FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Electric Vehicles Recharge Scheduling with Time Windows

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Damir Bučar
Matrikelnummer 1027323

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret Musliu
Mitbetreuung: Priv.-Doz. Dr. Sandford Bessler

Wien, 01.09.2014.

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Electric Vehicles Recharge Scheduling with Time Windows

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computational Intelligence

by

Damir Bučar
Registration Number 1027323

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dr. Nysret Musliu
Co-advisor: Priv.-Doz. Dr. Sandford Bessler

Vienna, 01.09.2014.

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Damir Bučar
Matetić-Ronjgovljeva 12, 10000 Zagreb, Kroatien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

To my nearest and dearest ones...

Abstract

The problem of scheduling the recharge operations of electric vehicles (EVs) is one we will inevitably face in the near future. An important aspect to consider is the impact that increasing number of EVs, with their frequent need for recharge, would have on the current electric grid infrastructure and how we can avoid overloading the grid at certain periods of the day.

The EV recharge scheduling task is a part of the system developed within the FFG[1]-funded KOFLA project, which has the goal of helping EV users to find the best opportunity for recharging (nearby, cheap and fast).

The optimization problem identified in this work as *Electric Vehicles Recharge Scheduling with Time Windows* (EVRSTW) is closely related to the resource allocation and resource-constrained scheduling problems. EVRSTW, in its special case, belongs to the class of NP-complete problems, meaning that exact methods are usually unable to cope with large problem instances in reasonable time, and display unpredictable runtimes. In order to build a real world dynamic system based on user requests and almost immediate system responses, we need to find methods which can operate within a short, bounded execution time.

This thesis investigates possible heuristic approaches to solving the EVRSTW problem. We first model a greedy heuristic, to simulate a "dumb", plugin-and-charge-immediately, approach. Since it performs poorly, we turn to the local ratio technique, tailored to our problem specification. This method still rejects some of the activities (EV recharge requests) which need to be scheduled, so we search for a new approach. Finally, we design a local search method based on the min-conflicts heuristic, along with defining an appropriate neighborhood relation.

We evaluate the three heuristic methods on a set of generic problem instances, according to multiple evaluation criteria, such as total profit, runtime, and number of unscheduled activities. The experiments show that only the MCH manages to generate feasible solutions, i.e. to schedule all of the activities. We analyze its performance on two different EV models and in two running modes of the system: a) offline, where all activities are presented at once and the schedule is built "from scratch"; and b) online, where activities arrive one by one and the schedule is built incrementally.

Comparing the min-conflicts heuristic against the exact method (CPLEX solver), we find that its runtimes are significantly shorter — in terms of less than 200 milliseconds versus 25+ minutes in the extreme case, with an acceptable solution quality of 75% to 93% of the optima. This makes the MCH a good candidate for application in the online environment, which we turn into reality by incorporating it within a charging station controller. We evaluate its performance

[1]The Austrian Research Promotion Agency (FFG), `https://www.ffg.at/en`

in this new setting and show that the parking period prediction error has a significant impact on the solution quality.

Kurzfassung

Die Ladeoptimierung Elektroautos wird in der nahen Zukunft ein immer wichtigeres Problem, wegen der hohen Energiemengen die in kürzester Zeit über das Stromnetz fliessen müssen. Ohne die Flexibilität bei der Aufladung zu nutzen besteht die Gefahr die vorhandene Netzinfrastruktur zu bestimmten Tageszeiten (Spitzenlast) zu überlasten.

Das in dieser Arbeit behandelte Ladeoptimierungsproblem ist ein Teil des Systems, das innerhalb des KOFLA-Projektes entwickelt und von FFG[2] finanziert wurde. Ziel des Projekts ist die Benutzer von Elektrofahrzeugen zu unterstützen, die besten öffentlichen Ladestationen zu finden (in der Nähe, günstig und schnell).

Das Optimierungsproblem, das in dieser Arbeit als *Electric Vehicles Recharge Scheduling with Time Windows* (EVRSTW) bezeichnet wird, ist eng mit den Problemen der Ablaufobtimierung (*scheduling*) mit Ressourcen-Restriktionen verbunden.

EVRSTW gehört der Klasse von NP-kompletten Problemen, was bedeutet, dass die exakten Methoden nicht in der Lage sind große Probleminstanzen in vernünftiger Zeit zu lösen. In einer realen Umgebung, in der der Benutzer sofort wissen muß, ob seine Ladeanforderung akzeptiert wird, sollten Algorithmen mit begrenzter Laufzeit eine (auch suboptimale) Lösung liefern können.

Diese Diplomarbeit untersucht die möglichen heuristischen Ansätze für das EVRSTW–Problem. Zuerst definieren wir die *greedy* Heuristik, welche eine sofortige Aufladung nach plug-in bewirkt. Da diese Heuristik schlechte Ergebnisse nachweist widmen wir uns der *local ratio* Technik, welche der Natur unseres Problems angepasst wird. Da diese Methode immer noch viele Ladeanforderungen ablehnt, suchen wir nach weiteren Heuristiken. Basierend auf der lokalen Suche, wählen wir schließlich die *min-conflicts* Heuristik (MCH) und definieren eine passende Nachbarschaftsrelation zwischen Lösungen.

Wir vergleichen die drei heuristischen Methoden nach den vielfältigen Kriterien der Evaluierung, wie z. B. nach dem gesamten Profit, der Laufzeit und Anzahl der abgelehnten Aktivitäten. Die Experimente werden mit einem Satz von generischen Probleminstanzen geführt. Diese zeigen, dass nur die MCH Methode es schafft, zulässige Lösungen zu generieren, indem sie die sämtlichen Aktivitäten erfolgreich einplant. Wir analysieren ihre Performanz und Qualität der Lösungen sowohl in dem offline Modus, wenn alle Aktivitäten auf einmal bekannt werden, und in online Modus, wenn die Aktivitäten als Ereignisse einzeln eintreffen und der Ablaufplan in der Zeit gerollt und permanent aktualisiert wird.

[2]The Austrian Research Promotion Agency (FFG), `https://www.ffg.at/en`

Im Vergleich mit der exakten Methode (CPLEX Solver) zeigen wir, dass die Laufzeiten der MCH Methode bedeutsam kurzer sind —– weniger als 200 Millisekunden gegen 25+ Minuten im Extremfall, und das bei einer akzeptabler Lösungsqualität von 75% bis 93% vom Optimum. Dadurch ist die MCH Methode ein guter Kandidat für die Anwendung in Onlinemodus, und kann im Ladestation-Controller eingebaut werden. Schließlich untersuchen wir den Einfluss von Fehlern bei der Abschätzung der Parkdauer und zeigen dass diese die Lösungsqualität ziemlich verschlechtern.

Contents

CHAPTER 1

Introduction

Electric vehicles (EVs) appear to have a bright future, since they promise, among other benefits, greenhouse gas emission reduction and independence from fossil fuels. However, their range is still quite limited and recharge facilities quite scarce in comparison to those of conventional vehicles. Additionally, the battery charging process is significantly longer (up to several hours) than what it takes a conventional internal combustion engine (ICE) vehicle to refuel. Thus, an important aspect to consider is their efficient recharge under the constraints imposed by the electric grid.

The focus of this thesis will be on the *problem of controlled EV recharge scheduling* at public charging stations, where the energy demands of EVs (activities) need to be satisfied, considering the constraints of the number of charging points (machines) and available power. The work described here was conducted within the KOFLA project [14], [7] at Forschungszentrum Telekommunikation Wien (FTW), funded by the Austrian Research Promotion Agency (FFG) under the "ways2go" programme. Its goal is to model an intelligent online brokerage system between users that need a battery charging service and a network of public charging stations that can supply that service. The study conducted in this thesis does not go into detail about electric vehicles' batteries and their behavior from the chemistry or electronics points of view — the accent is put solely on the recharge scheduling problem from the perspective of computer science.

The core scenario can be described as follows: An EV charging station contains charging slots (*machines*) for several vehicles. Its owner maintains data about a collection of vehicles, already parked or on the way. Each vehicle is associated with a planned parking time window and its battery's energy demand that needs to be satisfied. We will denote this data entity as an *activity*. The available power constraint dictates that not all vehicles can be charged voluntarily, but their charging needs to be planned within the given time frame. The owner's goal is to generate a schedule which would maximize his profit function, while keeping the customers as satisfied as possible.

We denote this problem as ELECTRIC VEHICLE RECHARGE SCHEDULING WITH TIME WINDOWS (EVRSTW) [7]. It is closely related with the *resource allocation problem* (RAP)

[2], [3], [10], [12], which is often regarded as *bandwidth allocation problem* (BAP); but also with *interval scheduling* [19] and *resource-constrained scheduling* [24] problems.

An additional requirement KOFLA framework has imposed is to have the scheduling algorithm perform in a dynamic, *online* manner, i.e. in real time, which requires maintenance of some sort of a rolling schedule, as well as swift performance (in terms of seconds, or even milliseconds).

Multiple EVRSTW problem parameters to consider: the planning time horizon, available power constraint imposed by the electric network, time windows defined by the parking duration of the vehicles, multi-valued energy demands to fulfill, different charging rates etc. make the decision variant of the problem NP-complete (it contains as a special case an instance of the *0–1 knapsack* problem, which is NP-complete) [7], [12]. Therefore, exact methods (e.g. CPLEX solver on MIP [7], [7]) are usually unable to find solutions to problem instances of increasing size in reasonable time. Using the CPLEX solver also entails *unpredictable runtimes* — for the same problem size the execution can take from a few seconds to tens of minutes [7]. In order to build a real-world dynamic system based on requests and almost immediate decisions, we need to employ methods running within a *short, bounded response time*.

According to [20], smart charging techniques would allow for increase of EV penetration without investments into current infrastructure. IHS [26] believes that fast and efficient charging is a necessary step to promote higher adoption of EVs, but there will need to also be better consumer education regarding behavioral changes that may need to happen when owning an electric vehicle — such as charging overnight or at work. It is therefore worth exploring this issue here, as it could be of great importance in the near future.

1.1 Objectives

The objectives of this thesis are:

- Exploring the aspects of the novel EVRSTW problem and its subproblems. Define its model and constraints.
- Designing a scheduling heuristic which can provide feasible solutions of acceptable quality in short execution time (especially for larger problem instances, for which exact methods, such as CPLEX, run for too long).
- Experimental evaluation of our solution approaches, as opposed to the exact method, by the use of a generic set of problem instances.
- Enabling the real-world, online application of the scheduling algorithm, within the wider framework of the KOFLA routing, reservation and scheduling system.

1.2 Main Results

The main results obtained by this work are:

- A detailed description and analysis of the EVRSTW problem, and a study of related work. Models of the electric vehicle, the charging station and the charging schedule.

- Design and implementation of a *greedy heuristic* to simulate "dumb" charging (but keeping the available power constraint satisfied). Implementation of the *local ratio technique* according to guidelines from [2], [3], tailored to our specific resource-constrained scheduling problem.
- Design and implementation of a new local search method based on the *min-conflicts heuristic*, which is expected to perform well on larger problem instances in reasonable time. In order to apply it successfully, we propose an appropriate neighborhood relation and investigate the impact of random noise.
- Performance evaluation of the three solution approaches — the greedy heuristic, the local ratio technique, and the min-conflicts heuristic — in a steady-state, offline mode, compared to the optimal results of the exact CPLEX solver [7], [9], on a generic set of problem instances. Successful application of the *min-conflicts heuristic* to the EVRSTW problem, which reaches up to 93% of the optimal solution profit (offline mode). Our heuristic managed to solve problem instances up to the size of 200 (not tested further) activities in less than 200ms.
- Design and implementation of a charging station controller, enabling the MCH to perform in an online, dynamic mode, within the KOFLA routing, reservation and recharge scheduling system. We identified the main issues and dealt with them successfully. We also discovered that the parking period prediction error has the greatest impact on the solution quality.

Parts of this work were presented in a MISTA conference paper [9].

1.3 Organization

Chapter 2 gives the EVRSTW problem formulation and describes its environment within the KOFLA system. This is followed by an overview of literature related to the problem itself, but also to a bit broader EV environment, including a case study on a real world Belgian EV charging system.

In Chapter 3 multiple solution approaches are proposed, describing in detail the simple greedy heuristic, the local ratio technique, and the min-conflicts heuristic, all tailored to the model of the problem at hand. Additionally, adaptations necessary to perform the transition into a dynamic environment are described.

Chapter 4 offers performance evaluation of the developed solution approaches, accompanied by a brief insight into the system's online behavior.

Finally, Chapter 5 contains the summary and discussion about possible future work.

CHAPTER 2

The EVRSTW Problem

In this chapter we describe the problem of the *Electric Vehicles Recharge Scheduling with Time Windows* (EVRSTW) and provide its formal definition.

We define the models of the electric vehicle, of the charging station with its machines and of the charging schedule. Then we discuss the evaluation (profit) function and show the formulation of the problem as a mixed integer program (MIP), based on [7] and [9].

In the final section we provide a thorough overview of work related to the EVRSTW problem and its wider environment.

2.1 Problem Statement

We first give a brief overview of the problem's broader environment within the KOFLA framework, and then continue to describe EVRSTW problem from a practical point of view. The formal, mathematical model of the problem is given in Section 2.2.

2.1.1 KOFLA Framework

Project KOFLA, carried out at Forschungszentrum Telekommunikation Wien, deals with the issues related to the new concept of personal mobility using electric vehicles (EV). The project, finalized in 2012, was funded by the Austrian Research Promotion Agency (FFG) under the "ways2go" programme.

Since recharging the EV battery can take up to several hours, and potential users still suffer from range anxiety [1], the EV adoption has not yet reached significant levels. Multiple potential options for recharge locations — parking lots, gas stations, shopping malls etc. — would quickly become unmanageable if handled independently. EV users would therefore need assistance in order to find the "best" charging station, and some kind of a centralized infrastructure management system would certainly allow for better utilization of the electric grid.

[1] *Range anxiety* is the fear that a vehicle has insufficient range to reach its destination and would thus strand the vehicle's occupants. [13]

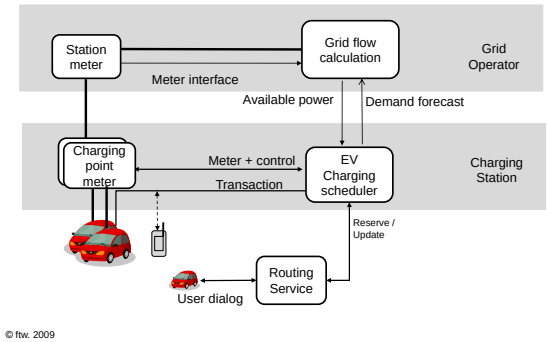
The main goal of the KOFLA project was to model an intelligent online brokerage system between users that need a battery charging service and a network of public charging stations that can supply that service. Not only do the users' energy demands need to be satisfied, but also the electric grid limitations have to be taken into account.

