Planning the Charging of an Electric Vehicle with a Minimum Run-time Constraint

by
Eveline Koster

University of Twente
2023

Faculty: Electrical Engineering, Mathematics and Computer Science
Master Programme: Applied Mathematics
Specialization: Operations Research
Chair: Discrete Mathematics and Mathematical Programming

Graduation Date: 08/03/2023
Assessment Committee:
Prof. Dr. J.L. Hurink
Dr. M. Schoot Uiterkamp
Dr. Ir. W. Scheinhardt

Planning the Charging of an Electric Vehicle with a Minimum Run-time Constraint

Eveline Koster

Abstract

With the recent changes in energy production and energy consumption comes a need for change in the management strategy for the electricity grid. One of the changes in consumption is caused by the increase in the number of Electric Vehicles (EV's) in need of charging. The charging behaviour of EV's has been extensively studied in literature, and efficient algorithms to calculate optimal charging schemes have been developed. This thesis introduces a new constraint for a single-device EV charging model, namely the constraint of a minimum run-time. We present both a discrete model with a minimum run-time constraint, and two potential continuous charging models, of which one is selected. For the discrete problem we develop a dynamic programming algorithm, based on another algorithm found in existing literature. We then analyse the structure of the continuous problem, and a derived result is used to develop an exact algorithm for this problem as well. This algorithm is found to be rather inefficient in terms of speed and memory-usage. Therefore, we proceed to use the same structure of the problem to design a local search approximation method. We utilize several properties of the specific problem we consider here to improve the efficiency of the local search algorithm. This results in a fast polynomial-time algorithm for the continuous problem.

Dankwoord

Het heeft een lange tijd gekost om de master thesis die u nu voor u heeft tot voltooiing te brengen, maar het resultaat is nu eindelijk hier: de verslaglegging over een nieuwe uitbreiding voor het slim opladen van een elektrische auto. Het was mij niet gelukt dit project zo goed af te ronden zonder alle steun van de vele mensen om mij heen, en deze wil ik dus ook graag bedanken.

Allereerst wil ik Johann Hurink en Martijn Schoot Uiterkamp bedanken, mijn twee begeleiders die mij door het complete process met onverminderd enthousiasme hebben geholpen. Na ieder overleg kwam ik thuis met nieuwe duidelijkheid over de volgende stappen die ik moest nemen. Ook ben ik naast de inhoud van onze gesprekken erg dankbaar voor de open en betrokken sfeer, waardoor er ook gepraat kon worden over de problemen waar ik tegenaan liep buiten de inhoud van het onderzoek. Ik wil hiernaast Lilian Spijker heel erg bedanken voor alle steun die zij mij heeft gegeven in haar rol als studieadviseur.

Hiernaast heb ik ook veel gehad aan het contact met andere onderzoekers binnen de Energy Group aan de Universiteit. Ik wil met name Gerwin Hoogsteen bedanken voor het aanleveren van meerdere handige tools waarvan ik uitgebreid gebruik heb gemaakt met het uitvoeren van de simulaties. Ook wil ik Victor Reijnders bedanken voor het op het juiste moment aanleveren van exact de wetenschappelijke bronnen waar ik nog hard naar op zoek was.

De afleiding van het werk was ook een belangrijk onderdeel van het succesvol afronden van deze masterthesis, en voor dit onderdeel moet ik vanzelfsprekend mijn vrienden bedanken. Ik ga geen namen noemen, aangezien ik gegarandeerd iemand ga vergeten, maar ik wil wel een eervolle vermelding geven aan de volgende zeer geslaagde afleidende activiteiten: de spelletjesavonden vol competitie, de gezellige wandelingen door de natuur, de twee warme vakanties in Limburg, de fanatieke pubquizen op dinsdag, en de avontuurlijke Dungeons & Dragons sessies. Allemaal voor herhaling vatbaar!

Tenslotte noem ik ook mijn familie, voor alle steun die ze mij nog altijd bieden: broertje Laurens en vader Willem, tante Annemieke en oom Ronald, oma Brekelmans en grootvader Koster, tante Rimkje en oom Eric, en *last but not least* de nichtjes Richtsje en Zwaantje. Ik weet dat mijn moeder Cecile heel graag wilde dat ik mijn master wist te voltooiien ondanks alle tegenslagen, en ik ben blij dat ik haar zorgzaamheid nog steeds bij me draag, en dat het inderdaad gelukt is.

Contents

1 Introduction

The battle against climate change is a topic getting more and more relevant as time goes by. Many reports have been written on the heavy negative consequences of the increase in temperature, which include rising sea levels and intensified extreme weather conditions (see e.g. [Aalst, 2006], [Pörtner et al., 2022]. Agreements between governments to drastically reduce the emissions of fossil fuels in the future, as well as international conferences organised by the United Nations, aim to address and limit climate change [Falkner, 2016], [Kuyper et al., 2018]. These developments and activities all ask for cleaner, renewable energy sources, such as wind and solar power, as well as sustainable developments on the consumer side to efficiently use the resources we have.

Among the proposed solutions we see a rise in the penetration of Electric Vehicles (EV's), which use electricity instead of fossil fuels to drive [IEA, 2022]. Together with the increasing installation of solar panels by households in residential areas this rising demand for the charging of EV's causes additional strain on the energy grid on a local level, due to higher peaks in both energy consumption and energy generation, which can in the worst case lead to power outages such as described in [Nykamp et al., 2015] and [Hoogsteen et al., 2017]. To overcome this challenge, the concept of a *smart grid* has been proposed, which makes it is possible to match this growing supply and demand using control strategies, without needing to upgrade the whole physical infrastructure of the electricity grid to compensate for the increasing loads ([Bienstock and Mattia, 2007], [Strbac, 2008]). This approach is also known as *Demand Side Management*. This approach has to use the flexibility in the charging speed of EV's, allowing for control in the charging of EV's within the framework of such a smart grid [Hemavathi and Shinisha, 2022].

[van der Klauw, 2017] proposes a particular framework for Demand Side Management based on a hierarchical structure that offers several advantages in terms of scalability, privacy, and local limitations. In this framework a device creates its own charging schedule based on a centralized control paradigm. This means that an EV or a household battery requires a strategy to calculate these schedules depending on the steering signals from the centralized controller. [van der Klauw, 2017] presents several single-device models and corresponding charging strategies, however these models allow a battery to switch its operational level in a charging schedule every time interval. Such frequent switching of the power level may cause unnecessary wear on the battery, thus causing faster deterioration of its capabilities and reducing its life-span [Khalid et al., 2022], [Rachid et al., 2023].

A counter to such unnecessary wear of the battery is suggested in [van der Klauw, 2017]. It aims to limit how often the operational level of the device can be switched. In this research we look into a potential strategy to incorporate such a constraint into the models presented in [van der Klauw, 2017], by imposing a minimum run-time constraint. A more detailed formulation of this research topic is given in the next section.

1.1 Problem Statement/Research Question

The aim of this thesis is to investigate the possibility of the addition of a minimum run-time constraint to the single-device models developed in [van der Klauw, 2017], as well as possible solution strategies to these newly defined problems. This leads us to state the following research question:

What are extensions to the single-device models in [van der Klauw, 2017] that incorporate a minimum run-time constraint, and do efficient solution strategies still exist for these extended models?

To answer this question first an overview of relevant existing literature relating to this topic is discussed in Chapter 2. The existing models as well as extensions for these models with a minimum run-time constraint are presented in Chapter 3. In the remainder of this thesis the possibilities for efficient

solutions are then investigated. In Chapter 4 a dynamic programming approach is presented for one of the models. The two following chapters are dedicated to another model, where Chapter 5 analyses the structure of the problem and presents a first solution approach, and Chapter 6 presents an approach to approximate the optimal solution. For these approaches, in Chapter 7 simulation results are presented to support the conducted research. The thesis ends with some conclusions drawn in Chapter 8, as well as a discussion of some improvements and suggestions for further research.

2 Background

This chapter gives an overview of literature in the areas of research related to the work in this thesis. Specifically we discuss some topics related to demand-side energy managment, and the smart charging of an EV.

2.1 Energy Management

Energy management is a wide, multi-disciplinary field of research. Several different energy management approaches are reviewed in [Jebaraja and Iniyan, 2006]. We can distinguish between two different types of energy management, namely centralized and decentralized. In centralized energy management all demand for electricity is directly supplied by a number of external power plants. The problem of determining the energy production at each moment in time is known as the *Unit Commitment Problem* (see for example [Carrion and Arroyo, 2006]). On the other hand decentralized energy management (DEM) attempts to match demand and supply on a local level. This has several advantages compared to the central paradigm in a situation where there is on the one hand much uncontrollable energy production from renewable energy sources and on the other hand flexibility in the demand of energy. Some of these advantages include: the prevention of power outages due to a local overproduction of electricity, and the matching of the growing demand for electricity (due to the transition to electric devices) with local supply. By using such DEM strategies the costly undertaking of upgrading the physical electricity infrastructure to compensate for the increasing loads may be largely avoided.

One example of an energy management strategy is Demand Side Management. In Demand Side Management (DSM) consumers or suppliers are incentivized to shift or change the consumption or production of electricity to achieve a common system goal. Different approaches are discussed for example in [Barbato and Capone, 2014], [Siano, 2014], and [Esther and Kumar, 2016]. One method to influence consumers is to use a pricing scheme. Another method is to directly match the supply and demand of energy, taking into account any additional constraints or desires. Such a planning could both be implemented as a centralized as well as a decentralized energy management method. A decentralized approach is generally preferred because of the aforementioned reasons. In [van der Klauw, 2017] the DSM approach Profile Steering is proposed, which is a decentralized energy management approach with a hierarchical structure.

2.2 Profile Steering

The profile steering heuristic solves the problem of coordinated scheduling of smart appliances by splitting the overall problem into subproblems. This is achieved by letting each device calculate its own schedule and letting a centralized controller steer the individual scheduling of the devices. The steering signal from the central controller can for example be directly based on the desired profile of the system. A detailed description of the Profile Steering algorithm can be found in [van der Klauw, 2017], and is described there in Algorithm 3.1. The algorithm has an *initial phase* and an *iterative phase*, which we briefly summarize here.

In the *initial phase* each device must calculate a first schedule, and the central controller combines all schedules into an initial complete schedule. How to construct these initial schedules is open; a number of different approaches may be followed. One approach is to make each device fit the system objectives as well as possible. For example the objective could be to flatten the energy profile, in which case each device could initially attempt to make a schedule that flattens their own profile.

In each iteration of the *iterative phase* exactly one device updates its schedule. To achieve this the central controller requests each device to construct a new *candidate schedule* to fit the system objectives, given the current total profile as communicated by the central controller. This means that there is a

current total schedule $\mathbf{x}$ and an objective $\mathbf{f}$, and each device m has a current device schedule $\mathbf{x}^m$. Each device now constructs a candidate schedule $\hat{\mathbf{x}}^m$ (that must be feasible) that minimizes $f(\mathbf{x}-\mathbf{x}^m+\hat{\mathbf{x}}^m)$. The candidate schedules are all sent to the central controller, which picks the one that improves the objective the most, and then updates the total schedule accordingly. This process is repeated until no significant improvement is made and results in a final schedule for each device. Additional details such as possible designs for the hierarchical structure and possible steering signals are discussed in Section 3 of [van der Klauw, 2017].

The research in this thesis focuses on possible extensions of the device-level models developed for the profile steering approach in [van der Klauw, 2017]. For algorithms to be applicable in the sketched setting there are some requirements on the speed and the memory usage. This is because in Profile Steering many schedules need to be calculated, namely an initial one, and a number of updated schedules within the iterative phase. This means that the calculation of one schedule should not take much time. Moreover, the smart devices that perform these calculations usually have limited memory available, meaning that memory usage of an algorithm must be minimized. These concerns are addressed in this thesis, and are criteria for the quality of the developed algorithms.

2.3 EV Charging

Recharging the battery of an electric vehicle (EV) is a process that requires time and power, which is a potential problem with the increased usage of EV's by consumers. The requirement of power means that an increased EV penetration in a certain neighbourhood causes a heavier strain on the electricity grid. Research has been done in smart strategies for the charging of EV's, both looking at a fleet of vehicles, and considering a single device. A recent review on different DSM EV charging strategies is given in [Mohanty et al., 2022].

As mentioned in the previous section [van der Klauw, 2017] proposes several single-device models for the charging of an EV as well as for a bidirectional battery, together with solution approaches for these charging problems. The models are based on the concept of resource allocation problems. An overview of continuous nonlinear resource allocation problems and solution strategies can be found in [Patriksson, 2008] and [Patriksson and Strömberg, 2015].

A much-used goal with respect to scheduling the charging of an EV in the energy grid is so-called *valley-filling and/or peak-shaving*. Here the goal is to fill any time-spans with low grid-usage and reduce any time-spans with high grid-usage, creating more balance in the electricity grid. A widely-used valley filling algorithm is presented in [Chen et al., 2014]. As input the concept of the baseload of the energy grid is used, represented by a value p_t for each interval t. This baseload represents the basic usage of the energy grid, and any charging of the considered EV is added on top of this baseload, to represent the new total strain on the energy grid. This is modelled by the cost function $f(x) = \sum_t (x_t + p_t)^2$. Minimising this cost function results in the desired behaviour of valley-filling.

In [Schoot Uiterkamp et al., 2018] a minimum threshold for the charging rate of the continuous charging variable is added to a single-device EV model. This threshold models one of the practical limitations of EV charging rates, namely that charging at a small charging rate is not possible in practice (see for example [Young et al., 2013]). The resulting optimisation problem is proven to be NP-hard in general, however [Schoot Uiterkamp et al., 2018] develops an efficient algorithm for two specific subclasses of instances.

2.4 A Minimum run-time

The concept of a minimum run-time constraint in relation to the charging of EV has been discussed in literature before, though not often. We discuss two examples here. [Cenedese et al., 2019] considers

the optimization of a fleet of EV's, with among other constraints a constraint of a minimum run-time. Their solution approach uses the concept of a mixed integer aggregative game. This approach is not directly applicable to the problem this research considers, because instead of a fleet of vehicles we consider only the planning of a single device. In [Gerards and Hurink, 2016] a single-device model with only an *on* state and an *off* state is considered, where there is a minimum on-time and a minimum off-time constraint. A polynomial-time algorithm based on dynamic programming is given for this problem. This approach is used as the inspiration for some of the algorithms that we develop later in this thesis.

3 The Problem

The aim of this thesis is to investigate the possibility of adding a minimum run-time constraint to the two single device models as presented in [van der Klauw, 2017]. To start, in this chapter we first give the definition of these single device models. Then we consider the addition of a minimum run-time constraint. Three different possibilities are identified for which a new model with a minimum run-time constraint is presented. The structure of these new models is based on the closely related model presented in [Gerards and Hurink, 2016].

3.1 Terms and Definitions

The following terms and parameters are used within this research:

- **Problem instance $I_{\mathcal{P}}$:** a given instance of the specified optimization problem, $\mathcal{P}$, where all parameters are assigned a specific value.
- **N:** the number of equal length time intervals.
- x_t: the amount charged at a time interval t.
- l_t, u_t**:** parameters indicating for an interval t the bounds for the charging variable, $l_t \leq x_t \leq u_t$.
- **Baseloads** $p_1, ..., p_N$**:** the given power profile of the remaining elements, i.e. excluding the charging x. p_t is a parameter indicating the base power load at time interval t. Any charging x_t of the controllable device is done on top of this baseload.
- **Required charge C:** the charging goal that must be reached by time interval N.
- $f(x)$**:** the cost function. For this research we assume throughout that $f(x) = \sum_t f_t(x_t)$, with $f_t(x_t) = (x_t + p_t)^2$, the objective function for valley-filling/peak-shaving as also discussed in Section 2.3.

3.2 The Simple Models

This section presents the basic models for single-device charging taken from [van der Klauw, 2017], Chapter 4 and 5. We present both the EV and the battery models here. We first present the models with a continuous charging variable for each interval, and then the models with a certain finite set of charging rates, resulting in a discrete model.

3.2.1 Continuous Charging Variable

The continuous model concerning the charging of an Electric Vehicle (denoted as EVC) is described as a simple resource allocation problem:

$$
\begin{aligned}
EVC: \quad \min_{x} \quad & f(\mathbf{x}) = \sum_{t=1}^{N} f_t(x_t) \\
\text{s.t.} \quad & \sum_{t=1}^{N} x_t = C \\
& l_t \leq x_t \leq u_t \quad t = 1, ..., N.
\end{aligned} \tag{1}
$$

The problem BC (Battery Charging) concerns the charging of a battery and is described as follows:

$$
\begin{aligned}
BC: \quad \min_{x} \quad & f(x) = \sum_{t=1}^{N} f_t(x_t) \\
\text{s.t.} \quad & B_l \leq \sum_{t=1}^{l} x_t \leq C_l \quad l = 1, ..., N \\
& l_t \leq x_t \leq u_t \quad t = 1, ..., N,
\end{aligned} \tag{2}
$$

where B_l and C_l are the cumulative lower and upper bounds for each time interval, ensuring that the SoC (State of Charge) of the battery stays within predefined bounds for each time interval. In both these cases the lower bound l_t can actually be assumed to be 0, by a variable substitution: $\mathbf{x}' = \mathbf{x} - l_t$.

An algorithm to solve Problem *EVC* to optimality is the waterfilling algorithm. We shortly describe this algorithm in the next section, before we continue to present the discrete models.

3.2.2 The Waterfilling Algorithm

In [van der Klauw, 2017] the waterfilling algorithm is applied to solve the *EVC* problem. This algorithm is a Lagrangian multiplier approach, which means that it finds the value of the Lagrangian multiplier λ, such that the generalised KKT conditions for optimality are met. It is discussed in [Patriksson, 2008], and also described in Algorithm 4.1 of [van der Klauw, 2017]. Intuitively it can be compared to increasing a water level, since always balance is maintained between all intervals: there is one surface level of liquid, and more liquid can be poured in, causing the level to rise but never to become imbalanced. We discuss some of the mathematical details here, insofar as they are relevant for our research.

We assume that the objective function is quadratic: $f_t = \frac{1}{2}a_t x_t^2 + b_t x_t + c_t$ and that $a_t > 0$. In this special case the KKT conditions for optimality for a given λ reduce to:

$$
x_t(\lambda) = \begin{cases} 0 & \text{if } \lambda \leq b_t \\ \frac{\lambda - b_t}{a_t} & \text{if } b_t \leq \lambda \leq a_t x_t^{max} + b_t \\ x_t^{max} & \text{if } \lambda \geq a_t x_t^{max} + b_t, \end{cases} \tag{3}
$$

together with the condition that $\sum_t x_t = C$. These conditions define an optimal solution for the *EVC* problem. The waterfilling algorithm finds the correct value for λ by first making an initial guess $\hat{\lambda}$, and calculating the total charge $\hat{C}$ for this guess (using Equation (3)). Once $\hat{C} = C$ all optimality conditions are met and an optimal solution has been found. To achieve this equality the value of $\hat{\lambda}$ is incrementally increased, leading to an increase of $\hat{C}$. The question remains by how much $\hat{\lambda}$ should be increased in each increment. This is determined by the so-called breakpoints of the problem instance.

