha(\cdot 100)$ | 0,2 | 0,35 | 0,35 | 0,25 |
| | value | 1,09992 | 1,03603 | 0,94400 | 0,90164 |
| | error | 0 | 1950 | 3550 | 5700 |
| | run time | 4,477 | 4,352 | 4,399 | 4,54 |
| | # of violated intervals | 0 | 1 | 1 | 1 |
| 4 | $\alpha(\cdot 100)$ | 0,2 | 0,35 | 0,35 | 0,35 |
| | value | 1,09992 | 1,03603 | 0,94359 | 0,90164 |
| | error | 0 | 1950 | 4450 | 5700 |
| | run time | 5,21 | 6,226 | 4,992 | 6,123 |
| | # of violated intervals | 0 | 1 | 1 | 1 |

Table 14: 24 intervals with one basis function left out

In some of the other cases, we did observe that the interval for choices of $\alpha$ in which the optimal solution was found became smaller. This indicates that our basis functions also carry some positive effects.

However, the contribution of these last two basis functions remains questionable, especially as the running time is also decreased significantly if they are not used.

## 6.5 Randomness

We also considered a possibility where we introduced some randomness into the algorithm. Currently we always take the decision with the highest value. Yet, in the taxicab problem we have already seen that not always taking the highest value decision leads to a larger range of states being reached, which in some cases leads to a better solution.

Although this problem is very different from the taxicab problem, we nevertheless tried a similar approach here. Therefore, instead of always taking the solution with the highest value, for the results in the next table we have taken with a probability of 5% the second best decision, if this decision is feasible.

| N | Elec bounds: | 0 - 75 % | 10 - 50 % | 20 - 40 % | 25 - 35 % |
|---|---|---|---|---|---|
| 25 | deterministic value | 1,09992 | 1,03603 | 0,94359 | 0,90164 |
| | best value using randomness | 1,10106 | 1,03843 | 0,94563 | 0,90547 |
| | error | 0 | 1950 | 4450 | 5700 |
| | run time | 43,996 | 45,271 | 37,3 | 46,348 |
| | # of violated intervals | 0 | 1 | 1 | 1 |
| 50 | deterministic value | 1,098180 | 1,036290 | 0,981824 | 0,914132 |
| | best value using randomness | 1,098610 | 1,036720 | 0,982695 | 0,915428 |
| | error | 0 | 4000 | 9000 | 16950 |
| | run time | 145,424 | 144,082 | 142,007 | 156,515 |
| | # of violated intervals | 0 | 1 | 1 | 2 |
| 75 | deterministic value | 1,10203 | 1,03613 | 0,96678 | 0,90454 |
| | best value using randomness | 1,10219 | 1,03736 | 0,97046 | 0,91131 |
| | error | 0 | 5950 | 12250 | 42850 |
| | run time | 305,433 | 314,621 | 300,191 | 313,958 |
| | # of violated intervals | 0 | 1 | 1 | 3 |
| 100 | deterministic value | 1,09372 | 1,04354 | 0,96971 | 0,89921 |
| | best value using randomness | 1,09397 | 1,04375 | 0,97313 | 0,90304 |
| | error | 0 | 8000 | 18000 | 39150 |
| | run time | 524,208 | 534,785 | 532,507 | 538,028 |
| | # of violated intervals | 0 | 1 | 1 | 3 |

Table 15: 24 intervals with random decisions

We have tested this for all cases with 24 intervals using the same values for $\alpha$ in the deterministic case. All other parameters were also all kept the same. As a different run can lead to a different result in this case (if the random seed is different), this algorithm was run a total of 10 times, where the best result was taken.

The results of this test are shown in Table 15. As we can see, in all cases we have tried lead to an improvement. In two cases the improvement did have an higher error, but the number of violated intervals remained the same. This means that the algorithm does not see this higher error, so these cases should be considered improvements as well.

Note that the improvements found are typically not too large, and for obvious reasons this algorithm takes about 10 times longer than the original. Also, in many of the runs no improvement was found, and the result found was actually worse than the original result. Still, for every case in at least one of the ten runs an improvement over the deterministic setting was found.

# 7 Conclusion and future work

In this thesis we have described an Approximate Dynamic Programming approach to the micro-CHP scheduling problem. This was done using the Dynamic Programming architecture for this problem that was described by Bosman [4]. Here the planning horizon was discretized in several intervals, so that in every interval the state of a micro-CHP remained unchanged. Then, there is only a finite combination of possible positions for the micro-CHPs in the different houses.

Our ADP algorithm translated such a combination into a value by using an estimated value function, that uses several features of the current positions of the houses. These features are converted into numbers, of which then a weighted sum is taken. This sum is used as an approximation of the value, and can be used to find a decision strategy. Using the decisions taken and considering the revenues received using this path, the weights are updated to form a better approximation.