The three main components of the KOFLA framework are:

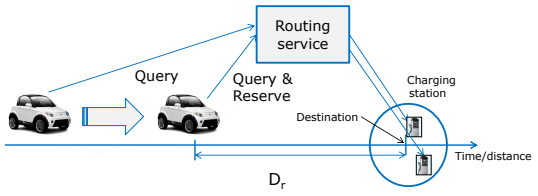
- A (multimodal) routing and reservation system, which interacts with the users and the grid, balances the energy load flow between multiple charging stations and directs the users towards the "best" of them.
- An online charging scheduler, employed at each of the charging stations, dealing with the reservation requests and employing optimization algorithms for the charging schedule generation, while taking the grid constraints into consideration.
- An e-mobility simulator, integrating the above.

The focus of this thesis is on the second point — we will deal with controlled charging schedule optimization at individual public charging stations.

Figure 2.1 depicts the high-level architecture of the KOFLA framework, alongside a typical reservation and routing scenario within the system [14], [7].



(a) KOFLA high-level architecture



(b) KOFLA reservation and routing scenario

Figure 2.1: KOFLA high-level architecture and reservation and routing scenario

2.1.2 The EVRSTW Problem

The problem considered in this thesis, *Electric Vehicle Recharge Scheduling with Time Windows* (EVRSTW), has first been proposed as an essential part of the KOFLA project [14], [7].

Looking from the application point of view, the problem of scheduling the EV charging can be described as follows: An EV charging station (CS) contains charging points (*machines*) to accommodate several vehicles. The service provider maintains data about a collection of vehicles, already parked or on the way. Each vehicle is associated with a planned parking period and its battery's energy demand that needs to be satisfied. This EV data entity will further be denoted as an *activity*. The fixed number of machines limits the number of vehicles which can be

concurrently parked at the CS. The available power constraint dictates that not all vehicles can be charged voluntarily, but their charging needs to be planned within the planning time horizon. The service provider's goal is to generate a schedule which would maximize his profit function, while at the same time serving as many users as possible, and optimally utilizing the available power throughout the day, without causing congestion. Customers should be served fair and get their vehicles' energy demands satisfied as much as possible.

The machines are able to operate at different charging rates, i.e. they can deliver less or more power into batteries, leading to longer or shorter charging durations, respectively. Once decided upon, the charging rates are constant over time — once the charging has started, it has to be completed at the determined rate. Another important note is that the scheduling of charging plans is *non-preemptive*, i.e. once the charging process has initiated, it can not be interrupted — the charging periods are necessarily contiguous.

Two resources are to be recognized here: the *parking resource* during the EV's parking period, and the *power resource* during its charging period. These immediately impose the constraints on the problem: we operate with a fixed number of *machines*, each being able to accommodate only one activity at the time; and with *available power* (fixed or variable over time) defined during the planning time horizon, which must not be exceeded at any point in time.

Under the assumption of identical machines (i.e. each one supporting all of the different charging rates), the EVRSTW problem can be split into *two subproblems* (which can be solved separately), according to the two resources:

1. **MACHINE ALLOCATION** — Assignment of the activities (EVs) to one of multiple machines (charging points). This problem is analogous to the one of *Interval Scheduling* [19] on multiple machines. Intuitively, at most one activity can be scheduled on a single machine at any given time. Intervals are defined by the arrival-departure time window and can not be shifted. This problem can be solved *optimally* by a *greedy method*, assigning activities to machines in the increasing order of arrival time. Solving this subproblem successfully allows us to address the second one separately, effectively as a single machine problem.
2. **CHARGING PLANS ASSIGNMENT** — Distribution of the available power throughout the activities by selecting appropriate charging plans, i.e. scheduling the parked EVs' charging periods within their parking periods. Here we need to select the start time, rate of charging and completion degree, which together determine the charging duration and the profit. The goal is to maximize the total profit induced by all scheduled charging plan instances. This problem is similar to the *Resource Allocation Problem* [10], [12] with time windows, often regarded as *Bandwidth Allocation Problem*; and very close to the *Resource-Constrained Scheduling Problem* [24]. Because of the problem's NP-complete nature, we will need to employ heuristic methods to efficiently solve larger problem instances.

The problem split is visualized in Figure 2.2.

The case of non-identical machines renders the problem even more complex, when in the most general case the problem split is not possible, and charging plan instances have to be created for each individual machine. The rest of the described work assumes *identical machines*.

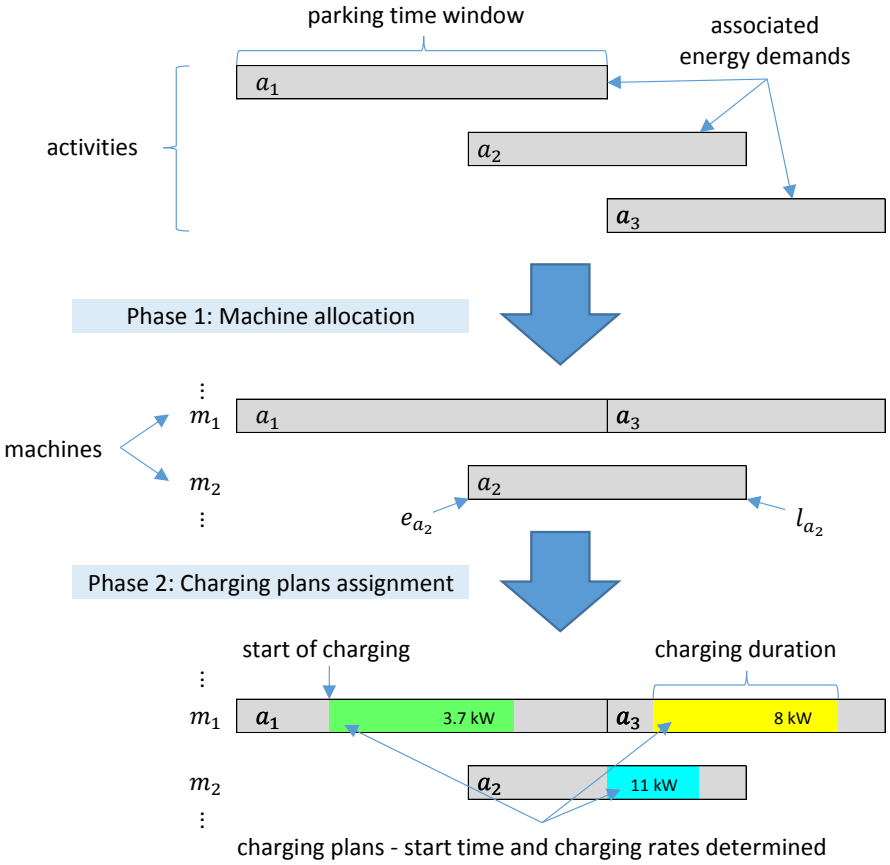


Figure 2.2: Two phases of solving the EVRSTW problem

Activities (parked EVs) define a fixed parking time window within which charging should take place. For each activity we can consider multiple charging plans, each determined by its charging rate, start time and the amount of energy demand it satisfies. Intuitively, only one charging plan can be assigned to each activity.

Several EVRSTW problem parameters to consider — multiple machines and activities, available power constraint imposed by the electric network, various energy demands to fulfill, different charging rates etc. — make the problem (more precise, the second subproblem, if we assume identical machines), in its decision form, NP-complete. It contains as a special case an instance of the *0-1 knapsack* problem [7], [12], which is known to be NP-complete. If we let the parking period intervals to be identical for all activities, and assume different profits of their charging plans, we can recognize the classical 0–1 knapsack problem, of choosing the set of charging plan instances of greatest profit, given the ”capacity” in terms of available power constraint. One must take care of covering all of the activities, and only one charging plan instance can be selected for each of them.

Thus, exact methods (e.g. CPLEX MIP solver) are unable to find solutions to larger problem instances in reasonable time. We will therefore attempt to apply heuristic strategies to attack the

problem, and obtain solutions that are "good enough" for our evaluation criteria.

2.2 Modeling the System

As we have mentioned earlier, the EVRSTW problem can be split (assuming identical machines) into two subproblems:

1. The *machine allocation* problem, of assigning activities ("jobs" with fixed start and end times) to multiple machines; analogous to *interval scheduling*
2. The *charging plans assignment* problem, of selecting a charging plan for each activity, with the goal of maximum total profit; analogous to problems of *resource constrained scheduling* (RCSP) and *resource allocation* (RAP) *with time windows*)

The first one can be efficiently (and optimally) solved in a preprocessing phase by the use of a greedy method, which will be presented in Section 3.1. The second subproblem will be solved by heuristic methods which we model in Section 3.3, with the use of the charging plan instance generation strategy designed in Section 3.2.

Table 2.1 provides an overview of the notation we will use.

Symbol	Meaning
$\mathcal{T}$	discretized time horizon with time slots $t \in \{0, 1, ..., T-1\}, T = \lvert\mathcal{T}\rvert$
$\mathcal{A}$	set of activities (electric vehicles)
$\mathcal{M}$	set of machines (charging points)
$\mathcal{R}$	set of charging rates machines can deliver (kW)
$\mathcal{C}$	set of possible completion degrees (used only in combination with d_a)
$\mathcal{I}$	charging plan instances generated from activities in $\mathcal{A}$
$\mathcal{I}_a$	charging plan instances generated from activity $a \in \mathcal{A}$
e_a, l_a	*earliest* and *latest* defining the parking time window of activity $a \in \mathcal{A}$
s_i, f_i	charging *start* and *finish* time of charging plan instance $i \in \mathcal{I}$
d_a	*energy demand* (kWh) of activity a (used in combination with $\mathcal{C}$)
$d_{min,a}, d_{max,a}$	*minimum and maximum energy demands* (kWh) of activity a
r_i	*charging rate* of instance i
k_r	*cost* for charging at rate r
p_i	calculated *profit* for instance i
x_i	variable depicting whether i is chosen into the schedule, $x_i \in \{0, 1\}$
a_i	*activity* that charging plan *instance* i was generated from
$P(t)$	total *available power* during time slot $t \in \mathcal{T}$
v_i^t	variable depicting whether $t \in [e_{a_i}, l_{a_i})$, $v_i^t \in \{0, 1\}$
m_i	*machine* at which charging plan instance i takes place

Table 2.1: Notation Overview

The planning time horizon $\mathcal{T}$ is divided into discrete time slots of fixed length (in our setting they are 15 minutes each, like in [11] and [31]). This was done to be able to handle the real time more efficiently. The rest of the notation will be described further in this chapter.

Next we present the models of the elements essential to our system — the *electric vehicle*, the *charging station* with its charging points and the *charging schedule* we want to generate, along with the *evaluation (profit) function* we will use.

At the end of this section we present the formulation of the EVRSTW problem as a mixed integer program (MIP), based on [7], [9].

2.2.1 Electric Vehicle

Each electric vehicle is represented by an activity $a \in A$ with the following attributes:

- parking time window $[e_a, l_a),\ e_a, l_a \in \mathcal{T}$ within which the non-interrupted charging period can be planned, i.e. the vehicle's arrival and departure time
- requested energy demand, which can be of two forms:

  MinMax model Minimum and maximum energy demands $d_{min,a}$ and $d_{max,a}$, where the minimum has to be satisfied, and the maximum is the amount required to completely charge the battery.

  SingleDemand model Single-valued energy demand d_a, used in conjunction with the predetermined set of completion degrees $\mathcal{C}$, which allows the fulfillment of prespecified amounts of the energy demand. In our setting these are 50%, 75% and 100%, where it can be observed that the analogy with the *MinMax* model is: $d_{max,a} = d_a$, $d_{min,a} = d_a/2$.

Note that the end of the parking time window is exclusive, i.e. charging can not be planned during l_a (details concerning discretization of time). The energy demand is determined either automatically by the battery's state of charge (SoC), or selected voluntarily by the EV owner (e.g. defining he wants to drive for at least 50km more).

The **MinMax model** offers more flexibility, but also introduces more complexity into the solution space, since more charging plan instances can be generated per activity. The single-valued demand and set $\mathcal{C} = \{0.5, 0.75, 1.0\}$ combined allow for at most 3 possible values of satisfied demand, while $d_{min,a}$ and $d_{max,a}$ allow for "any" value in between (depending on the charging rate selected, as we are about to see).

2.2.2 Charging Station

The first subproblem (interval scheduling) is the one of assigning activities to one of the charging points (machines) $m \in \mathcal{M}$, by the method which will be described in Section 3.1. These charging points can deliver power at charging rates from $\mathcal{R}$, which in our setting are $\mathcal{R} = \{3.7, 8.0, 11.0\}$ (kW).

We further assume the machines are all identical, i.e. all of them are able to operate at all of the charging rates. Each of those can, intuitively, accommodate at most one EV at every point in time.

Except for the parking resource, the charging station is also constrained by the available power curve, externally imposed by the electric grid. Available power is, analogous to time, discretized, with $P(t)$ representing the total power available at time slot $t \in \mathcal{T}$.

2.2.3 Charging Schedule

Here we will have a look at the charging schedule generation formalities and the profit of individual charging plan instances.

Charging Schedule Generation

Once the EVs have been allocated to their charging points, the next step is to solve the second subproblem — to devise a charging plan for each of the allocated activities. Charging should take place within the time window defined by activity's arrival and departure times. Each charging plan is determined by its charging rate, start time and the amount of energy demand it satisfies.

Charging takes place in the period $[s_i, f_i),\ s_i, f_i \in \mathcal{T}$, i.e. charging does not include the time slot f_i. The duration of a charging plan instance i is then $dur_i = f_i - s_i$.

To deal with multiple resources and variables (e.g. charging rates), we follow the approach of Bar-Noy *et al.* [2] who propose creating an "instance" for each combination of the variables for the considered activity. The problem is solved when a valid subset of instances is selected.

We simplify the problem by assuming identical machines and assigning activities to them in the first, preprocessing phase. This allows us to not consider machines when creating instances in the second phase, which reduces the search space significantly. The set of instances $\mathcal{I}$ is generated by creating an instance $i(a, r, dur, s)$ for each combination of the variables:

- $a \in \mathcal{A}$ — activity
- $r \in \mathcal{R}$ — charging rate
- $dur(d_a, r)$ or $dur(d_{min,a}, d_{max,a}, r)$ — charging duration, depending on the activity's energy demand(s) and the selected charging rate
- $s \in [e_a, l_a - dur]$ — charging start time, which has to respect the parking time window

In case the machines are non-identical, we would have to include a variable $m \in \mathcal{M}$ into the generating function as well.