If the guess $\hat{\lambda}$ increases then the guess $\hat{x}$ for the corresponding solution increases according to Equation (3). This means that for an interval t with $\hat{\lambda} \leq b_t$ $\hat{x}_t = 0$, and we call this interval *inactive*. When $\hat{\lambda} > b_t$ the charge $\hat{x}_t$ increases with $\hat{\lambda}$, and we call such an interval *active*. The maximum charge is reached when $\hat{\lambda} \geq a_t x_t^{max} + b_t$, and we call any maximally charged interval *full*. The breakpoints are the points where any interval t changes either from *inactive* to *active*, or from *active* to *full*. The first situation occurs when $\hat{\lambda}$ reaches one of the values in the set $A := \{b_1, ..., b_N\}$, and the second situation when $\hat{\lambda}$ reaches a value in the set $B := \{a_1 x_1^{max} + b_1, ..., a_N x_N^{max} + b_N\}$. These values are called the breakpoints, and they are the points where the algorithm calculates updates $\hat{C}$ as well as the set of active intervals, until $\hat{C} = C$.

$\hat{C}$ is updated by using the *increasefactor* of each interval, which is equal to $\frac{1}{a_t}$. This factor represents by how much $\hat{x}_t$ increases relative to the increase of $\hat{\lambda}$. $\hat{C}$ must thus be increased by an amount equal to the sum of the increasefactors of all active intervals.

3.2.3 Discrete Charging Levels

The second model is a charging model with a finite set of discrete charging levels, whereby for each interval the amount charged must be equal to one of these levels. The set of charging levels for interval t is defined as: $\mathbf{Z}_t = \{z_t^0, ..., z_t^{M_t}\}$, where $z_t^j < z_t^{j+1}$ for all t and j.

The problem discrete EVC (*dEVC*) is now defined by:

$$\begin{aligned} \textit{dEVC:} \quad \min_x \quad & f(x) = \sum_{t=1}^{N} f_t(x_t) \\ \text{s.t.} \quad & \sum_{t=1}^{N} x_t = C \\ & x_t \in \mathbf{Z}_t \quad t = 1, ..., N. \end{aligned} \tag{4}$$

Following the same line as for the continuous case, discrete BC (*dBC*) is defined by:

$$\begin{aligned} \textit{dBC:} \quad \min_x \quad & f(x) = \sum_{t=1}^{N} f_t(x_t) \\ \text{s.t.} \quad & B_l \leq \sum_{t=1}^{l} x_t \leq C_l \quad l = 1, ..., N \\ & x_t \in \mathbf{Z}_t \quad t = 1, ..., N. \end{aligned} \tag{5}$$

Again, we assume without loss of generality that $z_t^0 = 0$ for all t, by applying the transformation $x'_t = x_t - z_t^0$.

These models form the basis of the potential new models, where the constraint of a minimum run-time is added.

3.3 The Minimum Run-time

The remainder of this chapter focusses on modifying the models presented in the previous sections to include the constraint of a minimum run-time. There are multiple interpretations possible for this minimum run-time. The ones that we consider in this thesis are as follows:

- A minimum time to be in a specific charging 'region'. These constraints express a minimum required time for a device to remain charging, discharging, or remain off. Note that within each of these regions the charging value is still free to change.
- A minimum time to be charging at one specific charging rate: a device is constrained to charge at one charging rate for a minimum number of time intervals.

In the next section some model modifications are presented based on these aspects. Finally, we select the options that seem to be the most promising, with respect to the possibility for efficient solution methods.

3.4 The New Models

3.4.1 Discrete and Fixed Charging Value

We use the discrete models described in Section 3.2.3 as basis for these models. It is assumed that the set $\mathbf{Z}_t$ is the same for each time interval, resulting in the following simpler restriction: $x_t \in \mathbf{Z} = \{z_0, ..., z_M\}$ for all t. This adaptation is chosen to simplify the addition of a minimum run-time on a fixed charging value over multiple intervals.

We introduce for each operational level z_j of the device a minimum run-time R_j. This means that if the device charges at rate z_j it must do so for at least R_j consecutive intervals. This constraint is added to the basic model to create a discrete model with a minimum run-time constraint. To define this formally we define for each discrete charging level z_j a subspace of the total solution space as being the set of solutions that adhere to this minimum run-time constraint. In detail, for each $j \in \{0, ..., M\}$ we define a set:

$$\mathcal{X}_j := \{x \in \mathbf{Z}^N \mid \text{if } x_i = z_j \text{ there exists a } k \text{ with } k \leq i \leq k + R_j - 1 \text{ and } x_k = ... = x_{k+R_j-1} = z_j\}.$$

Or in other words: for any interval that charges at a charging rate z_j, there must be at least $R_j - 1$ neighbouring intervals that charge at this same rate for a solution to be feasible. Using this defintion, a solution fulfills all minimum run-time constraints if $x \in \bigcap_{j=0}^{M} \mathcal{X}_j$. We define Problem *dm1EVC* as:

$$\begin{aligned} \textit{dm1EVC:} \quad \min_{x \in \bigcap_{j=0}^{M} \mathcal{X}_j} \quad & f(x) = \sum_{t=1}^{N} (x_t + p_t)^2 \\ \text{s.t.} \quad & \sum_{t=1}^{N} x_t = C. \end{aligned} \tag{6}$$

The same adaption can be applied to problem BC, leading to problem *dm1BC*:

$$\begin{aligned} \textit{dm1BC:} \quad \min_{x \in \bigcap_{j=0}^{M} \mathcal{X}_j} \quad & f(x) = \sum_{t=1}^{N} (x_t + p_t)^2 \\ \text{s.t.} \quad & B_l \leq \sum_{t=1}^{l} x_t \leq C_l \quad l = 1, ..., N. \end{aligned} \tag{7}$$

The problems described and solved in [Gerards and Hurink, 2016] are a special case of these two problems, where $\mathbf{Z}_t = \{0, 1\}$ $\forall t$, and are described in more detail in Section 4.

Concerning the time complexity of these two problems, van der Klauw proved that the two problems *dEVC* and *dBC* without the minimum run-time constraint are NP-hard (see [van der Klauw, 2017]). As the problems with the minimum run-time addition are a generalization of problems *dEVC* and *dBC*, we can conclude that the problems *dm1EVC* and *dm1BC* are also NP-hard.

3.4.2 Continuous and Fixed Charging Value

In this and the next sections, following the lines of Section 3.4.1, two variations of the minimum run-time constraint are considered for the continuous model, based on the continuous problem described in (1). The basis for the variation presented in this section is a minimum run-time on a fixed value (see also the second bullet-point in Section 3.3). We make a similar assumption as for the sets $\mathbf{Z}_t$ in Problem *dm1EVC*, namely we assume that the maximum charging rate is the same for all intervals t,

i.e. $u_t = u$ where u is some positive value.

We again define the minimum run-time as a minimum number of consecutive intervals where the device should charge exactly the same amount. This means for example that if $x_1 = s$, then it also must be that $x_2 = x_3 = ... = x_{R_1} = s$, since interval 1 is the first interval of the planning horizon. Only at interval $R_1 + 1$ the device may switch to another charging value, whereafter it again is required to remain at this value for at least R_1 intervals. However, the device may also make the decision to keep charging at value s after the first R_1 intervals.

To formally add this constraint to the model we define three subspaces of the solution space:

$$\mathcal{X}_{-1} := \{x \in \mathbb{R}^N \mid \text{ for } s < 0, \text{ if } x_t = s \text{ there exists } k \text{ with } k \leq t \leq k + R_{-1} - 1 \text{ and } x_k = ... = x_{k+R_{-1}-1} = s\}$$
$$\mathcal{X}_0 := \{x \in \mathbb{R}^N \mid \text{ if } x_t = 0 \text{ there exists } k \text{ with } k \leq t \leq k + R_0 - 1 \text{ and } x_k = ... = x_{k+R_0-1} = 0\}$$
$$\mathcal{X}_1 := \{x \in \mathbb{R}^N \mid \text{ for } s > 0, \text{ if } x_t = s \text{ there exists } k \text{ with } k \leq t \leq k + R_1 - 1 \text{ and } x_k = ... = x_{k+R_1-1} = s\},$$

where R_j is the minimum run-time of the device on mode j (discharging, off, charging). A solution fulfills all minimum run-time constraints if $x \in \mathcal{X}_{-1} \cap \mathcal{X}_0 \cap \mathcal{X}_1$. The *EVC* model with a minimum run-time on one charging value (abreviated to *m1EVC*), expanded from problem definition (1), is now defined by:

$$\begin{aligned} \textit{m1EVC:} \quad \min_{x \in \mathcal{X}_0 \cap \mathcal{X}_1} \quad & f(x) = \sum_{t=1}^{N} (x_t + p_t)^2 \\ \text{s.t.} \quad & \sum_{t=1}^{N} x_t = C \\ & 0 \leq x_t \leq u \quad t = 1, ..., N. \end{aligned} \tag{8}$$

Similarly we define the adapted problem *m1BC* based on problem (2):

$$\begin{aligned} \textit{m1BC:} \quad \min_{x \in \mathcal{X}_{-1} \cap \mathcal{X}_0 \cap \mathcal{X}_1} \quad & f(x) = \sum_{t=1}^{N} (x_t + p_t)^2 \\ \text{s.t.} \quad & B_l \leq \sum_{t=1}^{l} x_t \leq C_l \quad l = 1, ..., N \\ & 0 \leq x_t \leq u \quad t = 1, ..., N. \end{aligned} \tag{9}$$

The above formulations concern one potential model for the definition of the minimum run-time constraint with a continuous charging variable. However, we also may define a second potential continuous model, with the minimum run-time constraint defined in a different way. This second variation is presented in the next subsection.

3.4.3 Continuous and Free Charging Value

In this section we consider a variation of the minimum run-time model which is also an extension of the *EVC* model. The basis for this variation is given in the first bullet-point of Section 3.3: there is a minimum required time to remain charging in a certain region.

We first present the most straightforward or intuitive way to define this variation. We consider the situation where the charging variables x_t are bounded between 0 and a maximum u_t, and we define a

minimum run-time R_1 in the following manner: if $x_t > 0$, then there must be R_1 intervals neighbouring interval t for which the charging rate is also strictly positive, i.e. larger than some minimum charging level $\epsilon > 0$, where ϵ may be arbitrarily small. Other than this addition of the minimum on-time R_1 the problem remains the same as (1). However, there are some problems with this model, which become apparent if we compare it to Problem *EVC* to which is forms an extension. For illustration we assume that x^* is an optimal solution for a certain instance of Problem *EVC* where there are three consecutive intervals $t-1, t, t+1$ such that $x^*_{t-1} = 0$, $x^*_t > 0$, $x^*_{t+1} = 0$. Note that x^* is not a feasible solution for our newly defined problem with the minimum run-time constraint (assuming $R_1 > 1$), since there is a single isolated interval with a positive charge. However, based on solution x^* an optimal solution x' can easily be constructed. Let $x'_i = x^*_i$ for all i except $t-1$, t, and $t+1$ and let $x'_{t-1} = x'_{t+1} = \epsilon > 0$, and $x'_t = x^*_t - 2\epsilon$. If $\epsilon \to 0$ this solution will converge towards x^*. Using this procedure, any infeasibility of solution x^* regarding the minimum run-time constraint can be solved, while the cost remains the same up to an ϵ, and the result is also an optimal solution for the new minimum run-time problem.

The reasoning above shows that the specified addition of a minimum run-time constraint will not have a meaningful purpose. Besides this, from a practical perspective, it is not possible to charge at very small powers (see for example [Young et al., 2013]), even if that would theoretically be the best solution. It makes more sense to define some lower bound X_{min} on the charging value for EVC. In the following such a model is presented.

For this new formulation we register whether an interval is charging or not. We define a binary variable for each time interval t, denoted by y_t, to indicate whether the device is *on* or *off*. These binary variables can be used to define a minimum run-time constraint. To this end we define:

$$\mathcal{Y}_1 := \{y \in \{0,1\}^N \mid \text{if } y_t = 1 \text{ there exists } k \text{ with } k \leq t \leq k + R_1 - 1 \text{ and } y_k = ... = y_{k+R_1-1} = 1\}$$
$$\mathcal{Y}_0 := \{y \in \{0,1\}^T \mid \text{ if } y_t = 0 \text{ there exists } k \text{ with } k \leq t \leq k + R_0 - 1 \text{ and } y_k = ... = y_{k+R_0-1} = 0\},$$

where R_1 is a minimum on-time, R_0 is a minimum off-time. This means that any solution in the subset $\mathcal{Y}_1$ is always *on* for at least R_1 consecutive intervals at a time, and the same holds for $\mathcal{Y}_0$ and being *off*. Now we define Problem *mEVC* as follows:

$$\begin{aligned} mEVC: \quad \min_x \quad & f(x) = \sum_{t=1}^{N} (x_t + p_t)^2 \\ \text{s.t.} \quad & \sum_{t=1}^{N} x_t = C \\ & y_t X_{min} \leq x_t \leq y_t u_t \quad t = 1, ..., T \\ & y \in \mathcal{Y}_0 \cap \mathcal{Y}_1. \end{aligned} \tag{10}$$

Without a minimum run-time constraint, the *EVC* problem with such a threshold is solved in [Schoot Uiterkamp et al., 2018] for a specific subclass of instances. It is also proven in [Schoot Uiterkamp et al., 2018] that the problem is NP-hard, meaning that Problem *mEVC* is also NP-hard. Furthermore, it is worth mentioning that approaches based on sorting the intervals according to their base loads cannot be used anymore, as was done in [Schoot Uiterkamp et al., 2018] to solve a certain subclass of problem instances.

3.5 Choice of Problems

The first of the presented models, *dm1EVC*, is a direct expansion of the on/off model presented in [Gerards and Hurink, 2016], where a polynomial solution method is given. Although the method does not hold up for the more general case presented here, the structure of the approach can be used to

design a similar, pseudo-polynomial algorithm. This idea is expanded upon in Chapter 4.

The second and third presented models are two continuous models, for which there is no existing structure in place to inspire an efficient approach. Of the two new models Problem *m1EVC* is expected to be simpler, based on the fact that Problem *mEVC* has the added difficulty of the charging bounds, which have already been proven in the past to complicate the problem in [Schoot Uiterkamp et al., 2018]. Therefore the focus of this research is on Problem *m1EVC*. Chapter 5 is devoted to determining the underlying structure of this problem, and to designing an exact method using this structure. In Chapter 6 a faster approximation method is developed.

4 Discrete EVC with Minumum run-time

In this chapter we develop a solution method for problem *dm1EVC*. The solution method is based on the research in [Gerards and Hurink, 2016], and it largely follows the same reasoning. We start with explaining the solution method for the on/off problem in more detail, and based on this we give an extentsion of this solution method to also solve problem *dm1EVC*.

4.1 On/Off Device

We use the models for an on/off device with a minimum run-time constraint as they were first defined in [Gerards and Hurink, 2016]. These follow the same structure as the minimum run-time models defined in Chapter 3.

The basis are subspaces which are defined as follows:

$$\mathcal{X}_1 := \{x \in \{0,1\}^N \mid \forall i \in \{1,...,N\} : x_i = 1 \Rightarrow \exists k : k \leq i \leq k + R_1 - 1 \wedge x_k = ... = x_{k+R_1-1} = 1\}$$
$$\mathcal{X}_0 := \{x \in \{0,1\}^N \mid \forall i \in \{1,...,N\} : x_i = 0 \Rightarrow \exists k : k \leq i \leq k + R_0 - 1 \wedge x_k = ... = x_{k+R_0-1} = 0\},$$

leading to the following model for On/Off EV charging:

$$\textit{On/Off m1EVC:} \quad \min_{x \in \mathcal{X}_0 \cap \mathcal{X}_1} \quad f(x) = \sum_{t=1}^{N} f_t(x_t) \tag{11}$$
$$\text{s.t.} \quad \sum_{t=1}^{N} x_t = C,$$

and following the same structure for Battery charging:

$$\textit{On/Off m1BC:} \quad \min_{x \in \mathcal{X}_0 \cap \mathcal{X}_1} \quad f(x) = \sum_{t=1}^{N} f_t(x_t) \tag{12}$$
$$\text{s.t.} \quad B_l \leq \sum_{t=1}^{l} x_t \leq C_l \quad l = 1, ..., N$$

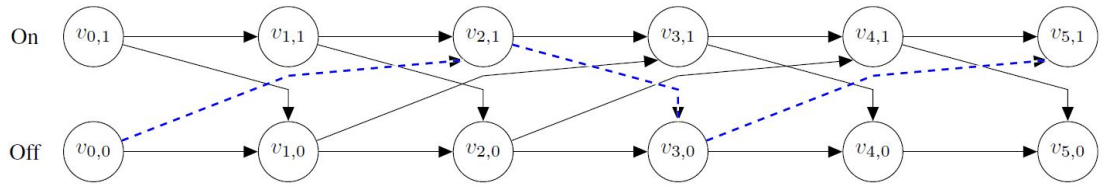
In [Gerards and Hurink, 2016] a dynamic programming approach to solve *On/Off mEVC* and *mBC* is presented. This dynamic program determines for each state the cheapest way to get to this state, resulting in the cheapest feasible solution. This approach is represented as a directed graph, which is explained in the following section. We expand this graph to also represent Problem *dm1EVC*.

4.2 The Graph Representation

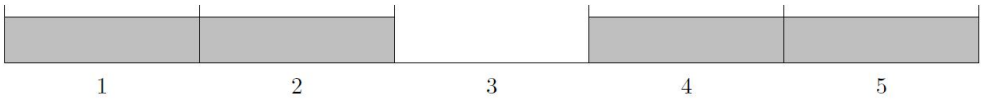
4.2.1 The On/Off Graph

In [Gerards and Hurink, 2016] a problem instance of *On/Off m1EVC* is represented as a directed graph. For each time interval n there exist two nodes, *on* and *off*: $v_{n,0}$ and $v_{n,1}$, meaning that the set of all vertices is given by $V = \{v_{1,0}, ..., v_{N,0}, v_{1,1}, ..., v_{N,1}\}$. The edges in this directed graph only go forward in time. This ensures that any path starting at interval 0 and ending at interval N defines a possible charging plan. Furthermore, the graph is constructed such that the minimum run-time/off-time constraint is met for any such path in .

An edge from a node v_{n,z_j} to $v_{n',z_{j'}}$ $(n' > n)$, where $z_{j'} \neq z_j$, represents a switch at interval $n+1$ from state $z_{j'}$ to state z_j, i.e. either from *off* to *on*, or from *on* to *off*. This arc corresponds to the setting



(a) Graph depicting the minimum continuous run-time/off-time constraints.



(b) On/off schedule of device corresponding with the dashed blue path from Fig. 1a.

Figure 1: *Picture taken from [Gerards and Hurink, 2016].* Example of a graph with the minimum continuous run-time/off-time constraint, with the dashed path corresponding to a single schedule. There are 5 intervals, and $R_0 = 1, R_1 = 2$. The graph nodes from Fig. 1a correspond to the schedule from Fig. 1b.