This approach showed promising results, and withstood the comparison with the results presented in Bosman [4] well. Compared to the local search method, the errors found were significantly lower, and the values were approximately equal.

The comparison with the column generation method is more difficult. The ADP approach was slower and in most cases showed larger errors, but the value found was significantly higher. It also became clear, the approach appears to be some difficulties if the instance has a minimum on time or a minimum off time which of more than 1 interval.

Yet, there are many different implementations of ADP possible for this problem, and here we only tested one approach (and some small variations thereof). For now, it at least seems to be an interesting addition to the spectrum of heuristics for this problem, also because it uses an entirely different approach. While the other heuristics tried to make schedules for individual houses, which were merged together, in this approach the scheduling is done time-based, so in a given interval a decision is made for all houses.

To find improvements, the following changes to the chosen approach are worth investigating:

- In generating the results, the optimal values for $\alpha$ are currently found manually, which is quite labour intensive and uses a lot of time. It would be better to make this a part of the algorithm, although some care is required to simulate the current optimization process.

- We could have chosen another reduction of the decision space: we have in our approach decided to sort the houses on how long they could be left turned off. We have already seen that this may cause difficulties when the minimum off time and minimum on time are more than one interval. Therefore it could be better to consider methods where the current on/off status is considered in making the decision for the next interval.

- We can also consider other basis functions in the value function. As we have seen in the Results section, the last two basis functions we have chosen do not

seem to contribute much to the solution. Therefore, it seems to be a good idea to replace these with better basis functions, if these can be found.

- Updating the weights could be done in a different way. We have now chosen a gradient method, but in literature also a Kalman filter is mentioned. Also, the gradient method used is sort of counter-intuitive, as there is no direct relation between the weights and the size of the weight change. Therefore, this method could be adapted by introducing such a relation, for example by changing the weights in such a way that a certain percentage of the gap between the value estimates is closed. Another possibility for changing the updates of the weights is not just considering the estimate in the value function in the next step, but also the value that is eventually reached, or perhaps other combinations of value estimates and revenues reached in between.

- In the standard algorithm, we assume that the decision that gives the highest estimated revenue is always taken. This appears to be obvious, but in multiple paths this often leads to the same path being taken again and again. Therefore, it could be a good idea to not always choose the optimal decision. In the Results section we have already seen that a 5 % probability of choosing the second best decision improves the results in all of the tested cases. This indicates that there is some room for improvement here. Yet, the question remains what the best way is to deviate from the optimal decision.

- As mentioned in the Results section, in considering the penalties, we currently only observe whether a constraint has been violated. We do not consider by how much it has been violated, as we focus on finding an error-free schedule. We could also penalize the size of the error, i.e. adding a penalty cost per error point, but that would imply some relation between the errors and the revenue, making comparison between results. Adding a very large penalty per error point, so that a solution with smaller error is always chosen, is also an option. This would however lead to higher penalties if an infeasibility is encountered, which could cause the weights to over-correct, making it harder to find feasible paths in future iterations. Yet, if there is a clear idea of what infeasibilities are preferable, such as a desire to minimize the error, some adjustments can be made.

- Choosing a factor of $\frac{1}{p}$ in changing the weights in the value function could make the first iterations too powerful compared to the others. In this setting the first step is a factor 10 larger than the tenth step, which is quite a big jump. Perhaps it would be more appropriate to change the $\frac{1}{p}$ into e.g. a $\frac{1}{p+9}$-term, and then increase $\alpha$ by a factor 10. This would make sure that the first step is of the same size, but here the step size decreases much slower. Of course, this also implies that the final steps are also larger, and so the final weights will have had less opportunity to stabilize.

# References


[1] Bosman, M.G.C. and Bakker, V. and Molderink, A. and Hurink, J.L. and Smit, G.J.M. *On the microCHP scheduling problem.* In: Proceedings of the 3rd Global Conference on Power Control and Optimization PCO, 2010, 2-4 Feb 2010, Gold Coast, Australia. pp. 367-374. AIP Conference Proceedings 1239. PCO.

[2] Powell, Warren B. *Approximate Dynamic Programming: solving the curses of dimensionality* John Wiley & Sons, 2007.

[3] Terry James, (MRes student, University of Lancester). *Approximate Dynamic Programming.* `www.lancs.ac.uk/~jamest/content/ADP_Terry.pdf` May 24, 2011

[4] Bosman, M.G.C. *Planning in Smart Grids* Thesis, PhD, University of Twente.

[5] NPC Global Oil & Gas Study, `http://www.npc.org/study_topic_papers/4-dtg-electricefficiency.pdf` July 18, 2007