Charging Plan Instance Profit

Further on, for each instance we calculate the profit p_i, defined as a weighted sum of the two factors:

- *completion degree* c_i — the ratio between the energy charged and the energy requested
- *battery lifetime preservation factor* $k_{r_i} = \frac{k}{r_i}$ — here we assume the cost of charging at a certain rate is inversely proportional to the charging rate

The profit of a charging plan instance is then defined as

$$p_i = \alpha\ c_i + (1 - \alpha)\ k_{r_i}, 0 \leq \alpha \leq 1 \tag{2.1}$$

The charging duration dur_i and completion degree c_i calculations depend on the EV energy demand model selected, as we will see next.

MinMax Model The charging duration for this model is calculated based on the activity's minimum and maximum energy demands, and on the charging rate the machine provides. The (discrete valued) charging duration is then:

$$dur_i \in [dur_{min,i}, dur_{max,i}] \tag{2.2}$$

where

$$dur_{min,i} = \left\lceil \frac{d_{min,a_i}}{r} \right\rceil, \ \forall r \in \mathcal{R} \tag{2.3}$$

$$dur_{max,i} = \left\lceil \frac{d_{max,a_i}}{r} \right\rceil, \ \forall r \in \mathcal{R} \tag{2.4}$$

The charging completion degree is then calculated as:

$$c_i = \frac{dur_i}{dur_{max,i}} \tag{2.5}$$

SingleDemand Model When the energy demand is single-valued, the set of completion percentages $\mathcal{C} = \{0.5, 0.75, 1.0\}$ is used to determine the different possible charging durations. Effectively, this yields at most 3 different duration values (in our setting) for each of the charging rates.

The possible charging durations for this model are obtained by:

$$dur_i = \left\lceil \frac{d_{a_i} \cdot c}{r} \right\rceil \quad \forall c \in \mathcal{C}, \forall r \in \mathcal{R} \tag{2.6}$$

Naturally, only those charging plan instances that fit into the parking time window $[e_{a_i}, l_{a_i})$ are allowed.

Figure 2.3 illustrates the relationship between the activity and its charging plan instance, along with their respective attributes.

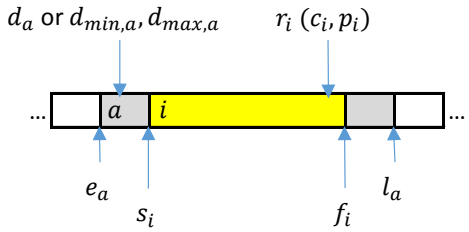


Figure 2.3: Activity (a) versus the charging plan instance (i) and their attributes

2.2.4 Evaluation Function

The total profit of the generated schedule is determined by the profits of each individual activity's selected charging plan instance — it is the sum of their profits ($x_i = 1$ depicts that instance i is included in the schedule):

$$Z = \sum_{i \in \mathcal{I}} p_i x_i \tag{2.7}$$

As we have seen, profits p_i depend on the fulfilled energy demand of the battery and on the chosen charging rate (following the *slower is better* principle for the battery health).

Additionally, we will evaluate schedules according to the number of EVs accomodated. This means not only the total induced profit is important, but a **fair ruler policy** is considered as well:

> *It is more fair to accomodate more customers with potentially lower total profit, than it is to satisfy less of their requests with higher total profit.*

Therefore, rejected activities (unsatisfied demands) will be considered an indicator of an undesired solution.

Other than that, we will also track the running times of different methods, as this will be of essential importance to the response time of the entire online system.

2.2.5 Mixed Integer Programming Formulation

The following MIP formulation we present is based on [7] and [9].

After generating the set of charging plan instances $\mathcal{I}$, the problem reduces to finding a set of binary decision variables $x_i \in \{0, 1\},\ i \in \mathcal{I}$ of maximum profit. $x_i = 1$ depicts that instance i was chosen as part of the charging schedule. We can now define the mixed integer programming (MIP) formulation for the EVRSTW problem, as follows:

$$Z = max \sum_{i \in \mathcal{I}} p_i x_i \tag{2.8}$$

$$s.t. \qquad \sum_{\substack{i \in \mathcal{I} \\ a_i = j}} x_i = 1, \quad \forall j \in \mathcal{A} \tag{2.9}$$

$$\sum_{\substack{i \in \mathcal{I} \\ t \in [s_i, f_i)}} r_i x_i \leq P(t), \quad \forall t \in \mathcal{T} \tag{2.10}$$

$$x_i \in \{0, 1\}, \quad \forall i \in \mathcal{I} \tag{2.11}$$

The objective is to maximize the total profit (Eq. 2.8). By equality (Eq. 2.9) we ensure that exactly one charging plan instance is chosen for each activity. In case of infeasible problem instances this should be turned into an inequality, as not all activities' demands would have been satisfiable. The inequality (Eq. 2.10) constrains the total power consumption per time slot to the available power.

In case the machines are non-identical, when we have to cope with the problem without being able to preprocess it (by solving the subproblem of allocating activities to machines separately), an additional constraint is needed ($v_i^t = 1$ if time slot t is part of a_i's parking time window, otherwise 0):

$$\sum_{\substack{i \in I \\ m_i = m}} v_i^t x_i \leq 1, \quad \forall m \in M, \forall t \in \mathcal{T} \tag{2.12}$$

Recall that in this case the instances are generated for *all* machines in $\mathcal{M}$. This constraint ensures that two vehicles are not parked at the same parking place (occupying the same charging point, i.e. machine) m in the same time slot.

The described MIP formulation was fed into the CPLEX solver and provided optimal solution benchmarks for our heuristics [7], [9].

2.3 Related Work

Here we thoroughly investigate the literature related to the EVRSTW problem at hand, with its wider environment being the general job (interval) scheduling, resource allocation and EV systems, including smart grid. A concise case study on a real world Belgian EV charging system, *BeCharged*, is also included.

Even though EVRSTW as depicted here has not been investigated before, similar problems, namely the ones contained within as its subproblems (see Section 2.1), do appear in literature.

The overview begins with the general job scheduling approaches, and continues with studies related to resource allocation. As local ratio technique offers good approximation algorithms for these problems, we have a brief note on it as well. Next, we take a look into known methods to schedule EV charging and also explore its wider environment embodied in EV systems. Finally, we provide a short analysis of the investigated literature, in relation to our EVRSTW problem.

2.3.1 Job (Interval) Scheduling

Arkin and Silverberg [1] analyzed scheduling of jobs with fixed start and end times back in 1985, mentored by Christos Papadimitriou. Their model does not allow for *preemption*, i.e. once the processing of a job has started, it can not be interrupted. Each job has an associated value, and the goal is to maximize the value of jobs completed by k identical machines. The algorithm they propose, based on the properties of interval graphs coloring problem and the reformulation as an instance of minimum cost flow problem, runs in $O(n^2 \log n)$ time, where n is the number of jobs. If the problem is further constrained by restricting each job to only a subset of machines on which it can be processed (i.e. non-identical machines), it becomes NP-complete. For this instance, given that the number of machines k is fixed, they propose an $O(n^{k+1})$ time algorithm.

Gabrel [15] analyzed the problem of scheduling non-preemptive jobs on k identical parallel machines. For each job, a subset of machines on which it can be scheduled is defined. Two cases have been studied: (i) each job has to be completed at fixed start and end times (Fixed job Scheduling Problem, FSP) and (ii) each job can be completed within a time window larger than its processing time (Variable job Scheduling Problem, VSP). A new model and heuristics, based

on graph theory, independent set and partition into clique concepts, are proposed to solve the two NP-complete problems.

He [16] studied the problem of scheduling non-preemptive jobs with time windows on a set of parallel, non-homogeneous machines. Every job has the following attributes: processing time, priority, machine subset satisfying its processing demands and multiple machine-dependent time windows during which it can be processed. The work presents an algorithm integrating constraint programming (used to check the validity of discovered solutions and to determine the values of constrained variables) and tabu search metaheuristic (which performs the neighborhood search process).

Kolen *et al.* [19] also focused on interval scheduling, a problem appearing in fields from crew scheduling to telecommunications. Here, not only the jobs' processing times, but also their starting times, are given. This case corresponds to the fixed start (and end) times setting of [1] and [15] (FSP case), with the distinct notion of multiple non-identical machines. An analogy is drawn between interval scheduling and dynamic resource allocation problem, with the machines being a resource whose allocation changes over time. The writing offers an overview of work on interval scheduling and then presents algorithmic results for two specific problem variants. In one setting, the difference between machines is presented by given availability intervals for each of them. In another one, the machines are continuously available, but ordered, with each job being given a "maximal" machine by which it can be processed. Algorithms based on circular arc graph coloring are provided. For the variant of ordered machine types (hierarchical interval scheduling), they show that the case of two machine types can be efficiently solved, whereas the problem for three machine types is NP-complete. Of special interest is the mentioned $O(n \log n)$ *left-edge algorithm* for optimal basic interval scheduling (as opposed to $O(n^2)$ complexity of Ford & Fulkerson's staircase rule), which minimizes the number of machines used. It assigns jobs to machines in order of nondecreasing starting times, using a machine used before whenever possible.

2.3.2 Resource Allocation

Phillips *et al.* [24] considered a class of difficult to approximate (NP) problems of non-preemptive scheduling of jobs with release and due dates, with the objective being maximization of the (weighted) number of on-time jobs. With the relaxed notion of approximation, they propose a pseudopolynomial-time algorithm that, given an instance of n jobs whose optimal schedule has value OPT, schedules a constant fraction of the n jobs within a constant factor times OPT (in many cases it can also be turned into a proper polynomial-time algorithm). The key idea of their *fractional schedule* setting is a novel α-point, which depicts the earliest point in time at which a cumulative α-fraction of the job has been scheduled up to. The algorithms' evaluation is based on the fraction of jobs scheduled, but they also explore the possibilities of getting "good schedules" where *all* the jobs are scheduled. While majority of the paper focuses on single machine scheduling, they do consider a problem of relevance to the thesis — the *Resource-constrained scheduling problem*, for which they propose first constant factor approximation algorithms (pseudopolynomial-time 4-approximation and polynomial-time 6-approximation), based on the framework of Bar-Noy *et al.* [2].

Darmann *et al.* [12] studied the *resource allocation problem* (RAP), where each job consumes some quantity of a bounded resource during a certain time interval (between start time and end time) and induces a given profit. This case, as opposed to the EVRSTW and the generalization in [24], does not allow for scheduling within a time interval larger than the job processing time. The goal here is to schedule a subset of jobs with maximal total profit such that the resource constraint remains non-violated. In general case the problem is NP-hard. Given the uniform resource consumptions, the problem turns into the interval scheduling problem, which is polynomially solvable. Main result of the paper is the proven NP-hardness even of the case when the profit values are uniform and the intervals proper, where the goal is to maximize the number of performed jobs. The method is based on proper interval graphs, cliques and ordered partition problem (NP-hard). They also give a deterministic $(1/2 - \epsilon)$-approximation algorithm for the general problem on proper intervals, improving on the previously known $1/3$ ratio for general intervals. The $1/2$-approximation greedy algorithm offered in the end requires both proper intervals and uniform profits to run in $O(n \log n)$ time.

Calinescu *et al.* [10] study the problem of finding a most profitable subset of n given tasks, each with a fixed start and finish time, a profit and a resource requirement that must not surpass the fixed quantity of available resource at all times. Three algorithms for restricted versions of RAP are presented, namely *Large Tasks Algorithm* (via dynamic programming), *Small Tasks Algorithm* (via randomized rounding) and *List Algorithm*. Suitably combined, these yield approximation algorithms for the general case. They propose a method of $(1/2-\epsilon)$-approximation (using LTA and STA) in polynomial time for this NP-hard RAP. In addition, a simple and fast $1/3$-approximation (using LA and an $O(n \log n)$ algorithm for maximum weight independent set in interval graphs [2]) is given.

2.3.3 Local Ratio Technique

Bar-Noy *et al.* [2] presented a general framework for solving resource allocation and scheduling problems. Their algorithms approximate maximum throughput or minimum loss by a constant factor for several resource-constrained problems, some of which are closely related to ours, such as job scheduling on parallel machines, resource allocation and bandwidth allocation for sessions in communication networks. Some other problems that fit into the framework are general caching, dynamic storage allocation and finding a maximum weight independent set in interval graphs. Models used combine time and resource constraints in a novel manner, and for some of the problems they offer the first constant factor approximation algorithms. These are simple, efficient and based on the *local ratio technique* (LRT), which is to be applied to our problem as well. The *Maximization problems and scheduling applications* chapter in [2] is of special interest to the thesis. Some of the EVRSTW notation has been taken over from this work. We will describe the details of the LRT in Section 3.3, where it has been tailored to our problem.

Bar-Yehuda *et al.* [3] further demonstrated LRT use in design and analysis of algorithms for various minimization and maximization problems (including covering, packing and scheduling problems). They also introduced the idea of the *fractional local ratio* paradigm.

2.3.4 EV Charging Strategies

Sugii *et al.* [29] proposed a genetic algorithm based scheduling method of charging electric vehicles. It manages to determine a sub-optimal charging timetable which satisfies the given electric load curve under the structural constraints of their system. The charging facility consists of several charging bays which offer multiple connectors to vehicles, but of which only one can be actively charging at a time. An ideal load curve is defined as a step function in order to make use of the late-night off-peak electricity prices. Problem is split into two: (i) *connecting* problem (selecting a charger to which the EV should connect) and (ii) *scheduling* problem (determine the starting time of charging). The genetic algorithm is applied to the latter, with the goal off fitting the total charging load curve to the ideal load curve. Individual evaluation is based on the sum of the squares of difference between the ideal load value and the total charging load value at each sampling time. They simulate several battery charge state settings on 20 EVs and 10 chargers with 2 connectors. The results show that controlled charging manages to level off the electric load as well as to reduce the capacity and cost of charging equipment, as opposed to the uncontrolled one.