$x_{n+1} = ... = x_{n'} = z_{j'}$. These intervals thus all have the same state, *on* or *off* in a solution using this edge. On the other hand, an edge connecting a node to the same node of the next time interval, i.e. with the same charging rate, so v_{n,z_j} to v_{n+1,z_j}, means that the charging rate remains z_j for interval $n+1$.

It remains to explain how the minimum run-time constraint gets incorporated into this graph representation. In the *on/off* model a switch from level 0 to 1 or 1 to 0 can only be made *after* a minimum time of R_1 or R_0 respectively. We represent a switch from 0 to 1 after a time interval n by an edge connecting node $v_{n,0}$ to $v_{n+R_1,1}$, thus exactly R_1 steps in the future. This implies that if a path uses this edge in the corresponding solution the charging will always be *on* for at least R_1 consecutive intervals, and likewise *off* for at least R_0 consecutive intervals after a switch. An example of such a graph and feasible path can be found in Figure 1. Note that a situation where the charging is *on* for $k > R_1$ intervals is represented by one arc from the *off* level to the *on* level and $R_1 - k$ subsequent arcs between neighbouring vertices on the *on* level.

Finally the cost of an edge is defined with the cost function $f = \sum_{t=1}^{N} f_t(x_t)$ in mind. If there is an edge e from vertex v_{t,z_*} to v_{t',z_j} used in a path this represents the intervals $t+1$ to t' in the actual charging solution, where each interval should charge at level z_j (either *on* or *off*). The cost of this edge can be defined accordingly as: $F(e) = F(v_{t,z_*}v_{t',z_j}) = \sum_{i=t+1}^{t'} f_i(z_j)$. Now to minimize the cost all one has to do is select a cheapest feasible path, and the corresponding charging solution can be determined by backtracking through this solution. In the next section we generalize this graph representation to Problem *dm1EVC*.

4.2.2 Expanding the Graph

For the expanded problem *m1EVC* there are a total of M charging levels, each level z_j with a minimum on-time of R_j intervals. We define the nodes of the graph in the same manner as in the previous subsection, resulting in a total of NM nodes $V = \{v_{1,z_0}, ..., v_{N,z_0}, ..., v_{1,z_M}, ..., v_{N,z_M}\}$. Any edge from v_{n,z_j} that represents a switch to a different operational level $z_{j'}$ connects to a node $R_{j'}$ steps into the future, to $v_{n+R_{j'},z_{j'}}$. This is a straightforward expansion of the *on/off* graph.

Figure 1 can be adjusted to reflect this expanded graph in the following way: each row of nodes now represents one charging level z_j and there are M rows in total. From each node in row j there is a connecting edge to a node of every other row j' exactly $R_{j'}$ steps into the future, as well as an edge

connecting to the next node on row j, in the same manner as the connecting arrows in Figure 1 are drawn. Any path from an initial node to an end node now corresponds to a charging solution for problem *m1EVC* respecting the minimum run-time constraint. The cost of each edge can be defined based on the cost function of *m1EVC* in the same way as before for *On/Off m1EVC*.

With the graph defined this way any feasible path the first interval ($n = 0$) to the final interval ($n = N$) is a candidate solution to the problem as defined in (6) since each of these solutions adheres to the minimum run-time constraints. But in this graph no restrictions are laid upon the amount charged (C) at $n = N$, meaning that not every such path gives a feasible solution to *dm1EVC* yet. However, the graph can be used as the basis of a dynamic programming approach to find an optimal solution to Problem *dm1EVC* ((6)), in the same way as was done in [Gerards and Hurink, 2016] for the special case of $M = 2$. This algorithm is described below.

4.3 The Dynamic Programming Algorithm

We use the expanded graph as the basis for a dynamic programming approach in order to solve Problem *dm1EVC* to optimality. To this end we first define a look-up matrix T along the same line as [Gerards and Hurink, 2016]. In this matrix there is an entry $T(v, m)$ for each vertex and each amount m charged at this vertex in the cheapest path. Each entry $T(v, m)$ equates to (v', c) where v' is the predecessor in the cheapest path to v, charging a total of exactly m, and c is the cost of this path. The size of T is $MN \times Nz_M$. When defined in this manner we can conclude that any entry $T(v_{N,z_i}, C)$ encodes the entire path of a feasible solution, since exactly C has been charged at the final interval. The optimal solutions to Problem *dm1EVC* are the ones with the lowest cost c.

To determine the elements of the matrix T we move forwards through time in the graph, starting at time interval 0 (the dummy interval). For every possible charged amount m and each possible operational level j the next step in the current cheapest path is explored by jumping to a new operational level j' for the subsequent $R_{j'}$ intervals, or remaining on this operational level for 1 more interval. For each of these possibilities the cost is compared to the current cost at the relevant node, and the cheapest of the two is saved as the new entry of T. This algorithm is presented in 1. The time complexity of the algorithm is $O(N^2M^2z^M)$.

4.4 Improvements on the Run-time

A possibility to speed up the algorithm is to bound off certain states, states that will certainly not lead to a feasible solution in the end. First of all transitions from a state do not have to be considered (in line 9) if $c = \infty$, since this will not lead to a feasible path. Secondly when transitioning to a new state it can be checked if the end goal is still reachable, so the goal of having charged C at the end. Two checks can be made for this when transitioning to a different level, $z_{j'}$ (lines 10 to 16):

$$m + R_{j'}z_{j'} + (N - n - R_{j'} - 1)z_M \geq C$$
$$m + R_{j'}z_{j'} + (N - n - R_{j'} - 1)z_0 = m + R_{j'}z_{j'} \leq C$$

If one of these two checks fails, the loop can break and continue with the next j'. Also, in the special case where all minimum run-times are the same ($R_j = R_i$), then once the second check fails it will fail for all higher j' for this state (v_{n,z_j}, m), so the inner loop can be stopped entirely.

In a similar vein two checks can be done for continuing on the same level (lines 17 to 21):

$$m + z_j + (N - n - 1)z_M \geq C$$
$$m + z_j + (N - n - 1)z_0 = m + z_j \leq C$$

Algorithm 1 Create T

Initialization:
$T(v_{0,i}, 0) = (0, 0)$ for all i
else: $T(v, m) = (v, \infty)$

Loop:
for $n = 0, ..., N-1$ **do**
  for $m = 0, ..., n \cdot z^M$ **do**
    for $j = 0, ..., M$ **do**
      $(v, c) = T(v_{n,z_j}, m)$ (*consider transitions from state* (v_{n,z_j}, m))
      for $j' \in \{0, ..., M\} \backslash \{j\}$ **do**
        $(v', c') = T(v_{n+R_{j'}, z_{j'}}, m + R_{j'} z_{j'})$

        if $n + R_{j'} \leq N$ **and** $c + F(v_{n,z_j} v_{n+R_{j'}, z_{j'}}) < c'$ **then**
          $T(v_{n+R_{j'}, z_{j'}}, m + R_{j'} z_{j'}) = (v_{n,z_j}, c + F(v_{n,z_j} v_{n+R_{j'}, z_{j'}}))$
        end if

      end for
      $(v', c') = T(v_{n+1,z_j}, m + z_j)$

      if $c + F(v_{n,z_j} v_{n+1,z_j}) < c'$ **then**
        $T(v_{n+1,z_j}, m + z_j) = (v_{n,z_j}, c + F(v_{n,z_j} v_{n+1,z_j}))$
      end if

    end for
  end for
end for

4.5 Battery Charging

In [Gerards and Hurink, 2016] *On/Off m1BC* is solved with a simple extension of the algorithm for EV charging. This same extension works for our purposes as well, in the case of problem *dm1BC*, and we describe it here. There is a different set of constraints on the variable x, namely a cumulative lower and upper bound for each interval. These cumulative constraints restrict the loop over $m = 0, ..., n \cdot z^M$ further in the algorithm, by restricting the value to be between B_n and C_n. This means that this variant can easily be solved by the same algorithm approach.

4.6 Conclusion

In this chapter the problems described in Equations (6) and (7) were investigated as being an extension of the problem described in [Gerards and Hurink, 2016]. A pseudopolynomial approach to calculate an optimal charging plan has been presented. Some simulation results are presented in Chapter 7. The next chapters are devoted to the same charging problem with minimum run-time, but now with continuous charging variables instead of discrete variables.

5 EVC with Minumum run-time

For the charging of an Electric Vehicle a simple model was presented in (1), and it is based on continuous charging variables. Two potential new models are presented in (8) and (10), each with a different way of adding a minimum run-time constraint to this simple convex model. For the two new models this means that the EV can no longer switch its charging value freely at each interval, since there is now a dependency between intervals, making the problem more difficult. As already stated only the model described in (8) is the focus of subsequent research, based on perceived viability. The model for *m1EVC* as considered in this chapter has one additional simplification compared to (8): we assume that $R_1 = R_0 = R$ for some positive integer R, in order to define a single minimum run-time for all charging values. In this section we analyse this specific model further to identify structures that could be used to approach this problem. In the following a first approach is presented, and an implementation of this approach is discussed at the end of this chapter.

5.1 Definitions

In order to more effectively reason about Problem *m1EVC* we introduce some new terminology, with the aim to describe a solution to Problem *m1EVC* with respect to the minimum run-time constraint. A charging solution is generally represented by a tuple $x \in \mathbb{R}^N$. For Problem *m1EVC* we know that each feasible solution x must adhere to the minimum run-time constraint, and therefore we can describe the structure of a feasible solution in a more specific way. Namely, any charging rate must always be charged for a minimum of R consecutive intervals, in order for a solution to be feasible. In other words, for a solution x we can always state that it has substructures $x_{t_1} = x_{t_1+1} = ... = x_{t_2}$ for some intervals t_1 and t_2, for which we have that $t_2 - t_1 + 1 \geq R$.

We can use the above observation to define an alternative representation of a solution to Problem *m1EVC* that is useful for this research. In the above example the only important information is the two intervals t_1 and t_2, as well as the charging rate. If this information is given the solution for all intervals t_1 to t_2 can be constructed. Therefore, to define an entire solution, we only need these 'endpoints', the beginning and end of a set of intervals that charge at the same rate, and we need the corresponding charging rates. We call such a maximum set of consecutive intervals charging at the same rate a *block* of intervals. So t_1 and t_2 would be the endpoints of one block, and the length of this block would be $t_2 - t_1 + 1$. Note that in a solution every interval is part of exactly one block.

Finally, we also define the concept of a *configuration*. A configuration lists the lengths of all blocks in the order they occur, and in this way codifies the endpoints of each block too. Therefore, a configuration together with the charging rate for each block codifies a solution to Problem *EVC*. Note that in this way the constraints are split: if all lengths in the configuration are greater than R, then the solution adheres to the minimum run-time constraint and we call the configuration feasible; if all the charging rates are between 0 and u then the solution adheres to the flow constraints. Representing a solution using such configurations makes it easier to reason about solution approaches for the new minimum run-time constraint. Below, we give the formal definitions of a block and a configuration as used in this research, as well as a small example.

Definition 5.1 (Block of intervals)**.** *There are N intervals of equal length. For $i, j \in \{1, ..., N\}$ with $j > i$ the subset of intervals $B = \{i, i+1, ..., j-1, j\}$ defines a block of intervals, where the length of the block is $j - i + 1$ (i.e. the number of intervals).*

Definition 5.2 (Feasible configuration)**.** *Given the set of intervals 1 to N, and given $R \in \mathbb{Z}$ with $R > 0$, let $n_1, n_2, ..., n_k \in \{R, ..., N\}$ represent a set of block lengths, such that $\sum_{i=1}^{k} n_i = N$. Then the corresponding configuration is defined as the tuple $(n_1, n_2, ..., n_k)$. Hereby, an element n_i of configuration $(n_1, n_2, ..., n_k)$ now represents the length of the ith block of this configuration.*

Example 5.1. *If $N = 7$ and the minimum run-time and minimum idle time are both equal to 2 ($R = 2$), then one possible configuration would be $(3, 2, 2)$, indicating that the first block has a length of three and the second and third blocks both have a length of two. In other words, this implies that intervals $1, 2, 3$ form the first block, $4, 5$ the second, and $6, 7$ the third.*

We note that the size of a configuration k, representing the number of blocks, is a variable for which we may set some restrictions. However, we can also consider the solution space of all possible configurations. To this end we define both the set of all feasible configurations of a certain length, as well as how these sets together form the total solution space.

Definition 5.3. *Let S^k be the set that, given the set of intervals 1 to N and given $R \in \mathbb{Z}$, $R > 0$, contains all feasible configurations of length k. Note that $k \in \{1, ..., \lfloor \frac{N}{R} \rfloor\}$. Also define S as being the set of all configurations: $S := S^1 \cup S^2 \cup ... \cup S^k$.*

In the following we consider the problem of finding the best charging plan for a given configuration $n = (n_1, n_2, ..., n_k)$ for a problem instance of *m1EVC*, **within the constraints of this configuration**. For any solution x of this problem given configuration n we have that:

$x_i = x_j$ *for all intervals* $i, j \in B_l$ *and for all blocks* B_l, $1 \leq l \leq k$.

We denote the optimal solution within these additional constraints of configuration n x^n. The cost associated with this optimal planning x^n is denoted by $f(x^n)$.

5.2 Analysing the Problem

In the first part of this section we create some more intuition for the problem and its structure. Some of this intuition gives rise to a first structural property, which we prove, and then use to develop an optimization strategy in the next section.

In definition Problem *m1EVC* is similar to Problem *dm1EVC*, which was investigated in Chapter 4. To develop a solution approach for Problem *dm1EVC* the problem was described using a directed graph (see an example in Figure 1), and using this graph representation a dynamic programming approach was developed. The question now is whether such a graph representation is also possible for Problem *m1EVC*. Unfortunately this is not the case. Both problems have the same type of minimum run-time constraint, where it is required to charge for at least the minimum time on one value. However, where Problem *dm1EVC* had a finite set of discrete charging values, in Problem *m1EVC* the charging value can be any real number within $[0, u_t]$. So, in theory, the graph representation would per interval have a node for each possible charging value, which would result in an uncountable infinite set of nodes for each interval. This is not a possible approach, and we need to take a different direction for a solution.

5.2.1 Comparison to *EVC* and *mEVC*

As Problem *mEVC* is an extension of Problem *EVC* (Equation (1)), we can look into where it differs compared to this simpler problem. This means that we should focus on the new difficulties in the solution structure that are not present in Problem *EVC*.

In an *EVC* problem instance each interval can be described by a lower and upper breakpoint, where the lower breakpoint indicates when it is profitable to activate the interval, and thus start charging in this interval, and the upper breakpoint indicates when the interval has reached the end of its charging. A solution can be characterized by one λ value (the dual) and these breakpoints. One algorithm using this principle is the waterfilling algorithm (also discussed in Section 3.2.2), where the λ value is incrementally increased and intervals are activated or deactivated according to their breakpoints. However, in Problem *m1EVC* a solution can no longer be characterized by only a λ value and the

breakpoints of the intervals, because the intervals cannot be activated and deactivated separately, as now the charging of one interval directly influences whether other intervals charge, due to the minimum run-time constraint.

Secondly we observe that, given a feasible configuration n and a corresponding solution x^n we can raise or lower the charge of a block by increasing or decreasing the charge of all intervals within this block by the same value. In this way a similar process to that of the waterfilling algorithm should be possible, within one configuration, and as we discuss in more detail in Section 5.2.2 we can even define a λ along the same lines. Now, given this concept, the ideal situation for a solution strategy would be if the optimal configuration for a certain problem instance of *m1EVC* would be independent of the needed charge C. The reason for this is that in this case we could start at a charge of 0, calculate the best block to raise in charge, and iteratively continue this process in the same way as the waterfilling algorithm for Problem *EVC*. However, unfortunately it is not the case that the optimal configuration is independent from C, or in other words: the optimal configuration changes depending on how much must be charged in total. The following example proves this point.

Example 5.2. *Consider a m1EVC problem instance with $N = 6$, $R = 2$ as the minimum run-time, a baseload energy profile $p = (2, 1, 2, 3, 3, 2)$. First we set the required charge C to 1. In this case there is one unique optimal solution, $x_1^* = (\frac{1}{2}, \frac{1}{2}, 0, 0, 0, 0)$, corresponding to the configuration $n_1 = (2, 4)$. Second, for $C = 5$ there is another unique optimal solution: $x_2^* = (\frac{4}{3}, \frac{4}{3}, \frac{4}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, with another configuration: $n_2 = (3, 3)$.*
This shift happens the moment that in configuration n_1 the second block should be activated to keep the optimal charging balance within this configuration. As soon as this happens, configuration n_2 becomes more cost-efficient in its grouping of the intervals. We can calculate the tipping point where the second block should activate, which is at $C = 2$, as at this tipping point the costs of the optimal solutions for configurations n_1 and n_2 are exactly equal, and for $C > 2$ the cost becomes lower for configuration n_2.

This observation implies that we cannot infer the optimal configuration of a certain problem instances from the baseloads p_t alone; it also depends on C. However, we may separate the selection of a configuration from the calculation of the corresponding optimal charge. A configuration n is independent from C or p, as it only depends on R. In the continuous problem *mEVC* presented in Chapter 3 (see 10) this separation is made explicit through the binary variables y_t, where y is an assignment of the intervals into a certain configuration, meeting the constraint of the minimum run-time, and y only depends on R. Problem $m1EVC$ has this same structure, but more implicit: a solution x must be selected from the set $\mathcal{X}_0 \cap \mathcal{X}_1$. This independence means that it is possible to first select a feasible configuration n. Then this configuration for Problem *m1EVC* puts restrictions on the charging solution x: all intervals within a block of the configuration must charge at the same value.

We aim to use this property to calculate the optimal solution x^n of a certain configuration n. If we know that intervals t_1 to t_2 ($t_1 < t_2$) are contained in the same block of configuration n, then this implies that $x^n_{t_1} = x^n_{t_1+1} = ... = x^n_{t_2}$. Therefore changing the value of any one of these intervals changes the value of all these intervals by the same amount. As such we may treat all these intervals together as one 'interval'. Such a transformation makes the problem of finding x^n quite similar to Problem *EVC*, and we can use the waterfilling algorithm to find the optimal solution for Problem *EVC*. This similarity is given shape and proven in the next section. Finally, if we know how to calculate x^n for each configuration n we can calculate the *globally* optimal configuration x^* as: $x^* = \arg\min_{n \in S}(f(x^n))$.

5.2.2 Proof of Concept

We prove that given a problem instance of *m1EVC* and given a configuration, we can reduce the resulting optimization problem to a problem instance of *EVC*, and thus can solve it by e.g. the water filling algorithm. We first formulate what a problem instance I_{m1EVC} of *m1EVC* given a configuration is, and construct a corresponding *EVC* instance. Then we prove that an optimal charging solution for

this newly defined problem defines a corresponding optimal charging solution for I_{m1EVC} given the configuration.

Let I_{m1EVC} be an instance of *m1EVC*, defined by a given number of intervals N, an amount of required charging C, an upper bound u_t for each interval t, and a minimum run-time of R. For the results in this section we assume a convex and separable cost function $f(x) = \sum_{t=1}^{N} f_t(x_t)$, since the results established here also hold for this more general case. A feasible configuration $n = (n_1, n_2, ..., n_k)$ for this instance is defined in accordance with Definition 5.2. We also define for each block j the set $\mathcal{I}_j^n$, which contains the indices of all intervals t that belong in block j.