Hutson *et al.* [18] proposed an intelligent *binary particle swarm optimization* (BPSO) based approach to schedule the usage of available energy storage capacity from EVs. Next to the vehicles being able to take power from the grid and charge the batteries, they introduce an idea of also providing power to the grid when parked, so called *vehicle-to-grid* (V2G) concept. A scalable parking lot model, with the objective of maximizing profit for the operator, is developed. BPSO is applied to schedule whether each vehicle should buy, sell, or hold at every time step it is parked. This general setting renders the charging periods as preemptive (i.e. charging periods are not necessarily contiguous). Each vehicle's SoC is expected to be at 60%, which is to be satisfied even in the case of early departure. A day is split into 1 hour intervals (to match the hourly prices), and an infinitely large connection to the grid is assigned to each parking lot, meaning they do not consider available power as a resource. Two case studies have been conducted. The first one takes the price curve and determines the best (maximum) selling and best (minimum) buying price — this way only one transaction occurs for each vehicle. This results in lower profit potential but the schedule is easy to determine. The second case study considers multiple transactions throughout the parking duration. This allows for higher profits but greatly increases the scheduling difficulty. However, the problem is separable, since there are no common constraints and the solution can be found for each vehicle individually, which greatly reduces its dimensionality. The results show that multiple transactions result not only in significantly higher profits, but also reduce the net power out to the grid.

Clement *et al.* [11] mention several kinds of alternative vehicles: hybrid electric vehicls (HEV), fuel-cell electric vehicle (FCEV) and plug-in hybrid electric vehicle (PHEV), but focus on the last one, stating that PHEVs might take up to 5% of total electrical consumption in Belgium and up to 30% of the market share by 2030. The model, in which the PHEVs charge at home, defines four essential charging periods for a residential radial network. Daily (winter and summer) load profiles are selected from available historical data, and given on a 15 minute time base. First they describe uncoordinated charging, where the vehicles are charged (at constant power of 4 kW) immediately when plugged in or after a fixed start delay, which leads to grid problems. Next proposed is coordinated charging which computes the optimal PHEV charge

profile (at variable charge rate) by minimizing the power losses. EV owners are now only allowed to define a point in time when the battery must be fully charged. The starting time of charging is determined by the *sequential quadratic optimization* program. This approach manages to decrease the power losses for all charging periods and seasons. They also make use of *stochastic programming* to forecast the daily load profile, which introduces an efficiency loss (as the charge profiles are no longer optimal for the specific daily load profile, i.e. the stochastic optimum differs from the deterministic one).

Sundström and Binding [30] compare two different approaches to optimize EV battery charging behavior with the goal of minimizing charging costs, achieving satisfactory state-of-charge (SoC) levels and optimal power balancing. The first one is based on a linear approximation, whereas the second one uses a quadratic approximation. The battery's non-linear and state-dependent model is used to evaluate the obtained solutions. They do mention the V2G concept, but do not apply it in their setting. The driving pattern prediction (vehicle arrivals, departures and energy requirements) is assumed to be available. The difference between the two methods turn to be minor and they conclude that a linear approximation is sufficient, as the resulting violations of the battery boundaries are less than 2% (however, in a perfect, deterministic setting).
They continue their work in [31] by proposing a method of planning the individual charging schedules of a large EV fleet while respecting the *constrained low-voltage distribution grid*. It has been tested in a simulation environment in which the movement and charging of individual EVs are simulated simultaneously. Here, the planning period is represented by 96 slots of 15 minutes (as in [11]) and the charging spots are rated at 16 kW. A centralized fleet operator is assumed, which handles data storage, trip forecasting (here assumed perfect), optimization, customer relationship and billing information, and communication. Three different charging schemes are compared: (i) *Eager charging* (connect and charge), (ii) *Price-based charging* (unconstrained grid, minimizing only the total cost of electricity used for the fleet) and (iii) *Grid-aware price based charging* (respecting the grid capacity as well). For (iii) they clearly separate the grid congestion evaluation and charging schedules optimization phase, using the concepts of load flow network, maximum flow problem solving and linear programming, while iteratively updating the grid constraints. Results show that both for eager and pure price-based charging, the grid is significantly overloaded. Including the grid constraints reduces the overload (however, with still present peaks), but slightly increases the total cost.

Sánchez-Martin and Sánchez [27] focus on the electric power management at parking garages, as new potential high concentrated electric consumption nodes. By exploring the benefits of implementing a consumption management control (CMC) in a 50 plug-in vehicles case study, they analyze power capacity requirements and costs, savings on energy consumption and penalties for non-supplied energy. The vehicles are modeled as EV and PHEV entitites with specific attributes, such as maximum battery charge (23 kWh and 7.2 kWh), charging rate (3.8 kW and 2.2 kW), arrival time, stay duration, and initial and final SoC. At the parking garage, *power* capacity is considered an actual parking *resource*, next to the several parking zones managing different charging status of the vehicles: waiting, charging and resting (FIFO queues). Vehicle arrivals are modeled as a Poisson process [17] with a different arrival rate for each hour of the day. The CMC attempts to avoid excesses over the total power capacity and to minimize the total

energy cost, based on three charging time frames: two valleys (0:00 – 3:00; 3:00 – 6:00) and off-valley (6:00 – 0:00). Simulations show that implementing the CMC: (i) delays a significant percentage of the supply energy to a period of lower energy prices, (ii) avoids overcoming the power capacity, (iii) reduces the contracted power capacity, and (iv) reduces the non-supplied energy levels. Total cost savings are rated at 34% to 50% when CMC is employed. Next, they state that the optimal installed power capacity for their 50 vehicles case study, given different PHEV-EV scenarios, is 40-50 kW per garage (from purely economical perspective). A "rule of thumb" is also established, saying that 1 kW per plug-in vehicle is a reasonable rate to set an installed power capacity.

Verzijlbergh *et al.* [32] investigate the impact of electric vehicle charging on residential low-voltage (LV) networks. Their approach is different from other studies in that the analysis is based on a large data set of real life driving patterns and electricity networks in Netherlands, allowing construction of realistic load profiles (as opposed to those generated by probabilistic methods). By comparing uncontrolled and controlled charging scenarios, they aim to answer to what extent will the future transformers and cable loadings change in case of large scale EV adoption. Two charging scenarios are examined: uncontrolled charging with fixed rate (3 kW or 10 kW) and controlled charging (with variable charge speed). The latter shifts the charging away from household profile peak. First experiments show that there seems to be enough capacity in current LV networks to charge all EVs, even up to penetration degree of 100% (assuming all EV charging can be controlled). Next they focus on year 2040, when the EV market is supposed to be saturated. They calculate future cables and transformers loadings, results of which show that most problems could be expected with transformers rather than with cables or voltages. Controlled charging is shown to be able to reduce the number of overloaded transformers and cables with approximately 25% and 8%, respectively.

Xu and Pan [34] consider a stochastic dynamic system with a finite number of PHEVs and an underlying power grid, in which the PHEV battery charging is scheduled to maximize the overall social welfare — derived from total customer utility, electricity cost and the non-completion penalty for unsatisfied PHEV's charging requests. They bring the EV charging problem in relation to *deadline scheduling problems*, but also state the important differences: (i) the cost is stochastic (due to uncertain future loads and renewable generation) and (ii) there are multiple charging facilities able to charge PHEVs simultaneously. They formulate it as an *infinite-horizon Markov desicion process* (MDP), which, unlike previous examples, allows for *preemption*. The model incorporates an exogenous source of uncertainty — the state of the power grid; and the stohasticity in the PHEV arrival process through a *Markov chain*. Main result of the paper is the novel *Less Laxity and Longer remaining Processing time (LLLP)* principle: priority should be given to vehicles that have less *laxity* (maximum number of non-charging states a vehicle can tolerate before its departure time; i.e. remaining parking time minus remaining charging time) and longer remaining processing times, if the non-completion penalty is convex, and if the operator does not discount future cost. They compare the performances of three stationary heuristic policies: *Earliest Deadline First*, *Least Laxity and Shorter remaining Processing time*, and LLLP. Results show that the LLLP principle can only reduce the expected cost, and its violation may result in significant losses of social welfare.

Zhang *et al.* [36] studied delay-optimal charging scheduling of EVs at a charging station

with multiple charging points. In their setting, the charging station is equipped with renewable energy (RE) generation devices and can also buy energy from power grid into its storage battery. An interesting point is noted, stating that local RE sources are the ones that should be used at least partially, to achieve real environmental advantages of EVs, compared to conventional vehicles. As in [34], an MDP framework is proposed, modeling the uncertainty of EV arrival, intermittence of renewable energy and the grid power price variations as independent Markov processes. The goal is to minimize the mean waiting time under the long term cost constraint. *Queue mapping* is proposed, converting the EV queue to the charge demand queue, along with the proof of equivalence between the average length minimization of the two queues. They derive two necessary conditions of the two-dimensional optimal policy of the formulated MDP, and finally investigate two specific stationary policies — radical and conservative.

2.3.5 EV systems

The following works offer insight into a somewhat wider framework of EV domain, including comprehensive EV systems, charging station selection, routing and reservation. Some of them delve into the recharge scheduling at individual stations as well.

Peças Lopes *et al.* [20] presented a conceptual framework of EV integration into electric power systems, covering two different domains: the *grid technical operation* and the *electricity markets environment*. Along with describing in detail all of the involved parties and their activities, several simulations are performed in order to illustrate potential impacts and benefits of EV integration, comprising steady-state and dynamic behavior analysis. In this work, they consider slow charging at home and in public charging points in residential areas, for about 12700 EVs of different types. Two case studies, in order to prove the efficiency of suggested framework, are presented: uncontrolled charging and advanced control EV charging strategies. *Dumb charging* approach limits the EV penetration level to 10%. Two smart charging techniques are implemented: a simple *dual tariff policy* (DTP) and a complex *smart charging* (SC) mechanism. The allowable share of EVs that could be integrated into the network has risen to 14% with DTP, and 52% with SC, which shifted around 50% of EV charging periods from peak to valley hours. They also model the EV's grid interface, which offers a local level of control. The work shows that adoption of advanced centralized EV charging control strategies allows for highest EV penetration level, without the need of grid reinforcements. Smart charging results in improved voltage profiles and lower congestion levels of the network.

Bessler and Groenbaek [7] address the efficient operation of public charging stations by proposing a routing (CS search and reservation) service, the work which has been conducted within the KOFLA project. Charging schedule optimization is considered as well, performed independently at each CS. The routing service is designed to serve a geographical region and represents a broker between EV users and energy providers. The planned communication protocols run over wireless channels available in current cellular networks. Users provide a parking time window and the energy demand requests, while the decentralized CS architecture and the routing server provide feedback. Routing reachability simulations with different EV penetration levels are run on Vienna region for three schemes: (i) *direct drive* (no charging), (ii) *charging in destination vicinity* (walking distance allowed) and (iii) *multimodal routing* (including public means of transport in the CS vicinity). Multimodal routing, as expected, offers a significant

improvement in terms of CS reachability. Regarding controlled charging at individual charging stations, a new problem is introduced as *Electric Vehicles Recharge Scheduling with Time Windows*, which is the topic of this thesis, and will be discussed in detail in the following chapters. Here, an *integer programming* approach is evaluated against a heuristic based on the *local ratio* technique.

Mal *et al.* [21] present a comprehensive system leveraging mobile and RFID technologies, aggregation middleware, and an aggregated charge scheduling algorithm, that effectively manages charging and V2G operations for cost savings and peak load reduction. A wide system architecture is given, detailing the parking garage access gate, charging station equipment and activities, an EV command portal application, and parking garage aggregation middleware. The Aggregated Charge Scheduler (ACS) optimizes charge scheduling for *electricity price*, which is also implicitly optimized for *electricity demand*. Its task is to charge the EV from initial SoC to final SoC within the time window defined by time of arrival and time of departure (including a buffer time). In case the charging demand can not be satisfied, the plan is rejected and the EV owner gets notified. 24 hour period of day is quantized into smaller time intervals, as well as the electricity price curve. As in [18], V2G operations are utilized to incur additional profits, given that an EV owner has opted to participate. Simulations are run for a hypothetical parking garage with 10 chargers delivering 6.6 kW and 30 vehicles, and cost and power usage results are compared with those of unmanaged charging. Charging gets scheduled during the cheapest intervals an EV is parked, while most of the discharge happens when the electricity price and demand are highest. Managed charging provides cost savings of 7-10%, and reduces peak demand load by 46-56%. V2G services, which can be used to incentivize EV owners, have shown to be able to generate a net profit of 5.6-11.7 cents per vehicle. The proposed implementation would be best suited for an enterprise environment, allowing for aggregation of large number of EVs. The system is in many ways similar to KOFLA. However, there is no mention of the available power constraint when scheduling.

Qin and Zhang [25] aim to minimize charging waiting time through intelligently scheduling charging activities, temporally and spatially. A theoretical study deals with formulating the *waiting time minimized charging scheduling* problem and deriving the performance upper bound. A practical distributed scheme is proposed as well, whose simulation results show that its design can achieve a waiting time near the theoretical lower bound. The main idea is a system-wide balanced distribution of charging demands, which should result in minimized overall waiting times. This is to be approximated by balancing the distribution of charging demands locally in every small portion of the system. A highway road system is considered — a 300 km by 300 km grid road network with 100 charging stations (CS) randomly deployed at entrances/exits. Five types of EV are randomly generated following the Poisson process at each entrance. A distributed, *reservation-based* scheduling scheme is introduced, where the CSs are networked and collaborate via Internet. Stations estimate waiting times in a dynamic environment and exchange their waiting time information with each other (and with the EVs). *Success probability* of a reservation is quantified by *reservation stability*, which is calculated based on historical records. Queuing time estimation at individual stations is done by simulating charging behaviors on a FIFO reservation queue. The work focuses on improving the driving comfort, rather than cost, by reducing the charging waiting times. In addition, no driver intervention is required. The

paper deals mostly with the CS selection, rather than detailing recharge scheduling at individual charging stations.

Yang *et al.* [35] study the problem of searching and selecting charging stations on a Taiwan national highway. Two approaches are proposed — first one utilizing only *local* information of the vehicle; while the second, *global* one, makes use of a Global CS-selection Server (GCS) through mobile computing, which provides info on the waiting times at available CSs. The highway is 372.7 km long and equipped with six intermediate service areas distanced 30-70 km, each of them comprising one CS. Three local CS selection algorithms — shortest-first, random and longest-first — have an issue of not being able to estimate the workload of a specific station (thus, the waiting time as well). Two global algorithms are proposed, distinguished by the timing of CS selection. The simulation model defines several objects and event types, EV arrivals being once again generated by Poisson process. Average waiting time, number of visited CSs and number of exchanged messages are used for performance evaluation of the proposed algorithms. As expected, global algorithms perform better than the local ones, with their advantage being even more noticeable given larger numbers of EV arrivals.

Barth and Todd [4] developed a *shared vehicle system* (SVS) simulation model, applied to a resort community in Southern California. SVS typically consists of a fleet of vehicles used several times each day by different users, and its main advantage being the reduction of number of vehicles required to meet total travel demand. A detailed analysis of the SVS concept is given, mentioning two types of SVS: (i) *round-trip SVS* and (ii) *multiple station SVS* (requiring relocation mechanisms), an instance of which is the model proposed here. The model deals with issues such as vehicle availability, vehicle distribution and energy management. Electric vehicles recharge when they are idle at the stations, according to a battery recharge algorithm (not described in detail). Simulations have shown that the most effective number of vehicles (in terms of satisfying the waiting time, addressing the vehicle availability issue) is in the range of 3-6 vehicles per 100 trips. If the number of relocations is to be minimized as well, the number grows to 18-24 vehicles per 100 trips. The SVS is most sensitive to the vehicle-to-trip ratio, the relocation algorithm used and the charging scheme employed. A preliminary cost analysis is performed as well, showing that such a system can be very competitive with present transportation systems (e.g. rental cars, taxis).

2.3.6 Analysis and Comparison

Interval Scheduling and Resource Allocation

The first subproblem of EVRSTW is analogous to *interval scheduling*, where the machines (charging slots), to which we allocate jobs (activities, vehicles), can be considered a resource [19]. This is equivalent to parking the vehicle to its assigned parking spot. Approaches considered here do not allow for preemption, which is reasonable in our setting as well — the parking periods are contiguous. The fixed start and end times of jobs to schedule, present in all of the works, are equivalent to arrival and departure times of our electric vehicles. Only Gabrel [15] mentions the variable job scheduling problem (VSP), where each job can be completed within a time window larger than its processing time. Our charging periods (present in the second subproblem) fall into this case, as they can be scheduled anywhere within the parking period. The

assumption we took is the one of identical machines, but Arkin and Silverberg [1], He [16] and Gabrel [15] do consider the non-identical ones too, allowing job assignment to only a subset of machines. (*Note*: Gabrel's work introduces some contradiction regarding identical machines along with their subsets when assigning jobs, but we will consider this case as non-identical). This variant brings the interval scheduling problem into the NP-complete domain. Kolen *et al.* [19] mention the $O(n \log n)$ left-edge algorithm for optimal basic interval scheduling, whose principles are analogous to our approach to solving the first subproblem.

The second subproblem of EVRSTW considers another resource — the available power. This one is analogous to *resource allocation* problems. The resource in all of the literature considered here is fixed, while in our setting the available power curve can vary over time. As we mentioned in Subsection 2.1.2, we can observe this subproblem as a single machine problem, just as Phillips *et al.* [24] did. The most similar setting to ours is the resource-constrained scheduling problem therein, which allows for fixed length tasks to "slide" within a larger time window. As opposed, Darmann *et al.* [12] and Calinescu *et al.* [10] consider only jobs of fixed start and end times.

Both interval scheduling and resource allocation problems deal with "jobs". The concept of a job differs in the two subproblems of EVRSTW. If we look through the first subproblem's point of view, our jobs to schedule are *activities* (parking periods of the EVs) with fixed start and end times. In the second subproblem, the jobs are *charging plans*, which are to be scheduled within the time intervals (parking periods) potentially larger than the jobs' processing times. For each activity we consider multiple charging plan instances, each having determined its own charging rate, start time and the amount of energy demand satisfied. This implies that all jobs belonging to one activity need to be *mutually exclusive* — each activity can be assigned only one charging plan. Yet the third view is that a "job" is the one of satisfying the EV's energy demand. Here, an issue of defining its *processing time* occurs, which depends on the charging rate selected and the energy demand fulfilled, i.e. on the instance assigned during the charging plan selection.

From this point on, we will consider that the *activities* define a fixed *time window* (parking period) within which the *job* (charging period) has to be scheduled, satisfying a certain amount of EV's energy demand at the specified charging rate. Detailed charging plan instance generation will be discussed in Chapter 3.

EV Charging and Systems

Clement *et al.* [11] and Sundström and Binding [31] talk about splitting time into discrete 15 minute intervals, the idea which we will also use when dealing with real time. Sugii *et al.* [29], Sánchez-Martin and Sánchez [27] and Sundström and Binding [31] consider both load and price curve when optimizing the charging schedules, while we focus only on satisfying the available power constraint, whereas Mal *et al.* [21] take the available power for granted and do not consider it at all. Just as us, Sugii *et al.* [29] split the problem into two subproblems — allocating the vehicles to charging points first, and then scheduling the charging process.

Hutson *et al.* [18] and Mal *et al.* [21] include an interesting *smart grid* element of the vehicle-to-grid (V2G) concept, where the energy exchange can go both ways, with EVs serving as energy storage and even inducing profits for both the service providers and the users. We will not deal with this concept in our work.

Sánchez-Martin and Sánchez [27] and Sundström and Binding [31] emphasize the need for a centralized EV management system. The former focuses on parking garages as charging facilities, and even establishes a ”rule of thumb”, saying that at each station 1 kW per plug-in vehicle is enough to satisfy the energy demands.

Of all the studies, only Verzijlbergh *et al.* [32] base their findings on realistic driving patterns and electric networks, while the data of the rest are stochastically generated.

Qin and Zhang [25] focus on improving the driving comfort, rather than cost, by reducing the charging waiting times. They focus not so much on the charging schedule at the charging station, but on charging station selection; just as Yang *et al.* [35].

All of the studies prove that using controlled charging is only beneficial, as it enables cost savings, grid load balancing and higher EV penetration levels.

2.3.7 BeCharged Case Study

BeCharged [5], a belgian company, is a regional electric mobility market pioneer since 2009. They develop and distribute charging stations for electric vehicles (cars, scooters, bicycles, etc.) and contribute to the development of a smart, user-friendly and affordable network infrastructure in preparation for the imminent mass production of electric vehicles.

Their charging stations are mostly situated in the Benelux region, but they also actively expand their reach in Europe (France, Switzerland, Germany, Luxembourg) and Asia. They supply company and underground car parks, shopping malls' parking lots, private homes etc.

Services

BeCharged acts as a service provider to both the charging station owners (hosts) and the electric vehicle owners (users). Their *MyBeCharged* management and billing platform is accessible through the web browser and a smartphone application.

Hosts can manage, monitor and control their charging stations using the *MyBeCharged* back office management system. They can overview the charging stations' status and usage, monitor the financial transactions, adjust settings such as prices, user groups, zones, lighting, duration of charging sessions, etc. The host can also apply a price differentiation for use (€/hour) and/or consumption (€/kWh) via the web interface geographically, periodically and via user groups. The rates are notified to the users on the web platform, on the interactive display and/or the printout of the charging station. Financial transactions between users and hosts are carried out by BeCharged.

Their advanced software technology is in line with the *smart grid* concept, the ”intelligent electricity network”. This new electricity grid is envisioned as interactive — customers can generate and sell energy to the grid, instead of only consume it. They can decide when they wish to purchase or sell energy, and at what rate. This allows the hosts to time the replenishment of their station's electric supplies to ensure the lowest cost and the greatest efficiency, as well as to sell the energy stored in the battery back to the grid when it is most profitable.

BeCharged's charging stations ensure efficient charging cycles at the local level. Charging sessions can be set to preset time periods, when for example, more solar energy is generated locally, or when the local industrial grid has less load.

Corporate or personal **users** can recharge their vehicles at private and public charging stations, the usage of the latter requiring appropriate activation. Public charging stations can be activated in various ways — RFID cards, mobile phones, NFC (Near Field Communication) chips, debit cards or via SMS. A prerequisite is an online registration at *MyBeCharged* platform, where users can also request their personal RFID cards, manage their financial transactions, receive status information about the charging stations and make reservations.

The smartphone application allows users to locate charging stations in the area and make a reservation to ensure availability upon arrival. Reservations last for 15 minutes. It also enables them to remotely activate or deactivate charging sessions and follow the energy consumption.

The National Railway Company of Belgium (NMBS/SNCB) wants to provide 34 stations with electric charging points by October 2014. Currently, 7 Belgian cities host the service at their main train stations. Each of these stations provides 6 charging points for cars, 6 charging points for bicycles and 6 charging points for motorcycles. NMBS wants to further extend the station in Brugge and turn it into a mobility centre: drivers who use their car in a smaller radius could leave their vehicle here to charge and continue their journey by train.

Controlled Charging

Of special interest to the thesis is how does BeCharged plan the EV charging at individual stations. They named their applied algorithm *smart-grid* (not to be confused with the widely recognized, more general term *smart grid*, the modernized electrical grid).

On 2 July 2013 they implemented the first Belgian smart-grid application at the charging-enabled parking lots of several large train stations. This application distributes the currently available power among electric vehicles while taking into account each vehicle's priority, estimated battery capacity and departure time. It prevents a grid overload during peak hours and it plans ahead to shift most power consumption to off-peak hours to make sure the cars are fully charged by the time they leave. At these charging stations they operate with 6 sockets, each providing about 21 kW.

They allow for immediate reservations within a 30 km radius, and only for 15 minutes. The driver checks her smartphone for stations nearby to reserve one. In larger stations, there is a central device on which the user can activate his socket. When the smart-grid algorithm is active, the user will be requested to enter his vehicle data, desired battery charge percentage and departure information.

The scheduling is performed in presence of a reschedule event, initiating the division of the available power among plugged-in cars. These events are either induced by a timer every 5 minutes, or triggered when a new session starts or stops, or the available power changes. The algorithm plans the whole process until the last car is set to depart. At the next reschedule, it does the whole planning again but with updated information from the energy meters. The charging normally starts immediately with the minimum required power, so the driver feels more comfortable, as he can see his car charging before he leaves.

In contrast to our setting, their rescheduling allows for variable charging rates over the charging period. Also, the charging periods are interruptible (preemptive) — if there is not enough power, they shut down the sockets with the lowest priority, and possibly start them up again later.

Several ideas can be recognized as shared among the KOFLA project and the services provided by BeCharged. *Reservations* by users is one of the essential use cases in KOFLA, though not limited to only 15 minutes in advance as here. The concept of Brugge station as a mobility center — enabling users to leave their vehicles charging and continue commuting via train — is in line with KOFLA's *multimodal routing* concept, which considers the usage of public transport when proposing charging stations to the user.

The method by which *controlled charging* is performed at BeCharged, however, differentiates from the one proposed in this thesis. This will be described in the following chapter.

CHAPTER 3

Solving the EVRSTW Problem

In this chapter we describe the solution approaches to the EVRSTW problem developed under the course of this thesis.

First we will have a look at the common parts to all of the applied methods: *i*) solving the first subproblem of allocating activities to machines, and *ii*) generating the charging plan instances.

Next we continue with presenting the three heuristic methods to solve the second subproblem, of finding the best charging plan instances for activities allocated to machines. These are the *greedy heuristic* (GH), the *local ratio technique* (LRT) and the *min-conflicts heuristic* (MCH).

Finally we discuss the modifications necessary to provide an interface to the real-world online planning system, by integrating the scheduler into the KOFLA framework via a charging station controller.

3.1 Allocation of Activities to Machines

The first phase of solving the EVRSTW — allocating the activities to machines — is a preprocessing step, shared by all of the methods dealing with the second subproblem. We will therefore describe it up front, and then see how to solve the second subproblem in Section 3.3.

This step is possible only in the case of identical machines (all able to charge at all rates in $\mathcal{R}$), which we do assume. When the machines would not be identical, this step could not be used and the charging plan instance generation would have to include all of the available machines.

The proposed algorithm is in line with the one of Kolen *et al.* [19], the so called *left-edge algorithm* running in $O(n \log n)$ time. It is a greedy algorithm providing optimal solutions to the basic interval scheduling — the problem present as our first subproblem. It is optimal in the sense that it uses the *minimum number of machines* to accommodate the presented jobs (in our case — activities).

The procedure first sorts the activities in $\mathcal{A}$ according to their *earliest* (e_a) time in non-decreasing order. It then iterates over the sorted activities, finds the first machine free at that point in time, and allocates the activity to it. The machines are checked in a fixed order, i.e. considered a list, not a set. The method is outlined in Algorithm 3.1.

Algorithm 3.1: Allocation of Activities to Machines

Input: List of activities $\mathcal{A}$
Output: Optimal activities-to-machines allocation

```
sort activities A by non-decreasing order of e_a
for a ∈ A do
    for m ∈ M do
        if m not busy at e_a then
            assign a to m
            break
        end
    end
end
```

This yields an optimal solution to the interval scheduling problem. We are now ready to proceed with solving the second subproblem of EVRSTW — the *charging plans assignment*. But first, we will explore how to generate the feasible charging plan instances for our activities.

3.2 Charging Plan Instances Generation

Before we describe our solution approaches to the second subproblem of assigning the "best" valid charging plan instances, we will have a look at another common aspect of all the proposed methods — the charging plan instances generation for our activities. The formalities have been described in Subsection 2.2.3, so we will now have a look at the outline of the algorithm.

As mentioned, the charging plan instances for a certain activity are generated for all the combinations of the charging rates $r \in \mathcal{R}$, charging durations dur (determined by $r \in \mathcal{R}$ and $c \in \mathcal{C}$) and start times s. Of course, only the feasible plans that satisfy the constraint of the parking time window $[e_a, l_a)$ are accepted into the generated set.

The local ratio technique requires that set $\mathcal{I}$ consists of all charging plan instances of all the activities in $\mathcal{A}$. On the other hand, the min-conflicts and greedy heuristics only need instance sets of a single activity at a time — we depict these as $\mathcal{I}_a$. We will show how to obtain the individual sets $\mathcal{I}_a$, since $\mathcal{I}$ can be easily achieved from individual $\mathcal{I}_a$ sets by a set union:

$$\mathcal{I} = \bigcup_{a \in \mathcal{A}} \mathcal{I}_a$$

The approach to calculating the charging duration and completion degree of a charging plan differs between the two energy demand models. Algorithm 3.2 outlines the charging plan instance generation for the **MinMax model**. We see that the charging duration has to be determined (according to the chosen charging rate) *before* the completion degree (Eq. 2.2 — 2.5). Only then can we calculate the profit of an instance. Since the charging duration, and thus the completion degree, can possibly take more values than in the case of the **SingleDemand model**, we can expect to have more instances generated.