Given such a configuration n, we construct the corresponding problem instance for *EVC* by merging all intervals of a block j together to form one new interval of 'length' n_j. For these new intervals we also define a new lower and upper bound, as well as a cost function. The new lower limit l_j^n will remain 0. The new upper limits u_j^n we define by:

$$u_j^n = n_j u,$$

and we define the cost function by:

$$f_j^{EVC}(x_j) = \sum_{t \in \mathcal{I}_j^n} f_t\left(\frac{x_j}{n_j}\right) \tag{13}$$

for all blocks $j = 1, ..., k$. This results in the following optimization problem:

$$\begin{aligned} I_{EVC}(n): \quad \min_x \quad & f^{EVC}(\mathbf{x}) = \sum_{j=1}^{k} f_j^{EVC}(x_j) \\ \text{s.t.} \quad & \sum_{j=1}^{k} x_j = C \\ & 0 \le x_j \le u_j^n \quad t = 1, ..., k. \end{aligned} \tag{14}$$

Note that the required capacity C remains the same. $I_{EVC}(n)$ thus has one interval j for each block of intervals in configuration n, with a total of k intervals.

Let $I_{m1EVC}(n)$ denote the problem instance I_{m1EVC} under the additional constraint that configuration n must be followed. Given a solution x for $I_{EVC}(n)$, a corresponding solution for $I_{m1EVC}(n)$ is constructed in the following way:

$$\bar{x}_t := \frac{x_j}{n_j} \quad \forall t \in \mathcal{I}_j^n. \tag{15}$$

The solution $\bar{x}$ is a feasible solution for I_{m1EVC}, since it satisfies the minimum run-time constraint (the charging is equal for each interval in a block of configuration n), each interval charges within the bounds 0 and u (based on the construction of l_j^n and u_j^n), and finally a total charge of C is reached.

Now we prove the following result.

Result 5.1. *Given a problem instance I_{m1EVC} and a configuration n, a solution that is optimal for $I_{EVC}(n)$ also defines an optimal solution for $I_{m1EVC}(n)$, and this optimal solution can be constructed using Equation (15).*

The idea of the proof: We prove this by contradiction. We assume that the translated solution for $I_{m1EVC}(n)$ according to Equation (15) corresponding to an optimal solution to $I_{EVC}(n)$ is not

optimal for $I_{m1EVC}(n)$. This in turn implies there is at least one other *better* solution for $I_{m1EVC}(n)$. Taking this solution we can construct a corresponding feasible solution for $I_{EVC}(n)$, again using Equation (15). We show that this solution has a lower cost than the first solution to $I_{EVC}(n)$, which was assumed to be optimal. Hence this leads to a contradiction, and thus the result must be true.

The Complete Proof: Given a problem instance I_{m1EVC} and a feasible configuration n, we construct the corresponding problem instance $I_{EVC}(n)$ as in Equation (14). Let $x^* \in \mathbb{R}^k$ define an optimal solution for $I_{EVC}(n)$, meaning that $f^{EVC}(x^*) \leq f^{EVC}(y)$ for all $y \in \mathcal{F}^{EVC}$, where $\mathcal{F}^{EVC} \subseteq \mathbb{R}^k$ is the feasible solution region for $I_{EVC}(n)$.
Now assume that the result is false, meaning that the corresponding solution for $I_{m1EVC}(n)$, $\bar{x}^* \in \mathbb{R}^N$, is *not* optimal.
We construct $\bar{x}^*$ from x^* using Equation (15):

$$\bar{x}^*_t = \frac{x^*_j}{n_j} \quad \forall t \in [S_{j-1}, S_j]$$

Next, let $\mathcal{F}^n \subseteq \mathbb{R}^N$ denote the subspace of all feasible solutions for $I_{m1EVC}(n)$. We know that $\bar{x}^*$ is feasible and therefore in $\mathcal{F}^n$. It is also given that $\bar{x}^*$ is *not* an optimal solution for $I_{m1EVC}(n)$, therefore there must exist at least one solution $z \in \mathcal{F}^n$ such that $f(z) < f(\bar{x}^*)$.
We denote the total solution space of I_{m1EVC} by $\mathcal{F}$, and we note that $\mathcal{F}^n \subseteq \mathcal{F}$. So, since $\bar{x}^*, z \in \mathcal{F}^n$, also $\bar{x}^*, z \in \mathcal{F}$, and we can calculate the costs $f(\bar{x}^*) = \sum_{t=1}^{N} f_t(\bar{x}^*_t)$ and $f(z) = \sum_{t=1}^{N} f_t(z_t)$.
As z is under the constraint of configuration n, for any block j of configuration n the charging value z_t must the same for each interval t in the block. This knowledge can be used to rewrite the cost of z:

$$f(z) = \sum_{t=1}^{N} f_t(z_t) = \sum_{j=1}^{k} \sum_{t \in \mathcal{I}^n_j} f_t(z_t), \tag{16}$$

and since we know that $z_{t_1} = z_{t_2} = z'_j$ for all $t_1, t_2 \in \mathcal{I}^n_j$, for some value z'_j, we can write this as:

$$\begin{aligned} \sum_{j=1}^{k} \sum_{t \in \mathcal{I}^n_j} f_t(z_t) &= \sum_{j=1}^{k} \sum_{t \in \mathcal{I}^n_j} f_t(z'_j) \\ &= \sum_{j=1}^{k} \sum_{t \in \mathcal{I}^n_j} f_t\left(\frac{n_j z'_j}{n_j}\right) \\ &= \sum_{j=1}^{k} f^{EVC}_j(n_j z'_j) = f^{EVC}(n^T z'), \end{aligned} \tag{17}$$

using (13) for the last equation. $n^T z'$ is a vector of length k and defines a feasible solution to $I_{EVC}(n)$; all bounds are met due to the fact that z meets the bounds in I_{m1EVC}. Furthermore the amount that is charged over the timespan is:

$$\begin{aligned} \sum_{j=1}^{k} n_j z'_j &= \sum_{j=1}^{k} n_j \frac{\sum_{t \in \mathcal{I}^n_j} z_t}{n_j} \\ &= \sum_{j=1}^{k} \sum_{t \in \mathcal{I}^n_j} z_t \\ &= \sum_{t=1}^{N} z_t = C \end{aligned}$$

Let $z^* := n^T z'$. Then we have from (16) and (17) that

$$f(z) = f^{EVC}(z^*)$$

We know that $f(z) < f(\bar{x}^*)$ and

$$f(\bar{x}^*) = \sum_{j=1}^{k} f_j^{EVC}\left(\frac{x_j^*}{n_j}\right) = f^{EVC}(x^*),$$

which leads to the conclusion that

$$f^{EVC}(z^*) < f^{EVC}(x^*).$$

This contradicts the assumption that x^* was an optimal solution to $I_{EVC}(n)$. ■

In conclusion, if the *configuration* of an optimal solution is known, an optimal charging solution can be calculated efficiently using an existing algorithm for *EVC*.

This result gives part of a possible strategy to find an optimal solution for a problem instance of *m1EVC*. It implies that given an optimal configuration—i.e. a configuration for which the best charging profile is also a globally optimal solution—it is a relatively easy matter to calculate the optimal solution: the solution can be calculated using any efficient algorithm for Problem *EVC*, since we have proven that the problem *m1EVC* for a given a configuration is reducible to Problem *EVC*, using (14). Thus the remaining difficulty only lies in finding such an optimal configuration. Once this is achieved, there is an efficient solution method in place to find an optimal charging profile. In the next subsection we present a algorithm where this concept is used to calculate an optimal solution.

We have defined Problem $I_{EVC}(n)$ for every configuration n. Each of these problems has an associated optimal fill level λ^n defining the optimal solution x^n. This fill level can be calculated by for example the waterfilling algorithm. In simulations it was observed that often a lower fill level λ^n of a configuration results in a lower minimum cost $f(x^n)$. This raises the question whether there exists some monotone relation between configurations, meaning that $f(x^n) < f(x^{n'})$ implies either $\lambda^n < \lambda^{n'}$ or $\lambda^n \leq \lambda^{n'}$. However, in Example 5.3 we give a counterexample, showing that such a relation of monotonicity does not hold universally.

Example 5.3. *Consider a m1EVC instance with $N = 6$, $R = 2$, a baseload energy profile $p = (2, 1, 3, 2, 7, 2)$, costs $f_t = (x_t + p_t)^2$, and a required charge C of 3. We can enumerate all possible configurations, which are: $(6), (3, 3), (2, 4), (4, 2), (2, 2, 2)$, and calculate their optimal charging solution using the reduction to an EVC problem instance. For configuration $(4, 2)$ this results in the best solution having a dual value $\lambda_1 = 2.75$ and a cost of 85.25. For configuration $(2, 4)$ this results in the best solution having a dual value $\lambda_2 = 3$ and a cost of 84.5. Thus, the dual value for the second configuration is higher, while the cost is lower than for the first configuration.*

Finally, we can make the following observation, to ignore some configurations that are for certain superfluous for the calculations:

Lemma 5.1. *For any configuration $n \in S$ with at least one block of at least length $2R$ there exists another configuration $n' \in S$ such that $f(x^{n'}) \leq f(x^n)$.*

This lemma implies that for any such configuration n there exists another configuration for which the best charging solution $x^{n'}$ has a cost that is at least as low as the cost of solution x^n.

Proof: Let $n = [n_1, ..., n_k] \in S$ have a block i with $n_i \geq 2R$. We construct another feasible configuration n' by splitting block i into two new feasible blocks. This is possible because we know the length

of block i is at least $2R$, so there is at least one way that the intervals in block i can be distributed over two blocks, i_1 and i_2, such that $n_{i_1} \geq R$ and $n_{i_2} \geq R$. This results in a feasible configuration $n' = [n_1, ..., n_{i_1}, n_{i_2}, ..., n_m]$, $n' \in S$. Secondly we know that in solution x^n every interval in block i had the same charging value, which was optimal for configuration n. In solution $x^{n'}$ the intervals in blocks i_1 and i_2 can have a different charging value, but do not have to have this. This implies that $x^{n'}$ can never result in a worse (best) solution than x^n, implying that $f(x^{n'}) \leq f(x^n)$. ■

With this observation any configuration with at least one block i of length greater than $2R$ can be scrapped.

5.3 A First Algorithm for *m1EVC*

5.3.1 To Find an Optimal Configuration

In the last section we reached the conclusion that the remaining challenge is to calculate an optimal configuration. In this section we present an algorithm to this end, which involves enumerating the viable configurations in the search space. Once such a configuration can be calculated we have all the building blocks for a solution strategy for Problem *m1EVC*.

Given an instance of Problem *m1EVC*, we aim to search through the possible feasible configurations and check if they are optimal. The waterfilling algorithm is used to calculate the best charging solution for each configuration. Out of all these solutions then the best one can be selected, and this is an optimal solution to Problem *m1EVC*.

What is left to do is to enumerate all configurations. This translates to the problem of enumerating all different sequences of integers $(n_1, ..., n_k)$ where $\sum_{i=1}^{k} n_i = N$ and $n_i \geq R$ for all i. We denote the number of blocks in a configuration by k. For each k we calculate all viable configurations using a tree structure. In this tree structure there are $k+1$ levels. At level 0 lies the root node. Furthermore, level i represents the ith block of a configuration. We iteratively build the tree, starting at the root. We add for each possible length n_1 of the first block a branch. This branch has to cover a remaining length of $N - n_1$. We repeat this process to branch out to each possible length for the second block, given a length for the first block, and so on. An example of this structure in given in Figure 2.

For the variable k we have an upper bound of $\lfloor \frac{N}{R} \rfloor$, considering the minimum run-time R. By using Lemma 5.1 we can also determine a lower bound on the number k of blocks in a configuration, as all configurations containing a block n_i with $n_i \geq 2R$ do not have to be considered. If we have that $N \leq k(2R-1)$ then there is at least one configuration of length k where all $n_i \leq (2R-1)$, and so the condition is not met. Oppositely, if $N > k(2R-1)$ then there must be at least one block n_i in every configuration of length k for which $n_i \geq 2R$. This results in a lower bound of $k \geq \frac{N}{2R-1}$.

We implemented the algorithm by building this tree from the root node, where each node has a couple of arguments, besides the label indicating the length. The first of these arguments is the sum i of all block lengths in its path back to the root node, or in other words the number of intervals already used. This number is used to calculate all the possible block lengths branching from this node. The second argument denotes the *rest* value, which once it reaches 0 means the last node is added to this branch, which must have a length of $N - i$, i.e. the remaining tail of intervals must all be contained in this final block. So this algorithm creates m branching levels of blocks underneath the root node, and then adds 1 final node to each branch, meaning that m must be equated to the number of blocks minus 1 at the start. Pseudo-code for the used procedure is given in Algorithm 2.

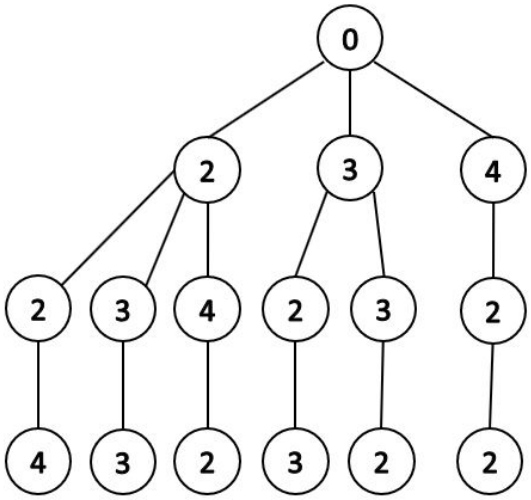


Figure 2: Example of a configuration tree for all configurations of size 3, where $N = 8$ and $R = 2$.

Algorithm 2 Enumerate Configurations

for $m = \lfloor \frac{N}{2R-1} \rfloor$ to $\lfloor \frac{N}{R} \rfloor - 1$ **do**
  Create root node: $(0, intervals = 0, rest = m)$
  Create queue *containing* root node
  while queue *is not empty* **do**
    Pop out the first node *in* queue
    i = node.intervals
    m' = node.rest
    if $m' > 0$ **then**
      for $l = i + R$ to $N - Rm' + 1$ **do**
        Create new node *with* node *as parent:* $(l - i, intervals = l, rest = m' - 1)$
        Add new node *to* queue
      end for
    else
      Create new node *with* node *as parent:* $(l - i, intervals = l, rest = 0)$
    end if
  end while
  Enumerate the path from each leaf node to the root and add each to the set of configurations
end for

We end this section by noting that for the trivial case of $R = 1$ the problem of enumerating all configurations is equivalent to the problem of finding the powerset for a set of size $N - 2$: all intervals can either be the beginning index of a block, or not, so this powerset encodes the set of all possible configurations for a problem instance of size N. This trivial case therefore results in a total of 2^{N-2} configurations, a number exponential in the size of the problem instance, N. Within the scope of this research we did not find a closed-form expression for the number of configurations depending on R and N for $R > 1$.

5.3.2 The Algorithm

Once we have enumerated all configurations to be considered, we calculate for every configuration calculate the best solution. The overall best solution is the optimal solution for Problem *m1EVC*. Our implementation of this algorithm solves Problem *m1EVC* for the cost function $f_t = (x_t + p_t)^2$. We use the waterfilling algorithm as presented in Section 3.2.2 to solve each of these optimization problems. To this end we define the parameters of the relevant cost function ($f_j = \frac{1}{2}a_j x_j^2 + b_j x_j + c_j$) for the waterfilling algorithm, for each 'interval'. First of all we can observe that the parameters c_j are not used in the waterfilling algorithm, and so these can be removed. We define a_j and b_j using the transformation in (13):

$$a_j = \sum_{t \in \mathcal{I}_j^n} \frac{2}{n_j^2} = \frac{2}{n_j} \tag{18}$$

$$b_j = \sum_{t \in \mathcal{I}_j^n} \frac{2p_t}{n_j} = \frac{2\sum_{t \in \mathcal{I}_j^n} p_t}{n_j}$$

With these parameters the waterfilling algorithm as it is defined in section 3.2.2 can be used for each configuration. The best solution is selected, and this gives a global optimum for Problem *m1EVC*.

5.4 Battery Charging

In Chapter 3 both problem *m1EVC* and problem *m1BC* were presented (see (8) and (9)). Analogous to Chapter 4—where the optimisation strategy for problem *dm1EVC* had a straightforward extension to create an optimisation strategy for problem *dm1BC*—it is an interesting question whether the new solution strategy we presented for *m1EVC* can be extended to a strategy for problem *m1BC*. We discuss the possibilities in this section.

A first thing to note is the existence of efficient solution strategies for problem *BC*, see for example Chapter 5 in [van der Klauw, 2017]. This means that the setup of the strategy for *m1EVC*—transform a certain configuration into an equivalent instance of *EVC*—has a possibility for success: if we can transform an *m1BC* instance given a certain configuration into an equivalent *BC* instance, we have an approach to calculate the optimal solution for this configuration. The question is whether such a transformation is possible.

The simple answer to this question is: No. It is not possible to follow the same steps as for the *m1EVC* strategy to achieve an equivalent transformation for *m1BC*. The reason for this lies in the cumulative constraint of Problem *m1BC*:

$$B_l \leq \sum_{t=1}^{l} x_t \leq C_l \quad l = 1, ..., N. \tag{19}$$

Given a configuration, the aim is to combine all the intervals contained in one block j of this configuration to one interval (or one variable) for our equivalent *BC* problem instance. In order to do this we need to translate the cumulative constraint of every interval t to a set of cumulative constraints for the new intervals j, of the form $B_l \leq \sum_{j=1}^{l} z_j \leq C_l \quad l = 1, ..., k$. The following example shows that this exact form is not possible for this transformation.

Example 5.4. *Consider an m1BC instance with $N = 6$, $R = 2$, and cumulative lower bounds B_l and upper bounds C_l for $l = 1, ..., 6$. Then consider the configuration $n = (3, 3)$. The aim is to transform the intervals of the first block of n (intervals $1, 2, 3$) to one interval with one cumulative constraint. Let*

z_1 be the variable for this new interval. As we know that for any solution z of the transformed problem we have that $x_1 = x_2 = x_3 = \frac{1}{3}z_1$, we can rewrite the constraint of $B_1 \leq x_1$ to $3B_1 \leq z_1$. Following this same reaonsing, we rewrite the constraints $B_2 \leq x_1 + x_2$ and $B_3 \leq x_1 + x_2 + x_3$ to $\frac{3}{2}B_2 \leq z_1$ and $B_3 \leq z_1$. This leads to a new lower bound of $B'_1 = \max(3B_1, \frac{3}{2}B_2, B_3)$. Similarly, we can define a new upper bound C'_1 as $\min(3C_1, \frac{3}{2}C_2, C_3)$. If we do the same procedure for the second block (this time using that $x_4 = x_5 = x_6 = \frac{1}{3}z_2$, and that $z_1 = x_1 + x_2 + x_3$), we again end up with three constraints for the lower cumulative bound: $B_4 \leq z_1 + \frac{z_2}{3}$, $B_2 \leq z_1 + \frac{2}{3}z_2$ and $B_3 \leq z_1 + z_2$. However, this set of bounds cannot be translated to one new cumulative constraint the way we did for the first block.