Algorithm 3.2: MinMax Model Instance Generation

Input: Activity a
Output: Set of valid charging plan instances for the activity a

$\mathcal{I}_a \leftarrow \emptyset$
for $r \in \mathcal{R}$ **do**
  $dur_{min} \leftarrow \left\lceil \frac{d_{min,a}}{r} \right\rceil$
  $dur_{max} \leftarrow \left\lceil \frac{d_{max,a}}{r} \right\rceil$
  for $dur \leftarrow dur_{min}$ **to** dur_{max} **do**
    if $dur \leq l_a - e_a$ **then**
      for $s \leftarrow e_a$ **to** $l_a - dur$ **do**
        $c \leftarrow \frac{dur}{dur_{max}}$ `// completion degree`
        calculate p based on c and r `// profit`
        $i \leftarrow$ `inst(`s, dur, c, r, p`)`
        $\mathcal{I}_a \leftarrow \mathcal{I}_a \cup \{i\}$
      end
    end
  end
end
return $\mathcal{I}_a$

On the other hand, the completion degrees are predefined by set $\mathcal{C}$ in the **SingleDemand model**. Charging durations are therefore calculated only *after* one of those has been selected, as seen in Algorithm 3.3.

Algorithm 3.3: SingleDemand Model Instance Generation

Input: Activity a
Output: Set of valid charging plan instances for the activity a

$\mathcal{I}_a \leftarrow \emptyset$
for $c \in \mathcal{C}$ **do**
  for $r \in \mathcal{R}$ **do**
    $dur \leftarrow \left\lceil \frac{c \cdot d_a}{r} \right\rceil$
    if $dur \leq l_a - e_a$ **then**
      for $s \leftarrow e_a$ **to** $l_a - dur$ **do**
        calculate p based on c and r
        $i \leftarrow$ `inst(`s, dur, c, r, p`)`
        $\mathcal{I}_a \leftarrow \mathcal{I}_a \cup \{i\}$
      end
    end
  end
end
return $\mathcal{I}_a$

Greedy Instances

We will be using a special subset of the charging plan instances for the greedy heuristic. Denominated as a set of *greedy instances*, it is comprised of charging plan instances having their:

a) charging rate fixed at the lowest charging rate $r_i = r_{min} = \min \mathcal{R}$

b) charging start time fixed at the beginning of the parking time window $s_i = e_{a_i}$

These will be incorporated into the greedy heuristic, to simulate the uncontrolled (unplanned) EV charging environment. Generation of these instances is simplified, requiring no loops over the charging rates nor possible starting times, but only over the possible completion degrees, resulting in much smaller size of sets $\mathcal{I}_a$ (and $\mathcal{I}$).

3.3 Heuristic Methods to Solve the Charging Plans Assignment Problem

Here we describe the three solution approaches to solving the second subproblem of EVRSTW — the one of assigning the "best" charging plans to *all* allocated activities, under the imposed available power constraint, with the goal of achieving maximum profit.

First we introduce the *greedy heuristic*, which should simulate the uncontrolled ("dumb") charging. Next we have a look at the *local ratio technique* [2], [3], adapted to our problem at hand. Finally we propose an algorithm based on the *min-conflicts heuristic*, which should improve on the two former approaches' results.

3.3.1 Greedy Heuristic

The greedy heuristic (GH) was designed to simulate "dumb", plugin-and-charge behaviour. But it is *not* an *entirely "dumb"* approach, since it is designed to preserve the available power constraint satisfied — it rejects the charging plans (and activities) whose addition to the schedule would overshoot the available power.

GH considers only those charging plan instances generated by the *minimum charging rate* ($r_i = r_{min}$) and starting at the *beginning of the activity's parking time window* ($s_i = e_{a_i}$), i.e. the *greedy instances*. The instances for a single activity are trialled in the order of the *highest profit first*. Since the charging rate is fixed, we can deduce (from Eq. 2.1) that profit is directly proportional to the completion degree of the charging plan instance.

This approach is outlined in Algorithm 3.4.

We can notice that the activity (which passed through the preprocessing phase, i.e. there was room for its parking) can be rejected from the schedule in two cases:

1. *Parking too short*. The evidence of this case is that no greedy instances (those charging at the slowest rate and starting at the beginning of the parking time window) could be generated.

Algorithm 3.4: Greedy Heuristic

Input: Set of allocated activities $\mathcal{A}$
Output: Charging schedule consisting of activities with assigned charging plan instances

```
for a ∈ A do
    I_a ← "greedy" instances for a
    if I_a is empty then                                   // parking too short
        increment the unscheduled counter
        continue
    end
    sort I_a by decreasing profit
    schedFlag ← False
    for i ∈ I_a do
        if i does not violate the available power constraint then
            assign i to a
            add a to the schedule
            schedFlag ← True
            break
        end
    end
    if not schedFlag then
        increment the unscheduled counter          // no power for this activity
    end
end
```

2. *Not enough power*. Even after testing all the possible greedy instances for the activity, none of them exist which do not violate the available power constraint during their charging period.

For these two cases we increment the counter and report the total number of unscheduled (in the manner that no charging plan has been devised for them) activities at the end.

This approach, as expected, performs poorly. The first issue is the activity's time window: if it is too short, the greedy instance set will be empty. This happens when even the minimum energy demand of the activity can not be satisfied by the charging rate r_{min} within the parking duration. The second issue is a common trait of greedy algorithms: if the current charging plan instance can not fit into the schedule, it gets thrown away and never observed afterwards. These two factors result in many activities not being scheduled at all. We look forward to improve that with the following approaches.

3.3.2 Local Ratio Technique

The local ratio technique belongs to a class of stack based heuristics [8], and has been proposed in packing, resource allocation and scheduling problems to, among other uses, maximize bandwidth throughput [2], [3]. This technique, adapted to our problem in [7], where the bandwidth corresponds to the available power, is described next. For this type of problems, the algorithm

can achieve a 1/3 approximation, which gets better as the ratio between the charging rate of individual activities and the total available power is smaller.

LRT General Framework

In the local ratio general framework the input consists of a set of *activities*, each requiring the utilization of a given limited *resource*. The amount of resource available is usually fixed over time, but the authors allow for generalization to the case where the amount of available resource may change over time.

The activities are specified as a collection of sets $\mathcal{I}_{a_1} \ldots \mathcal{I}_{a_n}$, each representing all possible instances of a single activity. An instance $i \in \mathcal{I}_{a_k}$ is defined by the following parameters:

1. A half-open time interval $[s_i, f_i)$ during which the activity will be executed. The s_i and f_i are called the *start-time* and *end-time* of the instance. In our problem, the "execution" of the activity depicts that the charging plan is being realized.
2. The amount of resource required for the activity, referred to as the *width* of the instance and denoted w_i (the terminology inspired by bandwidth allocation problems). Naturally, $0 < w_i \leq 1$. In the context of the EVRSTW problem, the "width" is the share of the charging rate in total available power, $w_i = \frac{r_i}{P}$.
3. The *profit* $p_i \geq 0$ gained by scheduling this specific instance of the activity.

Different instances of the same activity may have different parameters of duration, width, or profit. A schedule is then a collection of instances. It is feasible if: *a*) it contains at most one instance of every activity; and *b*) for all time instants t, the total width of the instances in the schedule whose time interval contains t does not exceed 1.

In the *throughput maximization problems* we are asked to find a feasible schedule that maximizes the total profit accrued by instances in the schedule. In the *loss minimization problems* we seek a feasible schedule of minimum penalty, where the *penalty* of a schedule is defined as the total profit of activities *not* in the schedule. Each activity is restricted to be assigned a single instance and the profit of an activity is then the profit of its instance. Our EVRSTW problem maps perfectly into the framework, under the niche of throughput maximization.

LRT Algorithm

The iterative LRT algorithm works as follows: the charging plan instances $\mathcal{I}$ are first sorted in non-decreasing order of their end-times (f_i). The process is then split into two phases:

Phase One We iteratively select an instance $\tilde{i}$ with minimum end-time and decrease the profits of all instances that *a*) belong to the same activity; and *b*) overlap with the selected instance. All encountered instances with non-positive profits are deleted, and the instance $\tilde{i}$ is pushed onto a stack. When all of the instances have been considered, we proceed to the second phase.

Phase Two We construct the schedule by popping the instances from the stack and adding to the current schedule those which do not violate the total power constraint (Eq. 2.10).

To introduce some additional notation, let $\mathcal{I}_{a_i}$ denote the set of all possible instances belonging to the same activity as i, and $\mathcal{I}'(i)$ the set of instances overlapping instance i but belonging to activities other than a_i. We consider the "overlapping instances" as those whose activities (i.e. parking periods) overlap, not just themselves (charging periods).

The profit of the instances is updated according to the rule:

$$p_i = p_i - p_{\tilde{i}} \cdot \begin{cases} 1 & i \in \mathcal{I}_{a_{\tilde{i}}} \setminus \{\tilde{i}\} \\ \beta \cdot w_i & i \in \mathcal{I}'(\tilde{i}) \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

We set $\beta = 2$, as our setting corresponds to "narrow" instances [3] ($w_i \leq 1/2, \forall i \in \mathcal{I}$) - meaning more than one instance can be scheduled to execute simultaneously .

Rather than using a stack, whose sole purpose is to revert the order in which we consider the instances, we implemented the **sweep-line approach** from [2], which reduces the worst case time complexity to $O(n \log n)$. The analogy to the stack is reached by sorting the instances in a list and traversing it in both directions.

Here, the time interval of each instance has two endpoints (start-time and end-time, or s_i and f_i in our notation). $i(t)$ denotes the instance which endpoint t belongs to. We sort the $2n$ endpoints of the instances in a list $\langle t_1, t_2, \ldots, t_{2n} \rangle$ such that $k < l$ if $t_k < t_l$ or if $t_k = t_l$ and t_k is the end-time of $i(t_k)$ and t_l is the start-time of $i(t_l)$. We then consider the endpoints in order, reducing profits as we go along and deleting an instance whenever its profit is found to be non-positive. The pseudo-code of the first phase is shown in Algorithm 3.5. π_i denotes the "current" profit of an instance.

Notice the application of the profit update rule (Eq. 3.1) in lines 10-15.

Once the first phase is completed, we traverse the surviving endpoints in the reverse order and construct the schedule $\mathcal{S}$. We maintain the variable W with the following invariant: when endpoint t is reached, W holds the total width of the instances containing t which have been added to $\mathcal{S}$. Algorithm 3.6 outlines the second phase.

The amount of resource used at any point has to satisfy $0 \leq W \leq 1$, and thus $0 \leq w_i \leq 1$. We scale the instance's resource consumption to this range by setting $w_i = \frac{r_i}{P}$. This is fairly simple when the amount of resource is fixed over time.

The problem arises if the available resource (power) varies over time (as can be expected in the real-world setting), when the calculation of w_i varies in the interval $[s_i, f_i)$. The conservative rule in (3.2) would only waste resources because even a short drop in the available power would affect all the charging periods around it. On the other hand, using the rule of averaging the consumption as in (3.3) causes overshooting the available power and overloading the grid.

$$\bar{w_i} = \min_{t \in [s_i, f_i)} w_i(t) \tag{3.2}$$

$$\bar{w_i} = \frac{\sum_{t \in [s_i, f_i)} w_i(t)}{dur_i} \tag{3.3}$$

After solving the integer problem exactly (which turned to be excessively time-demanding, even non-feasible for bigger problem instances, due to the problem's NP-complete nature), fol-

Algorithm 3.5: Phase One of the Sweep-line LRT

Input: Set of activities (with their instances)
Output: Set of feasible instances

sort the $2n$ endpoints in non-decreasing order
initialize $\pi_{\tilde{i}} \leftarrow p_i$ for all $i \in I$
for $k \leftarrow 1$ **to** $2n$ **do**
  $\tilde{i} \leftarrow i(t_k)$
  if t_k *is the end-time of* $\tilde{i}$ **then**
    $p \leftarrow \pi_{\tilde{i}}$
    if $p \leq 0$ **then**
      delete $\tilde{i}$
    else
      for $j \in \mathcal{I}_{a_{\tilde{i}}} \setminus \{\tilde{i}\}$ **do**
        $\pi_j \leftarrow \pi_j - p$
      end
      for $j \in \mathcal{I}'(\tilde{i})$ **do**
        $\pi_j \leftarrow \pi_j - \beta \cdot w_j \cdot p$
      end
    end
  end
end

Algorithm 3.6: Phase Two of the Sweep-line LRT

Input: Set of feasible instances' endpoints
Output: Schedule $\mathcal{S}$

$\mathcal{S} \leftarrow \emptyset$
$W \leftarrow 0$
sort the endpoints list E backwards
for $t \in E$ **do**
  $\tilde{i} \leftarrow i(t)$
  if t *is the end-time of* $\tilde{i}$ **then**
    if $W + w_{\tilde{i}} \leq 1$ **then**
      $S \leftarrow S \cup \{\tilde{i}\}$
      $W \leftarrow W + w_{\tilde{i}}$
      delete the endpoints of the instances in $\mathcal{I}_{a_{\tilde{i}}} \setminus \{\tilde{i}\}$
    else
      delete the endpoints of $\tilde{i}$
    end
  else // t is the start-time of $\tilde{i}$
    $W \leftarrow W - w_{\tilde{i}}$
  end
end

lowed by the use of greedy and local ratio heuristics (which had the problem of not scheduling all the activities), we propose a novel technique based on the ideas of the min-conflicts heuristic.

3.3.3 Min-conflicts Heuristic

This section describes our new solution for the EVRSTW problem, based on the core ideas of the *min-conflicts heuristic*. This method was proposed by Minton et al. [22] and has proven very useful on solving different constraint satisfaction problems (CSP). Min-conflicts heuristic has been applied, among others, for the Hubble Space Telescope scheduling problem [22], Workforce scheduling [23], Break scheduling [6], N-Queens problem [22], etc., and has also been used in combination with noise strategies such as random walk [33]. Furthermore, a similar algorithm to the min-conflicts heuristic is currently one of the best local search strategies for solving boolean satisfiability problems [28].

Instead of constructing a feasible solution gradually, the min-conflicts heuristic generates a suboptimal *initial solution* and then attempts to improve it iteratively. The main idea of this method is to concentrate the search in the neighborhood of the variables that are in *conflict*. This technique initially assigns random values to all variables of the problem to be solved. The current solution is then improved with the following strategy: in each iteration we randomly select a variable that is in conflict (i.e., the variable appearing in a constraint that is still violated). The new value that minimizes the number of conflicts is assigned to the selected variable. The ties are broken randomly. Note, that different strategies can be applied to escape from the *local optima*. If the number of conflicts can not be minimized, even the solutions with more conflicts or same number of conflicts can be accepted. Alternatively, random strategies like *random noise* or *random walk* can be introduced in the min-conflicts method.

In order to apply the min-conflicts strategy to our problem, we first formulate it as a *constraint satisfaction problem* (CSP) consisting of variables, their domain and constraints. In our case the activities (cars) represent the *variables*, the instances (charging plans) represent the *values*, and any time slot-wise power constraint violation represents a *conflict*. The final state with no conflicts is when the power consumption in each time slot is **smaller than or equal to** the available power, i.e. when the power constraint is not violated. As we will see, the concept of adding some random noise will be introduced to escape the local optima as well.

Internal Solution Representation

An essential part of good system design is an efficient, yet simple, data structure.

We represented the schedule as a two-dimensional matrix, of dimensions $|\mathcal{M}| \cdot |\mathcal{T}|$. The planning time horizon $\mathcal{T}$ is discretized into time slots of fixed length (in our setting they are 15 minutes each). This was necessary to make the search space feasible.

Presence of the parked EVs and their charging periods is appointed by their activity IDs. The absolute value depicts the activity's ID, while the sign depicts whether the charging is active or not: If the value is positive — the charging process is active; if it is negative — charging is not active, i.e. the vehicle is only parked at the charging point. Machines are free where the matrix value is 0. An example of an internal schedule representation can be seen in Figure 3.1.

```
::: Schedule :::
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 |  m
-------------------------------------------------------------------------------------------------------------------------------------
  0   0   0   5   5   5   5   5  -5  -5  -5   0 -15  15  15  15 -15 -15 -15 -15   0   0   0   0   0   0   0   0   0   0   0 |  0
  0   0   0   0  -4  -4  -4  -4   4   4  -4  -4  -4  -4   0 -17 -17  17  17 -17 -17   0   0   0   0   0   0   0   0   0   0 |  1
  0   0   0   0   7   7   7   7   7  -7  -7   0 -16 -16 -16 -16  16  16 -16 -16 -16   0   0   0   0   0   0   0   0   0   0 |  2
  0   0   0   0  12  12  12  12  12  12 -12 -12 -12 -10 -10 -10 -10 -10  10  10  10  10 -10   0   0   0   0   0   0   0   0 |  3
  0   0   0   0   0   1   1   1   1  -1  -1   0   0   0   0   0  -9  -9  -9  -9  -9  -9  -9   9   9  -9   0   0   0   0   0 |  4
  0   0   0   0   0 -19 -19 -19 -19  19  19  19  19  19 -19   0   0   0   6   6   6   6  -6  -6  -6  -6  -6   0   0   0   0 |  5
  0   0   0   0   0   0   0   0 -14 -14 -14 -14 -14 -14  14  14  14  14 -14   8   8   8   8   8   8   8  -8  -8  -8   0   0 |  6
  0   0   0   0   0   0   0   0   0  -2  -2  -2   2   2   2   2  -2   0  18  18  18  18  18  18  18  18 -18 -18 -18   0   0 |  7
  0   0   0   0   0   0   0   0   0   0   3   3   3  -3  -3  -3   0   0   0  20  20  20  20  20 -20   0   0   0   0   0   0 |  8
  0   0   0   0   0   0   0   0   0   0  11  11  11  11  11  11  11  11 -11   0   0   0   0   0   0   0   0   0   0   0 |  9
  0   0   0   0   0   0   0   0   0   0 -13  13  13  13  13  13  13 -13 -13 -13   0   0   0   0   0   0   0   0   0   0 | 10
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 | 11
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 | 12
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 | 13
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 | 14
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 | 15

  0   0   0   8  20  28  28  28  28  15  15  19  23  26  26  26  22  26  27  27  27  27  19  23  15   7   4   0   0   0   0   0 | P
Power Limit:
 30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30  30
```

Figure 3.1: Example of internal schedule representation

Additional to this 2-D integer matrix, a hashed map is used with activity IDs as keys. Map values are activities themselves, containing all the information about the assigned charging plan instances. This representation is of huge importance for the performance, since it allows for rapid access to the activity (and its instance) attributes while traversing the search space and looking for improvements.

The available power and power consumed are represented as integer arrays, per time slot. This data structure choice plays a remarkable role in the optimization step, as we will soon see.

Neighborhood Search

After the initial charging plan instance has been assigned to each of the activities, potentially violating the power constraint, the local *neighborhood search* process starts, in an attempt to reduce the conflicts. The *neighborhood* is defined by all feasible charging plan instances of a randomly chosen activity which is charging in a randomly selected time slot. The *search* is then performed by inspecting the defined neighborhood in a manner presented by Algorithm 3.7.

The time slots violating the power constraint (T^-) are those having a *negative power difference* — the difference between available power and power consumption. Local optimization stops when there are either no more time slots with negative power differences or when the time limit is reached.

We introduce *stochasticity* into the heuristic in three occasions:

- randomly choosing a violating time slot (line 3)
- randomly choosing an activity whose charging is active (line 4)
- random noise in choosing a new instance of the activity (lines 6-8)

The amount of random noise is specified by the probability parameter p_{noise}, meaning that in p_{noise} fraction of cases we choose a random instance instead of the one that most improves the current. This is to enable *escaping the local optima*.

Algorithm 3.7: Search

Input: Activities with assigned initial instances
Output: Locally optimal schedule

$T^- \leftarrow$ list of time slots violating the power constraint
while T^- *is not empty and time limit not exceeded* **do** `// conflicts exist`
  $t \leftarrow$ a random time slot from T^-
  $a \leftarrow$ a random activity charging at t
  $\mathcal{I}_a \leftarrow$ all feasible instances for a
  $p \leftarrow$ a random number from $[0, 1]$
  if $p < p_{noise}$ **then** `// some random noise`
    $i \leftarrow$ a random instance from $\mathcal{I}_a$
  else
    $i \leftarrow$ `bestImprovement(`a, $\mathcal{I}_a$`)`
  end
  assign instance i to a
  update schedule and T^-
end

In the remaining cases, we select the best possible instance by inspecting the neighborhood using the strategy of *best improvement*. This is where the most of the complex computation is done, the outline of which is shown in Algorithm 3.8.

Algorithm 3.8: Best Improvement

Input: Activity a with an assigned charging plan instance, and a set of all feasible instances $\mathcal{I}_a$
Output: A new charging plan instance for the activity

$i \leftarrow i^* \leftarrow a$'s current instance
$\Delta P^* \leftarrow$ `calculatePwrDiffs()`
negSumBest $\leftarrow$ `sumNegatives(`ΔP^*`)`
(*) *sort* $\mathcal{I}_a$ by profit; or *shuffle* them `// set as a parameter`
for $i' \in \mathcal{I}_a \setminus \{i\}$ **do** `// search for the best instance`
  $\Delta P' \leftarrow$ `updatePwrDiff(`i', i`)`
  negSumNew $\leftarrow$ `sumNegatives(`$\Delta P'$`)`
  if *negSumNew* $>$ *negSumBest* || (*negSumNew* $==$ *negSumBest* $\&$ $p_{i'} > p_{i^*}$) **then**
    $i^* \leftarrow i'$
    $\Delta P^* \leftarrow \Delta P'$
    if *negSumNew* $== 0$ **then** `// all conflicts resolved`
      break
    end
  end
end
return i^*

ΔP is an array of power differences of length $|\mathcal{T}|$, values of which are calculated per time slot as:

$$P_{available}(t) - P_{consumed}(t).$$

This array also serves to determine the set of time slots where the power constraint is violated (Algorithm 3.7): T^- contains time slots where the difference is *negative*.

The sorting step (*) in line 4 is optional, and is given as a parameter. The sort arranges the instances of $\mathcal{I}_a$ in the non-increasing order of their profits, so that those of maximum profit value are considered first. Otherwise, the list of charging plan instances is shuffled before trial.

Next we discuss the essential methods the algorithm is using:

calculatePwrDiffs calculates the difference between available power and power consumed per time slot. This approach is also used when determining the conflicting time slots in Algorithm 3.7 (those with negative values comprise the list T^-).

updatePwrDiff is a rapid way of determining the potential local improvement that the given instance i' offers, i.e. it serves to *evaluate the fitness of the neighbors*. We decrease the power consumed in the time slots where i is active, and increase the power consumed in those of the newly assigned instance i', by the amounts of their respective charging rates. The power consumption update is visualized in Figure 3.2.

sumNegatives serves as a fitness calculation here. It sums up only the negative values in the array of power differences given as its parameter. This is a swift method of determining the local fitness improvement. It is possible to apply this since all the charging plan instances of other activities remain fixed while we test the current activity.

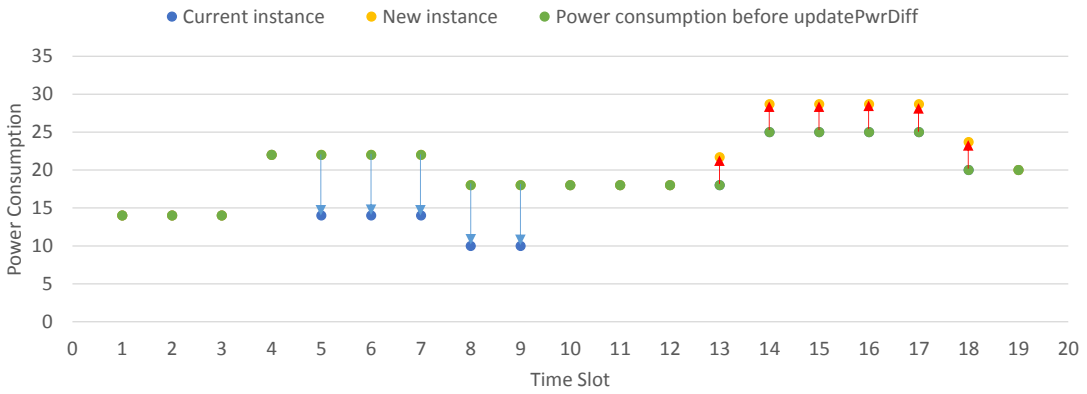


Figure 3.2: Power consumption update (*updatePwrDiff* method). Consumption in the time slots of the current instance (i) is decreased, in those of the new instance (i') increased.

If the sum of the negative values in array $\Delta P'$, induced by i', is greater (i.e. less negative) than the previous best; or, alternatively, the two are equal but the profit of the observed instance i' is higher than the current one i, i' is assigned as the new best instance i^*. Finally, the overall best instance (implying least negative sum of negative values in ΔP) is returned.

Profit Maximization by Sorting the Charging Plan Instances

The min-conflicts heuristic (MCH) has proven very useful for constraint satisfaction problems. The constraint we are dealing with, once the first subproblem of assigning the activities to ma-

chines is solved, is the available power. Since our approach is based on MCH (cf. Algorithms 3.7 and 3.8), it is more akin to constraint *satisfaction* than it is to constraint *optimization*. This entails that our algorithm **stops when no more conflicts** are present (line 9-11).

In an attempt to *maximize the induced profit* and mimic the constraint *optimization*, we try to assign individual charging plan instances of highest profit by sorting the instances at two occasions:

- when assigning the initial charging plans
- before trialling them for best improvement (cf. line 3 in Algorithm 3.8)

The two strategies we have implemented for assigning the initial charging plans are:

a) **Random**: among all charging plan instances, assign a random one

b) **Max-profit-first**: sort all instances by profit and assign the maximum profit one

As far as sorting in the *Best Improvement* algorithm (Algorithm 3.8) is concerned, we either **shuffle** or **sort** (in the order of decreasing profit) the charging plan instances before trialling them.

We will see how the different combinations of instance choosing strategies affect the scheduling results in Chapter 4.

3.4 Steady-state to Dynamic Transition

The scheduling heuristics were intended to be employed within a wider framework of a dynamic, online reservation system, mimicked by a simulator developed under the KOFLA project. Therefore we have integrated them in a *charging station (CS) controller* which can react to various *events* generated either from the customers (their EVs) or from the electric grid distribution system, mediated by the KOFLA system. The online scheduling is *event-driven*, and not run periodically (e.g. every 15 minutes).

Main differences to the static version of the problem are:

1. **Time transformation**: We still deal with this dynamic variant of the problem by discretizing the time, but we need to standardize the way of translating real time (e.g. 8:27 a.m.) into discrete intervals.
2. **Event handling**: The system now responds to various events, such as reservations, arrivals within or outside of the planned parking period, leave and timeout events. This is when rescheduling occurs.
3. **Rolling schedule**: Simulating the flow of real time, the schedule now ”rolls” over the time slots and keeps track of the NOW marker, which corresponds to the ”current” time.
4. **Reduced number of activities**: The scheduler now deals only with a subset of all the activities — those within the planning time horizon $\mathcal{T}$. In general, this reduces the total number of instances, and makes room for some potential optimizations in the algorithms.

5. **Possible rescheduling**: The same activity (that entered the system) can be rescheduled several times, as long as its assigned charging plan has not yet commenced execution. This gives room for some performance optimization, in terms of *caching*.
6. **Predictions errors**: Real world means things are constantly changing (e.g. arrival and departure times can turn to be different than predicted, the energy demand can change in the meantime), and the online system has to keep up with the pace and deal with the predictions, not factual data, efficiently. The *available power* is predicted as well, and it may take different values during the charging periods (the power *curve*, again discretized). This makes the problem even more difficult to solve.

The time transformation, various handled events, the rolling schedule and finally the controller providing all of these functions to the system, along with denial-of-service notifications, are described next.

3.4.1 Time Transformation

Since the heuristics operate on a discrete list of time slots ($\mathcal{T} = \{0, \ldots, T-1\}$), we need to transform the real time points into appropriate discrete representation. This is done in two stages:

1. *rounding* the *time points* onto 15-minute *timestamps*
2. *mapping* the 15-minute *timestamps* into discrete *time slots*

The event time points in real time are first converted by the following rounding principle into their corresponding parameters, to the nearest "round" 15-minute block:

Event	Real Time	Parameter	Time Slot
arrival	7:27:17	*earliest*	7:30:00
departure	9:37:43	*latest*	9:30:00

Table 3.1: Time Rounding Principle

When calculating the *earliest* (e_a) we round "up", and when calculating the *latest* (l_a) we round the time "down" to the nearest 15-minute block time point.

The **time slot** identified by a **timestamp** determines a 15-minute block starting with this timestamp, e.g. 7:30 identifies the time slot (interval) 7:30 - 7:45.

Here again, the time slot corresponding to *earliest* is inclusive, and the one corresponding to *latest* is exclusive to the parking period. If we follow the example from Table 3.1, the arrival and departure times transform into the parking period $[7{:}30, 9{:}30)$, within which we can plan the charging. This rounding is necessary as we operate with discrete blocks of time. We need *entire* intervals to be available for scheduling, and this rounding principle provides us with exactly that.