As the above example illustrates we cannot simply translate Problem *m1BC*(n) to an equivalent instance of Problem *BC*. However, the above example does show how we can translate Problem *m1BC* given a configuration n to a general linear program, since all the newly defined constraints will be linear. Such a linear optimization problem *can* be solved in polynomial time, by for example the simplex method or another polynomial-time linear programming solution method (see for example [Winston, 2003]). However since such a method is not tailored to the specific structure of our problem, this may be relatively inefficient, and it might be possible to develop a more streamlined strategy using the restrictions this program has. However, this direction is outside the scope of this research, but may be an interesting direction for future research.

5.5 Evaluation and Conclusion

The approach presented in Section 5.3.2 is guaranteed to calculate an optimal solution to Problem *m1EVC*. But there are some disadvantages to this approach, which we discuss here, and which are shown in the experimental results of Section 7.

The first disadvantage is perhaps the most obvious one, namely that this approach is a rather slow one: for each of the possible configurations the waterfilling algorithm has to be applied, and although the waterfilling algorithm is quite fast, running it so many times takes a while, and this running time grows quickly with N, because the number of possible configurations for a problem instance grows fast with N.

The second disadvantage has to do with the enumeration of all possible configurations. This requires a lot of memory to do and therefore, also depending on the implementation, it may not be practical for larger instances, even if time is not of the essence. This problem might be mitigated by not enumerating all possible configurations but focusing on promising ones, and thus arriving at only an approximation of the optimal solution.

Because of these disadvantages it is of interest to look into possible approximation methods as well. One promising approach is a local search approach, as the structure of the problem may lend itself to such a method quite well. We discuss the development of this method in the next chapter.

6 Local Search

In the previous chapter an exact approach to solve the optimization problem *m1EVC* ((8)) was presented. This approach was implemented and some results are presented in Section 7. The brute-force approach is rather heavy computationally due to the need to enumerate a significant amount of possible configurations. In this chapter we aim to approximate the optimum of an instance of Problem *m1EVC*. The setup of the problem as it is now, with configurations and a corresponding optimal solution, might lend itself well to a local search approach: if we define a candidate solution as being a feasible configuration, with attached the costs of the corresponding best solution, one can search the 'neighbourhood' of this configuration for a configuration with lower costs, and so, repeating this process until convergence, approach a local optimum.

In the next section a general overview of the different aspects of the local search algorithm is given, after which these different aspects are discussed in some more detail in the sections following. Finally, the implementation of the local search algorithm is presented.

6.1 Overview

The ingredients of the local search approach are as follows:

- The initialisation: An initial configuration is selected. Multiple strategies for this initialization can be tried, to test whether there are clearly better or worse strategies.
- The neighbourhood: the neighbours of a given configuration must be defined.
- The gain: For every neighbour a gain must be calculated. If the gain is positive, this neighbour is a candidate for the new best (up-till-now) solution. The gain can be calculated exactly or it can be approximated.
- At the end of an iteration, when all neighbours have been considered, a new best candidate is selected (if there is one). For this candidate the exact costs must be known before starting the next iteration.

To develop such a local search algorithm we first define formally what the neighbourhood of a configuration is. It is also proven that this definition is such that the entirety of the solution space is connected by the neighbourhood operation. Some useful observations of the solution structure of *m1EVC* will be discussed; these can be utilized for an efficient local search implementation. Two different methods to make an estimation of the gain of a neighbour will be discussed.

6.2 The Neighbourhood of a Configuration

In this section a set of neighbourhood operations are defined for the local search approach, and we prove that these operations connect the entire solution space of the problem.

6.2.1 The Neighbourhood Operations

The first two operators we define are the + and - operators. These two operators allow one interval of a selected block to shift one block. With the + operator the leftmost interval of a selected block shifts to the adjacent block to the left. With the - operator the rightmost interval of the selected block shifts to the adjacent block to the right. In the case that this would result in an infeasible configuration, which happens when the block losing an interval now has a length smaller than R, a feasible configuration is reached by shifting the leftmost or rightmost interval of this block to the next adjacent block too, respectively.

A configuration is again defined as being a vector $(n_1, n_2, ..., n_k)$ with the properties as described in Definition 5.2. Let $n = (n_1, ..., n_k) \in S^k$ define a certain feasible configuration. Formally, the + and - operators are defined as follows:

Definition 6.1 (+ Operator)**.** *For a given block i in configuration n, let $j > i$ be the first block such that $n_j > R$ (this implies that for all blocks l inbetween $(i < l < j)$ it holds that $n_l = R$)j.*
Then $+(n, i) := (n_1, ..., n_{i-1}, n_i + 1, ..., n_j - 1, n_{j+1}, ..., n_k)$

Definition 6.2 (- Operator)**.** *Given block i in configuration n with $n_i > R$. Let j be a block such that $i < j$ and for all blocks l inbetween $(i < l < j)$ it holds that $n_l = R$.*
Then: $-(n, i) = (n_1, ..., n_{i-2}, n_i - 1, ..., n_j + 1, n_{j+2}, ..., n_k)$

Next we also define a join and a split operator, to enable a jump from level S^k one level up or down. To jump one level down to a configuration in S^{k-1}, two blocks of a configuration are joined together. To jump one level up to S^{k+1} one block is split into two blocks, while maintaining feasibility, meaning that all blocks must still have a length of at least R after the operation. If this is not possible (because there is no block of length $2R$ or more), then we define the operation such that a block of maximum length is split, and a next block is split until a minimum of R intervals is available to create a new block of at least length R.

The two operations are formally described below:

Definition 6.3 (Join Operator $(S^k \to S^{k-1})$)**.** *For a given block $i < k$ in configuration n, we define: $join(n, i) = (n_1, n_2, ...n_{i-1}, n_i + n_{i+1}, n_{i+2}..., n_k) \in S^{k-1}$*

Definition 6.4 (Split Operator $(S^k \to S^{k+1})$)**.** *i) For a given block i in configuration n with $i \in argmax_j n_j$ and $n_i \geq 2R$, let $n'_i, n_* \geq R$ be two integers such that $n'_i, n_* \geq R$ and $n'_i + n_* = n_i$.*
Then: $split(n, i) = (n_1, n_2, ..., n_{i-1}, n'_i, n_*, n_{i+1}, ..., n_k)$. This is called the simple split operation.

ii) If no $n_i \geq 2R$ exists we let $i \in argmax_j n_j$, and we define two suboperators: split+ and split-, which use the blocks immediately to the left or to the right of the considered block respectively to split off enough intervals in total to create one new feasible block:

For split+ choose r such that $\sum_{j=0}^{r}(n_{i-j} - R) \geq R$, and let this be the smallest r for which the inequality holds. We set the new length $n'_j = R$ for each block $j \in \{i - r, ..., i\}$ and $n'_j = n_j$ for all other $j \in \{1, ..., k\}$, and we define the new length $n'_{k+1} = \sum_{j=0}^{r}(n_{i-j} - R)$.
Then we define: $split + (n) = n'$

For split- choose r such that $\sum_{j=0}^{r}(n_{i+j} - R) \geq R$, and let this be the smallest r for which the inequality holds. We set the new length $n'_j = R$ for each block $j \in \{i..., i + r\}$ and $n'_j = n_j$ for all other $j \in \{1, ..., k\}$, and we define the new length $n'_{k+1} = \sum_{j=0}^{r}(n_{i+j} - R)$.
Then we define: $split - (n) = n'$

Note that the *join* operator can never improve the cost, i.e. never result in a positive gain: the best possible gain for this operation is 0. The other way around the same principle holds for the simple split operator: For this operator there can never be a gain smaller than 0. The reasoning for these properties is similar to the proof of Lemma 5.1: in case two blocks of configuration n are joined by the join operator before the operation there was more freedom for the charging within the intervals in these blocks than after the operation. The opposite is true for the simple split operator: here the intervals of the two new blocks have more freedom in their charging than before, when they were contained in one block.

6.2.2 Connectivity

Let S be the space containing all feasible configurations n. Above, specific operations on a configuration have been defined. Now we aim to investigate whether these operations together connect the

entire space S. This would be desirable since then the local search algorithm in principle is able to search the entire solution space using only these operations.

To prove that this is indeed the case let $n \in S$ be a randomly chosen feasible configuration of length k, and let $n' = (n'_1, ..., n'_{k'}) \in S$ be another random feasible configuration of length k'. If the four defined operations, +, -, *join*, and *split*, can together define a path from n to n' this would indeed show that these operations connect the entirety of S. To prove this, we perform the following operations, starting with configuration n:

1. Apply the *join* operator until no two intervals can be joined together anymore (this occurs after $k-1$ steps). The result is the configuration (N) of length 1.
2. To this configuration (N) apply the simple *split* operator until configuration n' is reached by first splitting off $n'_{k'}$, then $n'_{k'-1}$ and so on, until configuration n' is reached in $k'-1$ steps.

This implies that any two configurations in S have at least one path connecting them by the four defined operations, implying that the operations connect S. Note that the above shows that already the two operators *join* and *split* are sufficient to prove connectivity. However, we have defined the + and - operators in order to shorten the path between two configurations in S, as the procedure used above to prove connectivity involves a lot of steps to connect two configurations.

6.3 Calculation of the Gain

We consider the application of the local search approach to Problem *m1EVC* with the cost function again defined as $f_t(x) = (x_t + p_t)^2$ for an interval t, and the total objective value as $f(x) = \sum_{t=1}^{N} f_t(x)$. Given a candidate configuration n we can calculate its optimal solution x^n and the corresponding cost $f(x^n)$ using the method developed in Sections 5.2.2 and 5.3.2. Now, the next step is to search the neighbourhood of this candidate for a configuration with a positive gain. For this there are two possibilities: we can use our exact method to also calculate the cost $f(x^{n'})$ of a neighbouring configuration n', or we can make an estimation of this cost to save time. The latter option may be a significant improvement on the calculation time for larger instances, since in each iteration the gain of all neighbouring configurations must be calculated and compared. In this section we first elaborate on the exact approach, and then we explore an idea for estimating the gain for each neighbour.

6.3.1 Exact Gain Calculation

From Chapter 5 we know how to calculate the optimal charging solution x^n for a given configuration n and the associated cost $f(x^n)$, using the waterfilling algorithm. This way we can calculate for any neighbour the corresponding exact gain.

To recap Section 3.2.2, the waterfilling algorithm functions with a fill level $\hat{\lambda}$. Once the correct fill level is reached we calculate the corresponding charging solution using Equation (3). So if we look at the candidate solution within local search there is a current candidate configuration n with an associated fill level λ^n calculated by the waterfilling algorithm, by solving Problem $I_{EVC}(n)$ (see (14)). Using the fill level λ^n a charging solution $\bar{x}^n$ of problem instance I_{m1EVC} follows, with a cost of $f(\bar{x}^n)$. Now to calculate the gain of every neighbour of this current candidate solution n we must repeat this approach to calculate the λ value of each neighbour and use this value to calculate the gain of the neighbour.

However, we can improve this process by using the properties of the neighbouring solutions. To this end we investigate the different cases for the relation between the candidate solution and its neighbours, and what the implications are for each of these cases. This enables us to develop a more efficient exact method for calculating the gain of each neighbour.

We start by looking at the + operator as an example of a neighbouring operator. The arguments for the other neighbourhood operations are similar, and are shortly discussed at the end of this section. Now suppose we have the current candidate configuration n and we have a neighbouring configuration n' that is reached by applying the + operator to some feasible pair of blocks i and j of n, i.e. n' looks like:

$$n' := (n_1, ..., n_i + 1, ..., n_j - 1, ..., n_k)$$

Note that λ^n is the optimal fill level of Problem $I_{EVC}(n)$, and let x^n be the corresponding optimal solution for Problem $I_{EVC}(n)$ (see section 5.2.2 for a more detailed explanation of problems $I_{m1EVC}(n)$ and $I_{EVC}(n)$). Furthermore, let C denote the total required charging.

One can distinguish between three cases for the relation between the current λ^n and the optimal fill level $\lambda^{n'}$ of the neighbour n'. For each of these cases we can characterize the gain, using the relation between λ^n and $\lambda^{n'}$. To this end let Δ_f denote the gain if we would move from configuration n to configuration n'. Note that the proof of Result 5.1 shows that the objective values of $f(\bar{x}^n)$ and $f^{EVC(n)}(x^n)$ (the objective value of the transformed problem $I_{EVC}(n)$) are equal, meaning that we can conclude that the gain Δ_f is equal to $\Delta_{f^{EVC}}$, where $\Delta_{f^{EVC}} = f^{EVC(n)}(x^n) - f^{EVC(n')}(x^{n'})$, which is the difference in cost between the optimal solution of problem instance $I_{EVC}(n)$ and the optimal solution of problem instance $I_{EVC}(n')$.

Case 1: $\lambda^n = \lambda^{n'}$, $j = i + 1$
If $j = i+1$ then the only difference between configuration n and configuration n' lies in the two blocks i and j. All other blocks of the two configurations remain the same, meaning that the parameters of the cost functions $f^{EVC(n)}$ and $f^{EVC(n')}$ are equal for these blocks: $a_l^n = a_l^{n'}$ and $b_l^n = b_l^{n'}$ (see Equation (18) for the definition of these parameters). This together with the fact that $\lambda^n = \lambda^{n'}$ implies that $x_l^{n'} = x_l^n$ for all blocks l (making use of Equation (3)), except for blocks i and j. This in turn implies that $f_l^{EVC(n)x_l^n} = f_l^{EVC(n')x_l^{n'}}$ for those blocks when jumping from configuration n to n'. Therefore the gain of neighbour n' depends only on blocks i and j, and we have $\Delta_f = f_i^{EVC(n)}(x_i^n) - f_i^{EVC(n')}(x_i^{n'}) + f_j^{EVC(n)}(x_j^n) - f_j^{EVC(n')}(x_j^{n'})$.

Case 2: $\lambda^n = \lambda^{n'}$, $j > i + 1$
This case is almost the same as Case 1, but now all blocks between i and j also change compared to configuration n. Δ_f can be written as:

$$\begin{aligned}\Delta_f &= f_i^{EVC(n)}(x_i^n) - f_i^{EVC(n')}(x_i^{n'}) + f_{i+1}^{EVC(n)}(x_{i+1}^n) - f_{i+1}^{EVC(n')}(x_{i+1}^{n'}) + ... + f_j^{EVC(n)}(x_j^n) - f_j^{EVC(n')}(x_j^{n'}) \\ &= \sum_{s=i}^{j} f_s^{EVC(n)}(x_s^n) - f_s^{EVC(n')}(x_s^{n'}) \qquad (20)\end{aligned}$$

Case 3: $\lambda^n \neq \lambda^{n'}$
This implies that either $\lambda^n > \lambda^{n'}$ or $\lambda^n < \lambda^{n'}$. In either case we cannot find any direct expression for a simplification of Δ_f. Thus, the new solution $x^{n'}$ and the cost $f(x^{n'})$ must be calculated, and $\Delta_f = f(x^n) - f(x^{n'})$.

The arguments made above for the + operator can be made in the same fashion for the other three operators, where in each case only the objective values of the blocks transformed by the operation are relevant for the change in the total cost, if $\lambda^n = \lambda^{n'}$.

Above, we have described how we can calculate the gain provided we know whether $\lambda^n = \lambda^{n'}$. It is relatively straightforward to check whether $\lambda^n = \lambda^{n'}$: given λ^n and configuration n' we can use

Equation (3) to calculate a candidate solution where λ^n is our 'guess' for $\lambda^{n'}$. According to the KKT conditions this solution is an optimal solution to $I_{EVC}(n')$ if it holds that the total charging is equal to C. It can be shown for the specific problem considered in this research (Problem *m1EVC* with a cost function of the form $\sum_t (x_t + p_t)^2$), that it frequently occurs that $\lambda^n = \lambda^{n'}$. A substantiation of this observation is given in the appendix at the end of this chapter, in Section 6.6. Here it is proven that for two configurations for which intervals are in the same state (off, active, or full) in the optimal solutions of the two configurations, then also the optimal fill level is equal.

If this is not the case, i.e. if $\lambda^n \neq \lambda^{n'}$, then $\lambda^{n'}$ must be calculated in order to determine the gain Δ_f. In this case using Equation (3) with λ^n as the 'guess' for $\lambda^{n'}$ resulted in either a charge $\hat{C} < C$ or $\hat{C} > C$, depending on whether $\lambda^n > \lambda^{n'}$ or $\lambda^n < \lambda^{n'}$ respectively. It is possible to use an alternative approach to the waterfilling method where we use λ^n as the initial fill level instead of the first breakpoint, and where $\Delta C = \hat{C} - C$ is the target charge. Now λ^n can be either increased or decreased until the correct charge C is reached. This results in the optimal solution to $I_{EVC}(n')$ and the gain of configuration n' can be calculated. This alternative approach can improve the calculation time if the values of the two optimal fill levels are generally close-by. This 'midpoint waterfilling' approach is compared to the traditional waterfilling algorithm in Section 7.4.5.

6.3.2 Estimating the Gain

In this section we present a method to estimate the gain and improve the run-time of the local search algorithm. Within this method the estimation is used to select the neighbouring configuration with the best gain. Once this neighbour is selected we still use the exact approach of the waterfilling algorithm to calculate the real cost of the new candidate configuration.

Firstly, we have observed that there are cases where $\lambda^n = \lambda^{n'}$. In these cases we can easily calculate the exact optimal solution $x^{n'}$ using Equation (3) and there is no need for an estimate instead. When $\lambda^n \neq \lambda^{n'}$ we have either that $\lambda^n > \lambda^{n'}$ or that $\lambda^n < \lambda^{n'}$, and the planning $\tilde{x}^{n'}$ calculated from Equation (3) using λ^n results in $\hat{C} < C$ or $\hat{C} > C$ respectively. This means results in an error in the total charge of $\Delta C = \hat{C} - C$.

We use the guess $\tilde{x}^{n'}$ made with λ^n. We make an estimate of the real optimal planning $x^{n'}$ by distributing the remaining charge ΔC over the active blocks of solution $\tilde{x}^{n'}$. This is the same procedure as the waterfilling algorithm, with the difference being that in the waterfilling algorithm the active blocks are only increased in charge until the next breakpoint, where the set of active blocks changes. This means that this method of estimating may result in a solution that is infeasible, because a block receives a charge that is higher than its maximum charging rate. Note that the bigger ΔC is, the bigger the difference to the true optimal cost $f(n')$ might be.

The estimation of the gain Δ_f now consists of 2 steps. The first step is analogous to Case 2: if $\lambda^n = \lambda^{n'}$ then Δ_f is given in Equation (20). If this is not the case we make an estimation using the current guess for the solution, $\tilde{x}^{n'}$. We calculate the error in the total charge, ΔC, and we define $\mathcal{Y}$ to contain the indices of all active blocks in $\tilde{x}^{n'}$, and $\bar{\mathcal{Y}}$ the indices of all other blocks. ΔC is distributed over the active blocks, where each active block l should be changed relative to its increase factor $\frac{1}{a_l}$, which means we estimate the cost of the optimal solution as:

$$\tilde{f}(x^{n'}) = \sum_{l \in \bar{\mathcal{Y}}} f(\tilde{x}_l^{n'}) + \sum_{l \in \mathcal{Y}} f\left(\tilde{x}_l^{n'} + \frac{1}{a_l} \frac{\Delta C}{\sum_{i \in \mathcal{Y}} \frac{1}{a_i}} \right). \tag{21}$$

Now the gain is estimated as $\tilde{\Delta}_f = f(x^n) - \tilde{f}(x^{n'})$.

This estimate is compared to the exact method in Section 7.4.6, where both the solution quality and the time of the algorithms are compared.

6.4 The Local Search Algorithm

Above, we have already discussed some of the important aspects of the local search algorithm. Some other aspects include the initialization of the algorithm and the choice of the neighbourhood operator. For the initialization some different possibilities are considered in Chapter 7. Hereby multiple deterministic initializations are discussed in Section 7.4.1 and a random initialization strategy is tested in Section 7.4.6.

For the neighbourhood operator we presented four basic versions in Section 6.2. In our implentation we use the + and - neighbourhood operators to search within a certain level S^k and select the best neighbour of the configuration n on this level. We use the *split* operator to search the neighbours of a configuration one level above, i.e. in level S^{k+1}. Finally we have the *join* operation. As mentioned at the end of Section 6.2 this operator can never result in a positive gain for a neighbour, in the way it is defined now. This leads us to perform a second operation whenever the *join* operator is applied to the candidate configuration, namely a *split* operation, and only after these two subsequent operations the gain is calculated, meaning that the 'neighbours' are achieved not by one but by two neighbourhood operations applied to the current candidate when the *join* operator is selected. We summarize this in Algorithm 3.

Algorithm 3 Local Search

Select initial configuration n
Calculate solution x^n and cost $f(n)$ of initial configuration

for K iterations **do**
  Randomly select neighbouring operation (+/- or join or split)

  for every neighbour n' **do**
    calculate $\tilde{x}^{n'}$
    if $\sum \tilde{x}^{n'} = C$ **then**
      Calculate the gain using Equation (20)
    else
      Calculate the gain using the waterfilling algorithm, or estimate the gain using Equation (21)
    end if
  end for

  if $\Delta_f > 0$ for some n' **then**
    Select the neighbour with the largest gain as the new candidate solution. For this configuration, if the gain was estimated, calculate the corresponding optimal solution and cost.
  end if

end for

6.5 Conclusion

In this chapter we developed a local search approach to approximate the optimal solution to an *m1EVC* problem instance, where we both presented a strategy to calculate the gain exactly for every neighbour, and a strategy to estimate this gain. In Chapter 7 some simulation results for this approximation method are presented. The remainder of this chapter is dedicated to a deeper analysis of the config-

urations and their λ values, in particular we determine some conditions for two configurations n and n' where we have that $\lambda^n = \lambda^{n'}$.

6.6 Appendix: Flow Equality of Configurations

It is notable that for Problem *m1EVC* with cost function $f = \sum_t (x_t + p_t)^2$ it is often the case that for different configurations n and n' the optimal fill levels of $I_{EVC}(n)$ and $I_{EVC}(n')$ are equal. In this section we prove some of the conditions under which this behaviour occurs. To this end we start with a specific case, proving that when all intervals are *active* in both x^n and $x^{n'}$ it is so that $\lambda^n = \lambda^{n'}$. Then we generalize this result further.

We define a problem instance I_{m1EVC} of Problem *m1EVC* with N intervals, a set of interval baseloads p_t, a maximum charge of u for each interval t, a required charge C, and a minimum run-time/off-time R.

6.6.1 A First Result

We start with the case where $n = (n_1, ..., n_k)$ is a configuration where every block is active in the optimal charging plan x^n of Problem $I_{EVC}(n)$, meaning that $0 < x_i^n < u_i^n$ for all blocks i of n. For this case we show that the value λ^n is independent of the configuration n, so independent of n_i for all blocks i. This then implies that for any other configuration n' where all blocks are also *active* in the optimal solution $x^{n'}$, it holds that $\lambda^{n'} = \lambda^n$.

Result 6.1. *Given an instance of Problem m1EVC and given that configuration n is such that $0 < x_i^n < u_i^n$ for all blocks i (where x^n defines the optimal solution to Problem $I_{EVC}(n)$ as given in (14), it is so that the optimal fill level λ^n is independent of the configuration $n = (n_1, ..., n_k)$, and*

$$\lambda^n = 2\left(\frac{C + \sum_{t=1}^{N} p_t}{N}\right).$$

***Idea of the proof**: we use the special-case KKT conditions for this problem to express λ^n in the parameters of the problem instance, and then we show that when all intervals are active this expression is not dependent on any parameters related to the configuration $(n_1, ..., n_k)$.*

The Proof: let j be the index of some block of configuration n. Using Equation (3) and the fact that we know that x_j^n is *active* we can express λ^n as:

$$\lambda^n = a_j^n x_j^n + b_j^n.$$

We use equation (18) to express a_j^n and b_j^n, resulting in,

$$\lambda^n = \frac{2x_j^n}{n_j} + \frac{2\sum_{t \in \mathcal{I}_j^n} p_t}{n_j},$$

or

$$n_j \lambda^n = 2x_j^n + 2\sum_{t \in \mathcal{I}_j^n} p_t.$$

We know that this relation is true for every block j, and therefore we can write

$$\lambda^n \sum_j n_j = 2\sum_j x_j^n + 2\sum_j \sum_{t\in\mathcal{I}_j^n} p_t$$
$$\iff N\lambda^n = 2C + 2\sum_t p_t$$
$$\iff \lambda^n = 2\left(\frac{C + \sum_{t=1}^N p_t}{N}\right) \blacksquare$$

6.6.2 A Generalised Result

We can use the same principle to prove a more general result. We again denote by x^n the optimal solution to Problem $I_{EVC}(n)$ for a certain configuration n and by $\bar{x}^n$ the corresponding optimal solution to Problem $I_{m1EVC}(n)$. Let P_A^n denote the set of all blocks *active* in x^n, so for which $0 < x_j^n < u_j^n$, and similarly let P_F^n denote the set off all *full* blocks, so where $x_j^n = u_j^n$, and P_0 the set of all *inactive* blocks, so where $x_j^n = 0$.

Result 6.2. *Given an instance of Problem m1EVC and a configuration n, we have that*

$$\lambda^n = \frac{C + \sum_{j\in P_A^n}\sum_{t\in\mathcal{I}_j^n} p_t - u\sum_{j\in P_F^n} n_j}{\frac{1}{2}\sum_{j\in P_A^n} n_j}.$$

Interpretation: This relation shows that the value of λ^n depends on the sets P_A^n and P_F^n, but not on the specific configuration n. The summations in the expression will be equal for two configurations n and n' when the sets P_A^n and P_F^n contain the same intervals, and so in this case λ^n would be equal to $\lambda^{n'}$.

The Proof: Given n and the corresponding optimal solution x^n to Problem $I_{EVC}(n)$ we have for each block j that it is either *inactive*, in which case $x_j^n = 0$, it is *active*, in which case the following relation holds:

$$\lambda^n = a_j^n x_j^n + b_j^n = \frac{2x_j^n}{n_j} + \frac{2\sum_{t\in\mathcal{I}_j^n} p_t}{n_j},$$

or it is *full*, in which case $x_j^n = u_j^n = n_j u$.

First of all, considering the set of all *active* blocks P_A^n, we can write, following a similar procedure as the previous proof:

$$\lambda^n \sum_{j\in P_A^n} n_j = 2\sum_{j\in P_A^n} x_j^n + 2\sum_{j\in P_A^n}\sum_{t\in\mathcal{I}_j^n} p_t$$
$$\iff \lambda^n \sum_{j\in P_A^n} n_j + 2\sum_{j\in P_F^n} x_j^n = 2\sum_{j\in P_A^n} x_j^n + 2\sum_{j\in P_F^n} x_j^n + 0 + 2\sum_{j\in P_A^n}\sum_{t\in\mathcal{I}_j^n} p_t$$
$$\iff \lambda^n \sum_{j\in P_A^n} n_j + 2u\sum_{j\in P_F^n} n_j = 2\sum_{j\in P_A^n} x_j^n + 2\sum_{j\in P_F^n} x_j^n + 2\sum_{j\in P_0^n} x_j^n + 2\sum_{j\in P_A^n}\sum_{t\in\mathcal{I}_j^n} p_t$$
$$\iff \lambda^n \sum_{j\in P_A^n} n_j = 2C + 2\sum_{j\in P_A^n}\sum_{t\in\mathcal{I}_j^n} p_t - 2u\sum_{j\in P_F^n} n_j,$$

which indeed gives us,

$$\lambda^n = \frac{C + \sum_{j \in P_A^n} \sum_{t \in \mathcal{I}_j^n} p_t - u \sum_{j \in P_F^n} n_j}{\frac{1}{2} \sum_{j \in P_A^n} n_j} \blacksquare$$

7 Results

In this section the efficiency of the methods developed in this research is tested and their properties are evaluated via simulation. We first discuss the simulation results for the dynamic programming approach for Problem *dm1EVC* developed in Chapter 4. The remainder of this chapter focusses on the *m1EVC* problem. We first discuss the observed shortcomings of the exact approach developed in Chapter 5, and subsequently we discuss the simulation results of the local search approach developed in Chapter 6.

7.1 Testing Parameters

The parameters that together define a problem instance of Problem *m1EVC*, as defined in (8), are the maximum charging rate u, the baseloads of each time interval p_t, the number of intervals N, the minimum run-time R, and the required charge C. The parameters N, R and C are assigned different values throughout this chapter. For the maximum charging rate we use a rate of 7400 Watt. Each interval has a length of 1 minute, meaning that the maximum charging rate of each interval is equal to $\frac{7400}{60} \approx 123$ Wh.

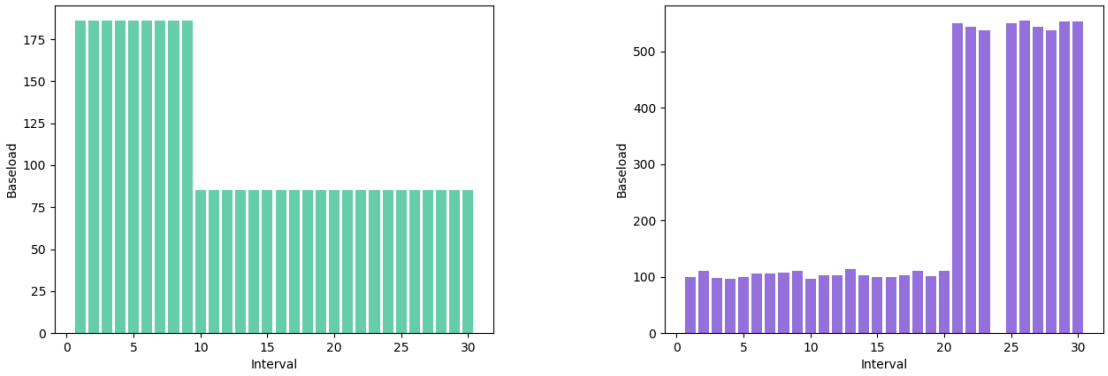
The parameters that define a problem instance of Problem *dm1EVC*, as defined in (6), are the set of discrete charging rates $\mathbf{Z}$, the baseloads of each time interval p_t, the number of intervals N, the set of minimum run-times $\mathbf{R}$, and the required charge C. For every test instance we use a subset of $\{0, 1..., 12\}$ Wh as the set of charging rates $\mathbf{Z}$, where it varies which subset is used in a simulation based on the properties we want to test. Note that multiplying all rates and the required charge C by some positive integer results in a problem instance that is equivalent to this problem instance, but potentially have a greater calculation time. This is the reason we selected these relatively low charging rates, while keeping all rates integer.

We generated testing data for the baseload input in three different manners. Firstly, to see the performance of the various algorithms in a setting relatively close to reality we generated some household energy profiles using the open source tool presented in [Hoogsteen et al., 2016]. These energy profiles are used as the baseloads p_t. Secondly we created some more 'difficult' instances. These were generated by creating bigger peaks and valleys, and by adding some random 'spikes' in these peaks and valleys, where a spike denotes a single interval t where the baseload p_t is suddenly much higher (in case of a valley) or much lower (in case of a peak) than the surrounding intervals. The third collection of datasets consists of valleys with value 0 throughout, and peaks with an extremely high value throughout, and no spikes. These datasets are called the 'extreme datasets'. The reasons for creating these specific datasets is explained in more detail in Sections 7.4.2 and 7.4.4. We show an example of the first 30 intervals for a realistic and a difficult dataset in Figure 3.

We implemented these algorithms in Python (version 3.5.0), and all simulations were executed on a Lenovo laptop with Intel Core i7.

7.2 *dm1EVC* Dynamic Program

The dynamic programming approach presented in Algorithm 1 is guaranteed to calculate an optimal solution to Problem *dm1EVC*, and has a worst-case time complexity of $O(N^2M^2z^M)$. To evaluate the practical behavior of the algorithm, we conduct several simulations. In these simulations we assume that $R_j = R$ for every charging rate $z_j \in \mathbf{Z}$, in order to study at the dependency of the perfomance on the value of the minimum run-time. We incrementally increase the various parameters, N, M, z^M, and the minimum run-time R, and some results are presented in Figure 4. In this figure the average runtime is shown when varying the value of each parameter seperately.



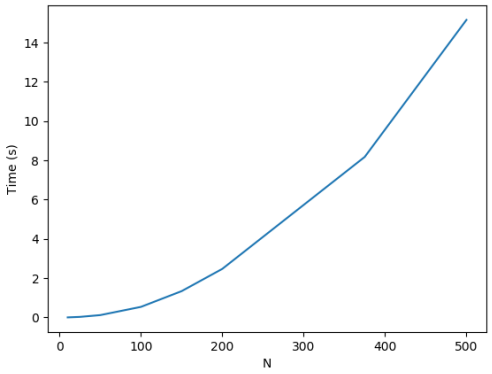
(a) *Realistic Dataset.* (b) *Difficult dataset.*

Figure 3: The baseloads of the first 30 intervals for two test instances.

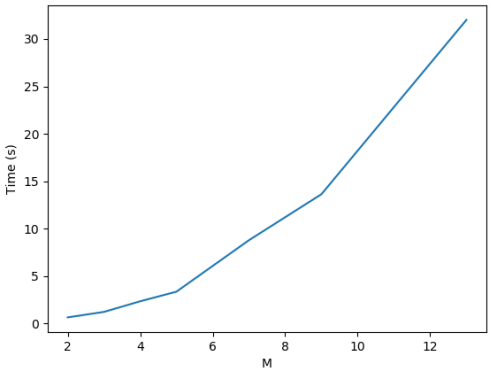
We see a roughly quadratic increase in run-time with an increase in N and M in the simulations, as the theoretical time complexity of $O(N^2M^2z^M)$ also implied. This also holds for the maximum charging rate z_M. We also varied the value of the minimum run-time R. While R is not presented in the theoretical time complexity we know that a greater value of R decreases the number of necessary calculations in Algorithm 1. We can see in Figure 4d that this is on average the case. Looking at the simulation results for each test instance we see that the first region, with R roughly between 3 and 20, there is a lot of variation in the run-time, though always the values are somewhere between 0.7 and 0.9 seconds. Only between $R = 20$ and $R = 30$ the run-time starts to truly decrease drastically. These relatively big values for the minimum run-time thus seem to simplify the problem by a large margin, as is in line with our expectations.

7.3 *m1EVC* Exact Algorithm

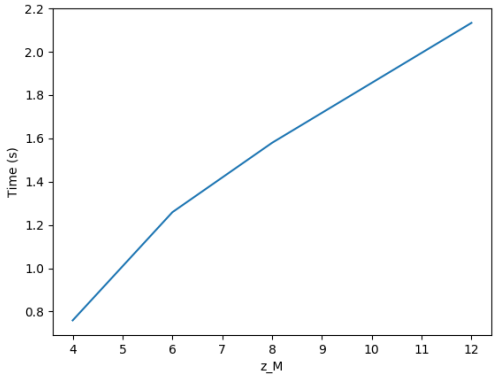
Algorithm 2 presented in Chapter 5 calculates an optimal solution to problem *m1EVC*. We expect this algorithm to perform quite poorly in terms of speed, because a large set of configurations must be enumerated, and the size of this set grows fast with the number of intervals N. Some tests were conducted to confirm this behaviour. The run-time of the algorithm quickly increases with an increase of N, as can also be observed in the simulation results shown in figure 5. In this figure the average runtime is shown when the number of intervals N is increased.



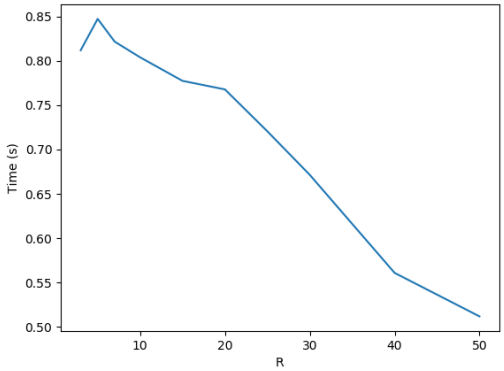
(a) *The run-time in seconds when the number of intervals N is increased. We have that $C = 2N$ always.*



(b) *The run-time in seconds when the number of charging rates M is increased, with always a maximum rate of $z_M = 8$ except for $M = 12$, where $z_M = 12$.*



(c) *The run-time in seconds when the maximum charging rate z_M is increased.*



(d) *The run-time in seconds when the minimum run-time R is increased.*

Figure 4: *Dependence of the performance of Algorithm 1 on its various input parameters, using the realistic simulation data. In the base case we always have that $N = 120$, $\mathbb{Z} = \{0, 2, 4\}$, and $C = 360$, and in each figure one particular parameter is varied.*

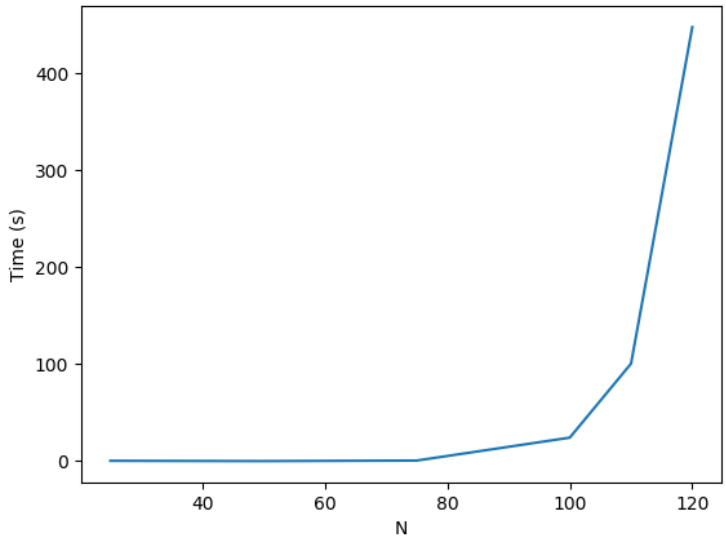


Figure 5: *Run-time of the enumeration algorithm for Problem m1EVC, for an increasing value of N. We have that $R = 15$ and $C = 10000$.*

The run-time of the algorithm increases relatively slowly, until there is a shocking jump going from $N = 110$ to $N = 120$. This fast increase in run-time can be explained by the fast-growing number of possible configurations, as is also shown in Table 1. The calculation time per configuration is not a bottleneck, as the waterfilling algorithm is an algorithm with a fast run-time, even for large instances, so it is the fact that such a charging solution has to be calculated for each one of so many configurations that makes this algorithm unsuitable for larger instances.

N	# of Configurations	run-time (s)
25	7	0.015
50	43	0.016
75	1359	0.565
100	40691	24.192
110	161480	100.493
120	630484	447.446

Table 1: *Number of considered configurations for each value of N (where Lemma 5.1 has been applied to cut off some irrelevant configurations).*

7.4 *m1EVC* Local Search Algorithm

In Chapter 6 we developed an approximation method for Problem *m1EVC* based on a local search approach. In this section we investigate whether such an approach works for this problem structure, and we study the initialization of the algorithm and the calculation of the gain. Finally, we also substantiate the claims made in Chapter 6 about the λ values of different configurations, both that they would generally be close in value, and that there are circumstances in which they are equal in value.

We use Algorithm 2 to calculate the optimal solution to some problem instances, as well as the worst solution (the charging solution for the worst configuration), in order to evaluate the solution quality

of the local search algorithm with an upper and lower baseline.

7.4.1 Initialization

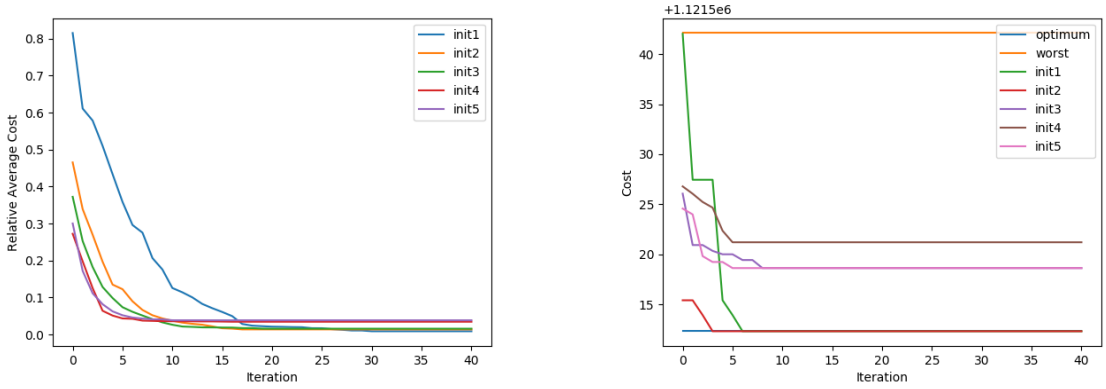
In this section we look at simulation results for five different initializations, which are given in Table 2. We have $N = 100$ and $R = 15$ for each test instance, and we used the realistic simulated baseloads as described in Section 7.1.

Name	Configuration
Init 1	$(50, 50)$
Init 2	$(25, 25, 25, 25)$
Init 3	$(20, 20, 20, 20, 20)$
Init 4	$(16, 16, 18, 18, 16, 16)$
Init 5	$(15, 18, 15, 19, 15, 18)$

Table 2: *The five different initial configurations for the local search algorithm.*

The results from these first simulations are shown in Figure 6. In this figure the average objective value is shown for each iteration of the algorithm, as well as one particular test instance where we saw some initializations converge to a local minimum. We see the local search algorithm converge to a solution within 40 iterations for all but 1 simulation, and in most cases the algorithm will converge within 20 iterations. In a lot of these simulations the algorithm converges to the global optimum, but there are also cases where the solution instead converges to a local optimum. An example of such an instance is shown in Figure 6b, where for three of the five different initialisations the solution does not converge to the global optimum (represented by the blue line).

We can observe some differences in the results for the different initialisations in Table 2. The initialisation with the least blocks, *init 1*, unsurprisingly results in the worst starting solution because of a lack of flexibility in the charging solution for each interval. We see in Figure 6a that on average the convergence for this initialisation is a much slower than for the other initialisations. However, on the other hand we can see in Figure 6b that it does not necessarily result in the worst final solution, where in this case it converges to the global optimum while three of the other initialisations do not. A possible explanation for this behaviour is that with this initialisation there is more opportunity to apply the split operator in an iteration and thus select the most promising split. If the algorithm initially starts with a candidate configuration where there are many relatively small blocks, it is harder to find improvement, since at some point no more splitting is possible.



(a) *Average objective value of the candidate solution in each iteration of local search. Average taken over 14 test realistic test instances.*

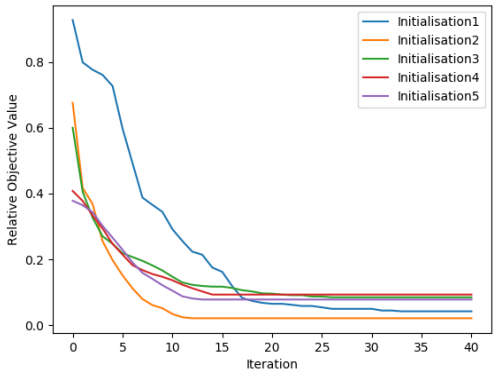
(b) *Objective value of the candidate solution in each iteration of local search, for one test instance with.*

Figure 6: Objective value per iteration of the local search algorithm, with $N = 100$, $R = 15$, and $C = 10000$, for the five different initializations in Table 2.

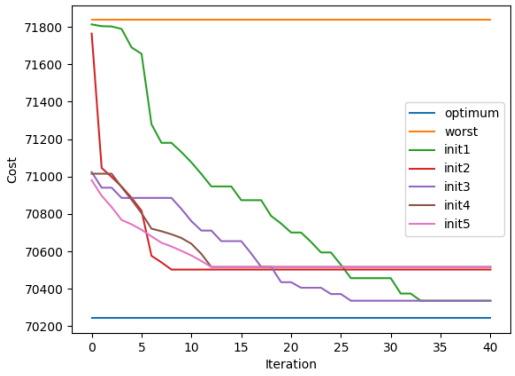
7.4.2 Local Minima

We saw in the results from the previous section that it is possible for the local search algorithm to get stuck in a local minimum and therefore never reach the global minimum. In this section we briefly look into what aspects of the structure of Problem *m1EVC* might cause more local minima to occur. Based on this we run some more simulations, this time with a 'difficult' baseload. With 'difficult' we mean that the instance has many local minima. A candidate solution is a local minimum when there does not exist any neighbouring configuration with a better objective value, even though within S (the set of all feasible configurations) there does exist another configuration with a better objective value. This better solution can never be reached by the algorithm once the local minimum is the current candidate. We designed a second set of problem instances that we expect will have more of these local minima, due to the structure of the baseload data. First of all we create clusters of low and of high values in this data, so-called valleys and peaks. However, unlike the realistic data we also add single-interval 'gaps' in these valleys and peaks: one interval with a high baseload in a valley, and one interval with a low baseload in a peak. This means the baseload data is less 'smooth', and that neighbours may be more likely to be worse than the current candidate solution even though there does exist a better configuration some neighbour steps away.

We ran the local search algorithm for these 'difficult' instances, using the same five different initialisations in Table 2. Some results are shown in Figure 7, where the average objective value is again shown for each iteration of the algorithm, as well as one particular test instance where we saw each of the five initializations converge to a local minimum. In Figure 7a we can see that on average the convergence to the global minimum is a more rare occurrence than for the simpler instances, and it can be observed that there are significantly more cases where the algorithm terminates in a local minimum.



(a) *Average objective value of the candidate solution in each iteration of local search with 'difficult' test instances. Average taken over 10 test instances.*



(b) *Objective value of the candidate solution in each iteration of local search, for one 'difficult' test instance.*

Figure 7: Objective value per iteration of the local search algorithm for the 'difficult' instances for the five different initializations in Table 2, with $N = 100$, $R = 15$, and $C = 10000$.

We can also observe in Figure 7a that on average initialisation 1 and 2 eventually converge to a better solution than the three other initial configurations, which contain more blocks. This again suggests that even though an initial configuration with many smaller blocks may initially result in a better object of value, it can be more likely that the algorithm ends up in a local minimum this way.

Until now we evaluated the local search algorithm using the same set of five initial configurations. In the next section we briefly look into a randomized initialization.

7.4.3 Random Initialization

In this section we use a randomized initialization to perform multiple runs of the local search algorithm for one problem instance, where we can select the best solution. A new initial configuration is randomly generated for each run. We generate these configurations by splitting the total number of intervals into two blocks, with the length of the first block chosen uniformly at random (but feasible, so no smaller than R), and repeating this process recurrently repeated for these two blocks, until finally no block of intervals can be split further without losing feasibility. This means that if $R = 15$, as in our test case, every block in a randomized initialization will have a length somewhere between 15 and 29. We discuss the results of this approach here.

We consider a test instance from Section 7.4.1 where we observed the occurrence of local minima, and we run 20 randomised runs of the local search algorithm. The resulting best objective value of each run is shown in Figure 8, as well as the best objective value of all runs up to and including the current run. We see that while the initial runs converge to a local minimum, eventually the global optimum is reached in the 17th run.

We have now discussed some simulation results with regards to the solution quality of the local search algorithm. In the next two sections we discuss some simulation results with regards to Result 6.2, and we compare the midpoint waterfilling approach to the traditional approach when calculating the gain of a neighbour.

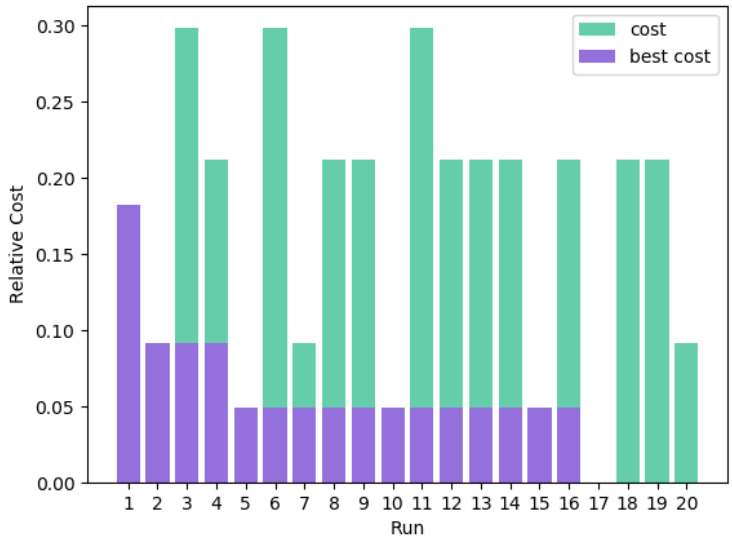


Figure 8: *Relative cost of the final charging solution calculated by the local search algorithm for 20 runs with randomized initialization, as well as the best relative cost up to and including the current run.*

7.4.4 Equality of Fill Levels

Result 6.2 suggests that the λ value of the current candidate solution and the λ value of a neighbouring solution are relatively likely to be equal. We investigate here whether this behaviour can be observed in practice. To this end we counted for a number of different test instances both the neighbours of a candidate configuration n for which $\lambda^n = \lambda^{n'}$ and the neighbours for which $\lambda^n \neq \lambda^{n'}$, over all iterations of the local search algorithm. We discuss the results here.

For the first simulations we used the realistic household data as described in Section 7.1, with $N = 100$, a minimum run-time $R = 10$, and a required charge $C = 10000$. We always used the initial configuration $(25, 25, 25, 25)$. It turns out that in these circumstances it was always the case that $\lambda^n = \lambda^{n'}$. This may seem strange, but there is a logical explanation using Result 6.1. This result states that if it is the case that all intervals are active in the optimal solution of two different configurations, then the λ values are equal. For these test instances it is very likely that all intervals have a charge that is somewhere between the minimum and maximum in the optimal solution of any configuration, since the required charge is not too high or too low. Thus we see that the requirements of Result 6.1 do occur frequently in practice.

From this explanation it follows that the λ values of different configurations should be unequal more frequently when the required charge C is very high or very low, since in this case the state of intervals (*inactive, active, or full*) will more often differ in the optimal solution of different configurations. To test this hypothesis we use the same datasets again but now with $C = 12300$, and indeed we see an increase in the number of neighbouring configurations where the λ value is different, as can be seen in Figure 9, where we show the total count of both equality and unequality of the λ values of the candidate and a neighbour. We expect the same trend to occur for a baseload dataset with extremely diverging peaks and valleys, by which we mean that there is a great difference between the high peaks and low valleys. We tested this hypothesis using the randomly generated datasets where there are valleys of baseload value 0, and peaks where the baseload value is somewhere between 5000 and 10000.

We set C to 10000 again. In these simulations we again see the expected increase in the number of neighbouring configurations where the λ value is different as can be seen in the second to last column of Figure 9.

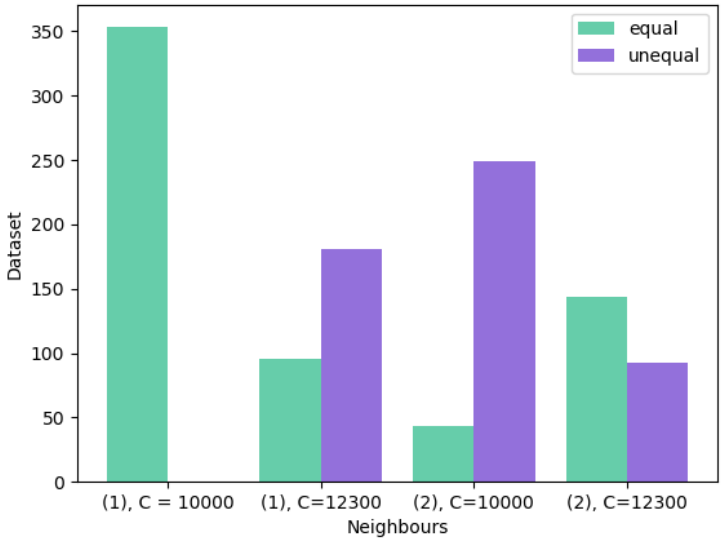


Figure 9: *The average count of neighbours n' for which $\lambda^n = \lambda^{n'}$, and the average count of neighbours n' for which $\lambda^n \neq \lambda^{n'}$, for a candidate configuration n in an iteration of local search. Averages were taken over 10 test instances, always with $N = 100$ and $R = 10$, and the algorithm was run for 50 iterations each time. (1) indicates the realistic baseload datasets. (2) indicates the baseload datasets of extreme valleys and peaks.*

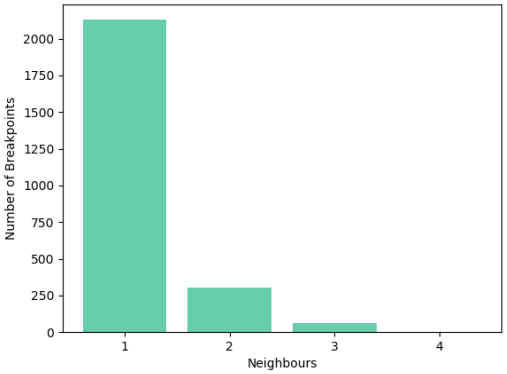
7.4.5 Midpoint Waterfilling

For the local search algorithm we applied an alternative approach to the waterfilling optimization step when calculating the gain, as was discussed in more detail in Section 6.3.1. Namely, we claimed that the values of λ would in general lie in close proximity to each other for different configurations. In this section we will present some results to substantiate this claim, and so support the choice for the alternative *midpoint* waterfilling algorithm.

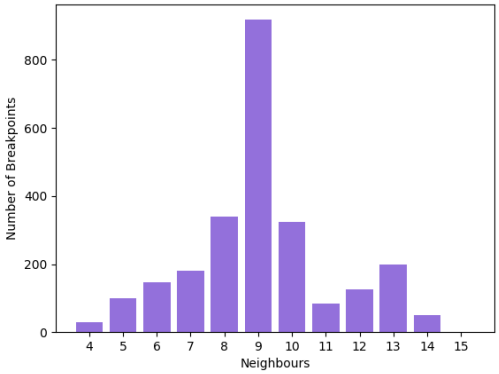
We use the datasets of extreme peaks and valleys and a required charge $C = 10000$, because we observed in the last section that in these circumstances the λ value of neighbouring configurations is frequently unequal to that of the current candidate, and therefore we can better compare the performance of the midpoint waterfilling approach to the traditional waterfilling approach (the algorithm as it is described in Section 3.2.2).

We compare the two algorithms by the number of steps the waterfilling algorithm takes to calculate the new λ value. This number of steps is determined by the number of breakpoints that the guess $\hat{\lambda}$ passes (see Section 3.2.2 for more details). We count the number of breakpoints that are passed for each neighbour in each iteration of local search. The results counted over a total of 10 test instances are shown in Figure 10. In this figure the number of breakpoints in the waterfilling algorithm is counted for each neighbour, both using the midpoint and using the traditional waterfilling algorithm. We see in Figure 10a that using the midpoint algorithm there is for most neighbours only one breakpoint step when calculating the new λ value, with at most three or four breakpoints in some rare cases. On the

other hand Figure 10b shows that in the traditional waterfilling algorithm there are more breakpoints before the new λ value is reached, where 9 breakpoints is most common. This shows that the midpoint waterfilling approach we proposed is indeed more efficient in calculating the gain of each neighbour.



(a) *Amount of neighbours counted in local search for which a certain number of breakpoints was passed in the midpoint waterfilling algorithm, over a total of 10 test instances.*



(b) *Amount of neighbours counted in local search for which a certain number of breakpoints was passed in the traditional waterfilling algorithm, over a total of 10 test instances.*

Figure 10: Comparison of the number of breakpoints when applying the midpoint waterfilling algorithm and when applying the standard waterfilling algorithm. The x-axis shows the number of breakpoints, and the y-axis shows the amount of neighbours counted.

7.4.6 Gain Estimation

In Section 6.3.2 we discussed a strategy to estimate the gain, instead of using the exact method of the waterfilling algorithm for each neighbour. In this section we apply this estimation strategy and compare it to the exact method.

We would expect the estimation to generally perform worse than the exact method, however, in Figure 11 the average objective value for both methods is plotted for each iteration of the local search algorithm, using the same test instances. It can be seen that on average both methods are comparable in their performance, where for some test instances the exact method performs better and for some test instances the estimation method performs better. This unexpected behaviour can partially be explained using the observations from the previous section. Here we saw that for a significant number of neighbours the midpoint of waterfilling algorithm only requires one step to calculate the gain. In this case the estimation strategy we implemented actually calculates the same charging solution as the waterfilling algorithm by adding the total remaining charge to the active intervals. This can partially explain the good performance of the estimation in this setting.

In terms of speed we see that the estimation strategy does make the local search algorithm faster in most test cases. This can be seen for 10 test instances in table 3. The table also gives the number times to gain was calculated/estimated in each test. We see that indeed the estimation strategy allows the local search algorithm to do the same number of calculations of the gain in a shorter amount of time, though the difference is not spectacular (this can again be explained by the little difference there is between the midpoint waterfilling method and the estimation in the number of steps).

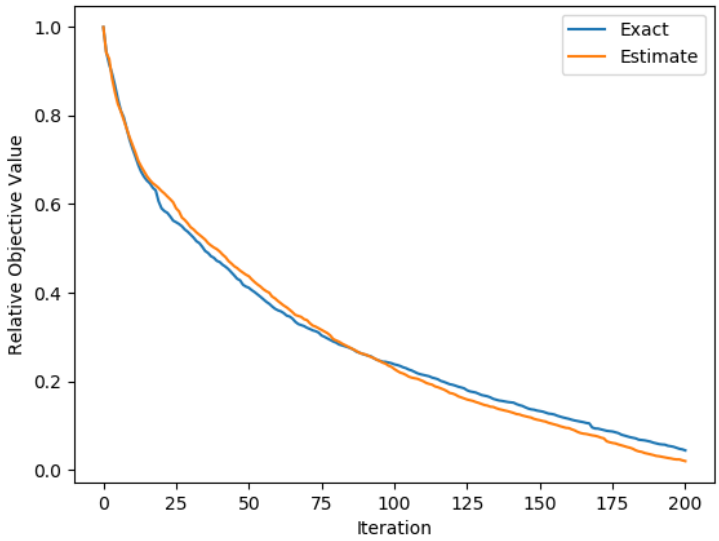


Figure 11: *Relative average objective value of the candidate solution in each iteration of local search. Average taken over 10 difficult test instances with* $N = 1440$*, using a required charge* $C = 175000$*.*

Dataset	Exact time (s)	# of calculations	Estimation time (s)	# of estimations
1	42.84	15055	35.46	15425
2	46.68	16344	40.92	16553
3	28.91	9480	28.94	11532
4	59.50	20680	59.14	22166
5	32.62	11203	41.91	17226
6	45.04	15543	39.67	16441
7	47.21	15180	46.72	15545
8	54.90	17701	48.97	16214
9	72.64	22634	80.22	29694
10	48.49	14466	32.29	11062

Table 3: The run-time of the local search algorithm both when the gain is calculated exactly, and when the gain is estimated. The number of calculations/estimations of the gain is also shown for each test instance.

7.4.7 Simulated Annealing

A specific local search methods that is often applied to mitigate the problem of local minima is simulated annealing [Eglese, 1990]. In this implementation of local search there is a possibility to jump to a neighbouring state even if the gain is negative, i.e. the objective value of this state is worse than the current candidate. We briefly discuss the results from our implementation here.

In our simulated annealing implementation we select one random neighbour in each iteration. We always accept this neighbour if the gain is positive. However, we also define p_A, which is the chance that the neighbour is accepted even though there is a non-positive gain. p_A should decrease both with a larger negative gain and with a higher iteration of the algorithm. The latter is to ensure there is still convergence to one solution.

Let n denote the current candidate configuration, and n' a neighbour with a non-positive gain. We define p_A as:

$$p_A = e^{-\frac{f(x^{n'}) - f(x^n)}{T_i}}.$$

T_1 is set to be equal to 1, and $T_5$00 equal to 0.005. We update T_i in each iteration i in the following way:

$$T_i = \alpha T_{i-1},$$

where $\alpha \approx 0.9895$ for the parameters in these test runs, to ensure $T_1 = 1$, and $T_5 00 = 0.005$. We use the same difficult datasets as in Section 7.4.2. The average objective value for each iteration of the algorithm is shown in Figure 12. We see the behaviour that we expected, namely fluctuations in the cost that become less and less, while the objective value of the candidate solution decreases. However we do not see a clear improvement to the other local search implementation. In fact we see more instances where the solution converges to a local minimum in the end, though we cannot explain this behaviour. Unfortunately due to time constraints there was no opportunity within the scope of this research to develop the simulated annealing approach further beyond this first implementation.

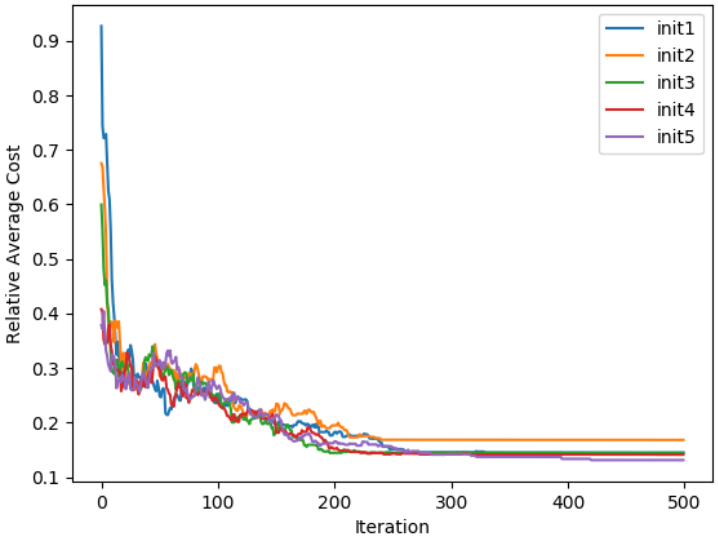


Figure 12: *Average objective value of the candidate solution in each iteration of simulated annealing, for the five different initializations and* $N = 100$, $R = 15$, *and* $C = 10000$. *Average taken over 10 difficult test instances.*

8 Conclusions and Recommendations

8.1 Conclusion

In this research we investigated the possibilities for a minimum run-time constraint in a single-device EV charging model, with the goal to flatten the load on the energy grid. Such a minimum run-time constraint can prevent unnecessary wear on the battery of the device, because it prevents the frequent switching of the operational level of the battery. We specifically focused on the single device models presented in [van der Klauw, 2017] within the Profile Steering framework, a specific demand-side management approach. To this end we first presented possible extensions to these single-device models, both for discrete and for continuous charging variables. We selected two of these extensions, *dm1EVC* as described in (6) and *m1EVC* as described in (8), to be the focus of this thesis. For these two problems we each presented an exact solution method, and we presented an approximation method for Problem *m1EVC*.

For Problem (6) we developed a dynamic programming algorithm that was based on an algorithm presented in [Gerards and Hurink, 2016], where a minimum run-time constraint was imposed on specifically an on/off single-device model. Our algorithm calculates an optimal solution to Problem (6) in pseudo-polynomial time, with a time complexity of $O(N^2M^2z^M)$, where N denotes the number of intervals, M the number of operational levels, and z^M the maximum charging rate of the device. Simulation results also showed that when increasing the minimum run-time R the calculation time decreases.

For Problem *m1EVC* we defined the concept of a block and of a configuration, to represent the minimum run-time constraint in a feasible solution, where a configuration codifies which sets of consecutive intervals charge at the same rate in this solution. We proved that given a configuration we can efficiently calculate an optimal charging solution. Using this result we developed a first algorithm for Problem *m1EVC*, which enumerates all potentially optimal configurations and calculates the best solution for each one. This algorithm is slow since a large number of configurations must be enumerated. Also its memory requirements are relatively high. Both of these properties make it not very suitable for the Profile Steering framework.

Secondly, we developed an alternative approach, using the same concept of configurations and their associated optimal charging solution. This second algorithm is a local search approach, which approximates the optimal solution of a problem instance by iteratively searching the neighbourhood of a certain candidate configuration. We defined neighbourhood operations based on the specific structure of the problem. In simulations the algorithm showed a fast performance, also with larger instances, as opposed to the exact approach. We see that the neighbourhood structure was well-chosen, as neighbours are close-by in their charging solution, and therefore a local search approach lends itself well to this problem. We do observe that there are local minima that sometimes prevent the algorithm from reaching the optimal solution. We also see that the choice of initial configuration can influence both the speed of convergence and whether or not the algorithm converges to a local minimum. When using a randomized initialisation we see that multiple runs of the local search algorithm can successfully solve the problem of local minima, also for the more difficult instances.

We developed two different approaches to calculate the gain of each neighbour of a candidate solution in the local search algorithm. The first approach uses a known exact method to calculate the optimal charging solution for each configuration, an approach named the waterfilling algorithm. The second approach is an estimation, which uses the set of active blocks in an initial guess of the charging solution, and then increases or decreases the charge of these blocks to reach the required charge. In simulations we see that the performance of the two methods is comparable in terms of solution quality. Estimating the gain results in a slight improvement in the speed of the algorithm compared to the exact method.

To calculate the optimal charging solution for a configuration we use the waterfilling algorithm, which is described in Section 3.2.2. This algorithm exploits the fact that the optimal charging solution for a given configuration can be characterized by a single value, namely the so-called optimal fill level. In our local search approach we applied a slightly different version of the waterfilling algorithm, where the fill level of the current candidate configuration is used as the starting point to calculate the optimal fill level for each neighbour. This adjustment is motivated by the expectation that these values are generally close for neighbours. Simulation results show that this is indeed the case when we compare this method to the traditional waterfilling algorithm. We also proved that there are circumstances in which two configurations actually have an equal optimal fill level, in Result 6.2. When this is the case we do not have to apply the water filling algorithm to calculate the gain and thereby we save calculation time. In simulations we observed that this property is indeed likely to occur and therefore using this property is a significant improvement in the performance of the local search algorithm in terms of calculation speed.

Finally, we implemented a second version of the local search algorithm, based on the concept of simulated annealing. In simulated annealing there is a possibility to jump to a neighbouring state even if the gain of this state is negative. We see that our simple simulated annealing implementation does not yet for this problem mitigate the issue that local minima pose.

8.2 Discussion

In this section we discuss some of the limitations of this research as well as give several recommendations for potential future research directions.

In In our first algorithm we enumerate the set of configurations we use to solve Problem *m1EVC* to optimality. Because this set grows large fast with an increasing number of intervals N, the algorithm is not scalable to larger instances. Secondly, the way in which the configurations are enumerated now requires a lot of memory. Both of these limitations of the algorithm could potentially be improved upon. In particular, there may be ways to limit the set of considered configurations further. Related to this is also the question how the size of the set of configurations may be expressed in terms of N and R, and whether this set grows exponentially with N.

Secondly, the initial model for Problem *m1EVC* presented in (8) defines both a minimum positive run-time and a minimum idle time, denoted by R_1 and R_0 respectively. We assumed for this research that $R_1 = R_0$. An interesting follow-up question is how much more difficult the problem becomes when this assumption is removed. We can already observe that the definition of a configuration as used in this work is not directly applicable to this more general case, because there is no longer one single minimum run-time R.

In a similar vein we have in this research considered a specific objective for the optimisation, namely the objective of flattening the load on the energy grid using the cost function $f(x) = \sum_t (x_t + p_t)^2$. Some of our results are no longer true when different objective functions are considered, for example the earlier mentioned function f where each term has its own weight coefficient. Future research could therefore look into solution methods for the *m1EVC* model with a quadratic or convex cost function, and whether any extension of the methods presented here is still possible in that case.

It remains an open question whether Problem *m1EVC* is NP-hard or not. We have neither managed to prove that it is within the scope of this research, nor have we been able to develop an optimal polynomial-time approach. We do deem it relatively likely that the problem is NP-hard, due to the difficulties we have encountered within this research in the characterization of an optimal solution.

Section 5.4 discusses Problem *m1BC*, which we described in (9). This problem considers the charging

of a bidirectional battery, which results in cumulative constraints that are not present in Problem *m1EVC*. We concluded that our approach to Problem *m1EVC* does not extend to solve this problem as well. But perhaps a similar solution method is possible, where the structure of the minimum run-time constraint is used similarly to what is done in this research for Problem *m1EVC*.

We defined four neighbourhood operations for the local search algorithm. One of these operations, the *split* operator, currently defines a much smaller set of neighbours than the other three operators when it is applied in an iteration. This is because the split operator only splits each of the largest blocks in a configuration to define a neighbour. In the worst-case this results in only one neighbour of the current candidate solution. On the other hand the other three operators all consider every feasible pair of blocks in the candidate solution to define a neighbour, naturally resulting in a larger pool of neighbours to consider. This imbalance could be improved upon by defining the split operator in a different manner.

We presented a simple simulated annealing approach in this research. One easy method to make such an approach more sophisticated and effective would be 'restarts', where the result of one simulated annealing run is used as the starting configuration for another run. Alternitively the state with the best objective can be stored for some iterations, and can be selected again if no better candidate is found in the meantime.

We note that another phenomenon that can cause faster battery wear in the charging of an EV is fast charging (charging at high charging rates), mentioned for example in [Trippe et al., 2014]. There may be possibilities to extend the models presented in this research to also limits charging at these extremely high rates and thus reduce battery ageing further.

Finally, we conclude this section by noting that an interesting follow-up question is how the solution quality of this newly defined problems relate to the old problem, *EVC*, and how much the difference in quality may depend on certain behaviour of the problem instance.

References

[Aalst, 2006] Aalst, M. K. V. (2006). The impacts of climate change on the risk of natural disasters. *Disasters*, 30(1):5–18.

[Barbato and Capone, 2014] Barbato, A. and Capone, A. (2014). Optimization models and methods for demand-side management of residential users: A survey. *Energies*.

[Bienstock and Mattia, 2007] Bienstock, D. and Mattia, S. (2007). Using mixed-integer programming to solve power grid blackout problems. *Discrete Optimization*, 4(1):115–141.

[Carrion and Arroyo, 2006] Carrion, M. and Arroyo, J. M. (2006). A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 21(3):1371–1378.

[Cenedese et al., 2019] Cenedese, C., Fabiani, F., M. Cucuzzella, J. M. A. S., Cao, M., and Grammatico, S. (2019). Charging plug-in electric vehicles as a mixed-integer aggregative game. In *2019 IEEE 58th Conference on Decision and Control (CDC)*.

[Chen et al., 2014] Chen, N., Tan, C. W., and Quek, T. Q. S. (2014). Electric vehicle charging in smart grid: Optimality and valley-filling algorithms. *IEEE Journal of Selected Topics in Signal Processing*, 8(6).

[Eglese, 1990] Eglese, R. (1990). Simulated annealing: a tool for operational research. *European Journal of Operational Research*, 46:271–281.

[Esther and Kumar, 2016] Esther, B. P. and Kumar, K. S. (2016). A survey on residential demand side management architecture, approaches, optimization models and methods. *Renewable and Sustainable Energy Reviews*, 59:342–351.

[Falkner, 2016] Falkner, R. (2016). The paris agreement and the new logic of international climate politics. *International Affairs*, 92(5):1107–1125.

[Gerards and Hurink, 2016] Gerards, M. E. T. and Hurink, J. L. (2016). Planning of on/off devices with minimum run-times. In *Proceedings of the 2016 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, pages 1–6. IEEE Power Electronics Society.

[Hemavathi and Shinisha, 2022] Hemavathi, S. and Shinisha, A. (2022). A study on trends and developments in electric vehicle charging technologies. *Journal of Energy Storage*, 52.

[Hoogsteen et al., 2017] Hoogsteen, G., Molderink, A., Hurink, J., Smit, G., Kootstra, B., and Schuring, F. (2017). Charging electric vehicles, baking pizzas, and melting a fuse in lochem. *CIRED - Open Access Proceedings Journal*, 2017:1629–1633.

[Hoogsteen et al., 2016] Hoogsteen, G., Molderink, A., Hurink, J. L., and Smit, G. J. M. (2016). Generation of flexible domestic load profiles to evaluate demand side management approaches. In *2016 IEEE International Energy Conference (ENERGYCON)*, pages 1–6. IEEE Power & Energy Society.

[IEA, 2022] IEA (2022). Global ev outlook 2022. *Technical report, IEA*.

[Jebaraja and Iniyan, 2006] Jebaraja, S. and Iniyan, S. (2006). A review of energy models. *Renewable and Sustainable Energy Reviews*, 10:281–311.

[Khalid et al., 2022] Khalid, M., Ahmad, F., Panigrahi, B. K., and Al-Fagih, L. (2022). A comprehensive review on advanced charging topologies and methodologies for electric vehicle battery. *Journal of Energy Storage*, 53:105084.

[Kuyper et al., 2018] Kuyper, J., Schroeder, H., and Linnér, B.-O. (2018). The evolution of the unfccc. *Annual Review of Environment and Resources*, 43:343–368.

[Mohanty et al., 2022] Mohanty, S., Panda, S., Parida, S. M., Rout, P. K., Sahu, B. K., Bajaj, M., Zawbaa, H. M., Kumar, N. M., and Kamel, S. (2022). Demand side management of electric vehicles in smart grids: A survey on strategies, challenges, modeling, and optimization. *Energy Reports*, 8:12466–12490.

[Nykamp et al., 2015] Nykamp, S., Rott, T., Dettke, N., and Kueppers, S. (2015). The project "elche" wettringen: storage as an alternative to grid reinforcements - experiences, benefits and challenges from a dso point of. In *International ETG Congress 2015; Die Energiewende - Blueprints for the new energy age*, pages 1–6.

[Patriksson, 2008] Patriksson, M. (2008). A survey on the continuous nonlinear resource allocation problem. *European Journal of Operational Research*, 185:1–46.

[Patriksson and Strömberg, 2015] Patriksson, M. and Strömberg, C. (2015). Algorithms for the continuous nonlinear resource allocation problem–new implementations and numerical studies. *European Journal of Operational Research*, 243:703–722.

[Pörtner et al., 2022] Pörtner, H.-O., Roberts, D. C., Adams, H., Adler, C., Aldunce, P., Ali, E., Begum, R. A., Betts, R., Kerr, R. B., and Biesbroek, R. (2022). Climate change 2022: Impacts, adaptation and vulnerability. *IPCC Sixth Assessment Report*.

[Rachid et al., 2023] Rachid, A., El Fadil, H., Gaouzi, K., Rachid, K., Lassioui, A., El Idrissi, Z., and Koundi, M. (2023). Electric vehicle charging systems: Comprehensive review. *Energies*, 16(1).

[Schoot Uiterkamp et al., 2018] Schoot Uiterkamp, M. H. H., van der Klauw, T., Gerards, M. E. T., and Hurink, J. L. (2018). Offline and online scheduling of electric vehicle charging with a minimum charging threshold. In *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 1–6.

[Siano, 2014] Siano, P. (2014). Demand response and smart grids—a survey. *Renewable and Sustainable Energy Reviews*, 30:461–478.

[Strbac, 2008] Strbac, G. (2008). Demand side management: Benefits and challenges. *Energy Policy*, 36(12):4419–4426.

[Trippe et al., 2014] Trippe, A. E., Arunachala, R., Massier, T., Jossen, A., and Hamacher, T. (2014). Charging optimization of battery electric vehicles including cycle battery aging. In *IEEE PES Innovative Smart Grid Technologies, Europe*, pages 1–6.

[van der Klauw, 2017] van der Klauw, T. (2017). *Decentralized Energy Management with Profile Steering: Resource Allocation Problems in Energy Management*. University of Twente.

[Winston, 2003] Winston, W. L. (2003). *Introduction to Mathematical Programming: Applications and Algorithms*. Duxbury Resource Center.

[Young et al., 2013] Young, K., Wang, C., Wang, L. Y., and Strunz, K. (2013). Electric vehicle battery technologies. *Electric vehicle integration into modern power networks*, pages 15–56.