To translate these rounded timestamps into integer-valued blocks of time, i.e. time intervals $(0, \ldots, T-1)$, we consider the NOW marker to correspond to the beginning of the time slot $t = 0$, and translate every subsequent 15-minute interval accordingly.

As an example, if NOW is mapped to 7:30, time slot $t = 0$ would correspond to the period $[7{:}30, 7{:}45)$, $t = 1$ to $[7{:}45, 8{:}00)$, etc.

Once this transformation has been performed, the data is ready to be forwarded to the scheduling heuristics.

3.4.2 Events

The events which can occur in our system, to which the controller corresponds appropriately, are the following:

reserve When the owner of an EV wants to make a reservation for a parking place (i.e. the charging machine), the KOFLA system provides the following data to the CS: the expected arrival and departure times, the energy demand and the vehicle's reservation ID. The controller checks whether there is enough resources (parking and power) for this vehicle and notifies the system by confirming or cancelling the reservation.

plugin The EV arrives to the charging station and connects to the machine. If necessary, the actual energy demand is updated, as well as the parameters of the actual parking period (if they do not correspond to the planned ones). In case there was no previous reservation, the service for this vehicle is denied.

unplug The EV unplugs from the machine and leaves the charging station (possibly even before it was planned). The resources it was occupying are released as necessary, and the data about the activity and its charging plan are updated.

timeout When the EV does not arrive on time (including the predetermined *slack* period around the planned arrival), the timeout event notifies the controller to remove the reservation from the system and release its assigned resources.

available power update Initiated by the externally imposed electrical grid, the available power curve for the planning time horizon is regularly updated, and rescheduling is performed when necessary.

More details on the steps taken at the occurrence of each of these events are to follow in Subsection 3.4.4.

3.4.3 Rolling Schedule

As mentioned, the schedule is "rolling" over the time slots, and does so at the occurence of the described events. Every event arrives with an assigned timestamp, and at each event the NOW marker is updated. This means we do not have to handle each and every time slot, but we can skip to only those of relevance — when actual events occur. As the NOW marker shifts, we also need to shift the power load (consumed power) and the available power curves (arrays) accordingly, so they represent the "actual" state, and keep the time parameters of the activities up to date.

The events can induce repeated scheduling of the activities currently present in the planning time horizon. *Rescheduling* is performed upon *reserve*, *plugin* and *available power update*

events, as they have a potential impact on the previously valid schedule. We maintain the "actual" schedule and try to add new activities as they arrive. If the new conditions — be it the new activity at *reserve* or *plugin*; or new available power curve at *available power update* — violate the power constraint, the optimization steps are taken. If the scheduling fails, the system is notified (and the new activity is rejected).

On the other hand, *releasing the resources* — reducing the power load, freeing up the machines (parking space) and removing the activities from the "active" ones — is performed at *unplug* and *timeout* events. At *unplug* event the corresponding activity is put into the set of "completed" activities.

Temporary Earliest

One important modification, in relation to the static variant of the problem, is the possibility of rescheduling a certain activity multiple times, as long as its charging has not already started. For this purpose we introduce an extension to the EV model in the manner of a *temporary earliest* (e'_a) variable. When any of the events occur, the NOW marker gets updated. If it enters the scheduled activity's parking period, but does not intersect the assigned charging plan instance's period, we allow for this activity's charging plan recalculation by assigning e'_a to the value of NOW. Figure 3.3 shows an example of activities excluded from rescheduling (their charging plans in red) at $t = 5$, since the NOW marker intersects their charging period at that time slot.

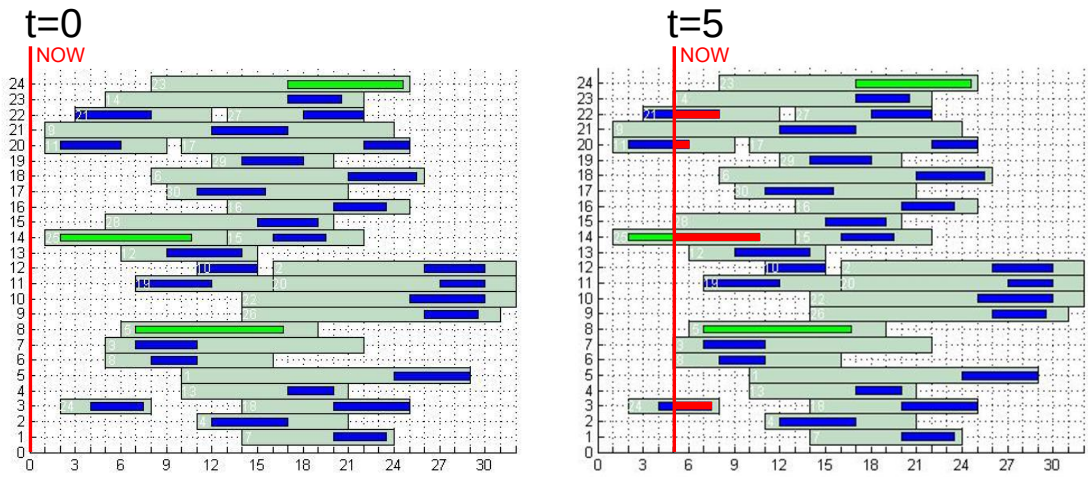


Figure 3.3: Example of the rolling schedule; NOW marker intersecting activities not allowed for rescheduling (charging plans in red)

The mapping of the time points into the discrete intervals for the scheduling heuristics is then based on this temporary value e'_a, instead of the "original" *earliest* (e_a). This allows for an *online adaptation* in case of early or late arrivals and departures.

Caching the Charging Plan Instances

In order to escape the unnecessary repeated generation of the charging plan instances as rescheduling occurs, we use a map to cache the generated instance sets for every activity in the system. The sets are recalculated only when some of the activity's attributes — the energy demand or the (temporary) parking time window — have changed. This can happen in two cases:

a) At plugin, the rounded timestamp can slightly differ from that of predicted arrival time at reservation. In this case, we update e_a (and e'_a) of the activity to the time slot corresponding to NOW. The energy demand (d_a or $d_{min,a}, d_{max,a}$) can change at plugin event as well, when the actual demand differs from the one at the reservation. This case affects only the activity currently plugging in.

b) At any event's occurence, the rolling schedule's NOW marker can intersect the activity's parking time window. If it does not intersect the charging period ($e_{a_i} \leq$ NOW $\leq s_i$), we update e'_{a_i} to NOW. This case affects all of the activities present in the planning time horizon whose charging has not commenced yet.

These modifications of $e'_a(e_a)$ and d_a (or $d_{min,a}, d_{max,a}$) result in recomputation of the activity's charging plan instances set as we prepare it for scheduling. We use the previously generated set if these attributes have not been changed. This subtle improvement is shown in Algorithm 3.9, presenting a snippet of the modified instance generation module.

Algorithm 3.9: Cached Instance Generation

Input: Activity a
Output: Set of valid charging plan instances for the activity a

```
1 if M.containsKey(a) and neither e'_a nor d_a have changed then   // or d_min,a, d_max,a
2 |   return M.get(a)
3 end
4 I_a ← ∅
  // generate the instances as usual
5 return I_a
```

3.4.4 The Controller

Here we describe the charging station (CS) controller, i.e. the interface to the KOFLA simulator. It acts as a wrapper for the scheduling heuristic algorithms, performing the necessary time transformations, handling the events, exchange of data and signals between the CS and the system.

The controller maintains the current charging schedule, information about the activities (those to schedule as well as completed ones), and the load profile of the charging station, which provides insight into the power consumed at each time slot.

The controller is able to notify the system about the unavailability of resources when trying to schedule a new activity or reschedule in new conditions. These signals are:

No parking space available [NPSA]: When there is no room for the new activity at the charging station, i.e. no machine is available for the given parking period.

No power available [NPA]: When the new activity can not be assigned a charging plan, i.e. there is not enough power to include it in the current schedule (or the new available power curve is violated).

Parking duration too short [PDTS]: When there is no feasible charging plan instance for the given parameters (parking time window and energy demand).

Next we have a look at the methods comprising the controller.

Reserve

At reservation, we need to make sure the charging station has enough resources to accommodate the EV, by attempting to include the new activity into the schedule. If so, the charging schedule is updated and the new load profile of the CS is returned to the KOFLA system. (cf. Algorithm 3.10)

Algorithm 3.10: Reserve Procedure

Input: Reservation ID, expected arrival and departure, power demand(s), current time
Output: Load profile

1 update NOW from current time
2 shift the power load and available power arrays correspondingly
3 $a \leftarrow$ a new activity generated from the input data
4 try to assign a to a machine, otherwise send [NPSA]
5 transform the activities' timestamps into discrete time slots, according to NOW
6 try to reschedule including a, otherwise send [NPA] or [PDTS]
7 **return** load profile

Plugin

The EV arrives at the charging station and attempts to plug into the machine assigned to its reservation ID. If there has been no previous reservation for this EV, it gets rejected. When the EV scheduled at the same machine as the one plugging in (scheduled before the current one) has not left yet, the controller attempts to assign it a new charging point. If it fails to satisfy the requirements by the currently available resources, appropriate signals are emitted and the activity is not included in the schedule. (cf. Algorithm 3.11)

Unplug

The EV unplugs and the resources it was assigned get released as necessary. If the vehicle left too early and the charging was interrupted, the power occupied by its charging plan instance needs to be released, so it does not falsely interfere with the scheduling. (cf. Algorithm 3.12)

Algorithm 3.11: Plugin Procedure

Input: Reservation ID, actual power demand(s), current time
Output: Load profile

if *reservation ID not present* **then**
  reject the EV and notify the system
end
update NOW from current time
shift the power load and available power arrays correspondingly
update e_a from current time
if d_a *has changed* **then** // or $d_{min,a}, d_{max,a}$
  update d_a
end
if *the machine assigned at reservation is not free* **then**
  try to assign a to a new machine, otherwise send [NPSA]
end
transform the activities' timestamps into discrete time slots, according to NOW
try to reschedule including a, otherwise send [NPA] or [PDTS]
return load profile

Algorithm 3.12: Unplug Procedure

Input: Reservation ID, current time
Output: Load profile

update NOW from current time
remove a from the set of *active* activities
remove a from its machine's schedule
update l_a from current time
$i \leftarrow a$'s instance
if *charging was interrupted* **then**
  release the power load of i
  raise the i's interruption flag and update f_i
end
put a into the set of *completed* activities
shift the power load and available power arrays correspondingly
return load profile

Timeout

In the event of a timeout (EV did not show up on time), the EV's activity needs to be removed from the planned schedule. This will cancel the reservation and release the associated charging point as well as the reserved amount of available power. (cf. Algorithm 3.13)

Algorithm 3.13: Timeout Procedure

Input: Reservation ID, current time
Output: Load profile

1 update NOW from current time
2 remove a from the set of *active* activities
3 remove a from its machine's schedule
4 release the power load of a's instance
5 shift the power load and available power arrays correspondingly
6 **return** load profile

Available Power Update

As time flows, and the station maintains predicted data about the next 24 hours, the available power curve needs to be refreshed periodically. Also, in case of a predicted change the charging station needs to be notified. This event is initiated by the electric grid, and the controller updates the predicted available power curve of the CS for the planned time horizon. Rescheduling needs to be performed, as the previous schedule might not be in line with the new power constraints. (cf. Algorithm 3.14)

Algorithm 3.14: Available Power Update Procedure

Input: Reservation ID, current time
Output: Load profile

1 update NOW from current time
2 update the available power array
3 transform the activities' timestamps into discrete time slots, according to NOW
4 try to reschedule, otherwise send [NPA]
5 **return** load profile

CHAPTER 4

Experimental Results and Analysis

To study the behaviour of the three developed heuristic algorithms, we have run multiple experiments to compare them mutually, as well as line them up with the optimal solutions achieved by the CPLEX solver (for its feasible problem instances) on the problem's MIP formulation [7], [9].

In the first section we describe the experimental setting — the hardware on which the experiments were run, the evaluation criteria, and the problem instances we have used to measure the performance of the heuristics.

Next, we deal with tuning the min-conflicts heuristic's (MCH) parameters on the *offline*, *SingleDemand* model, to reach its peak performance settings. Here we regard the random noise probability and various sorting strategies when assigning the charging plan instances. We compare MCH results with those of the greedy heuristic (GH), the local ratio technique (LRT) and the CPLEX solver.

Finally, we use the *MinMax* vehicle model, which is more flexible and closer to the real world setting, to provide experimental results for the system's offline and online modes.

Parts of results presented here were published in a MISTA conference paper [9].

4.1 Experimental Setting

In this section we will describe the problem instances we used, along with the evaluation function to assess the performance of the selected solution techniques.

The algorithms were implemented using the Java technology, and all experiments (with the exception of the CPLEX runs) were performed on a personal laptop with the following specifications: Intel Core2 Duo P8400 @ 2.26 GHz (3MB Cache), 4GB RAM Dual-Channel DDR2 @ 398MHz, 500GB HDD @ 7200rpm (SATA-III 6.0Gb/s).

4.1.1 Problem Instances

Since the EVRSTW problem as we defined it is quite novel — incorporating time windows combined with different charging rates and charging completion degrees — we generated a

synthetic set of problem instances.

The problem instances, denoted with $[A]$-$[M]$-$[P]$, differ in:

- A — the number of vehicles (*activities*) whose energy demand is to be satisfied
- M — the number of charging slots (*machines*) which can charge at different rates
- P — the available power limit

The power limit P is here set to a constant value, but the algorithms can work with a variable available power curve as well. The planning time horizon $\mathcal{T}$ was set to 8 hours, translating into 32 discrete 15-minute time slots to work with.

A single problem instance contains the list of activities, with each having assigned:

- *earliest* (arrival time)
- *latest* (departure time)
- *energy demand* (single value for the SingleDemand model; minimum and maximum value for the MinMax model)

These values were generated using the uniform distribution $U(a, b)$, in the following manner:

- $e_a = U(0, 20)$
- $l_a = e_a + U(4, 10)$
- d_a or $d_{max,a} = U(6, 10)$ (depending on the vehicle model)
- $d_{min,a} = \lfloor \frac{d_{max,a}}{2} \rfloor$ (in the case of MinMax model)

Given such a list of activities, the number of machines and the available power, the scheduler has to determine for each activity its charging plan instance, i.e.:

- the *machine* to which the vehicle will be connected,
- the *start time* of charging,
- the *charging rate* (which together with *start time* determines the *duration*, and thus *finish time* of charging),
- its *completion degree*, and
- its *profit*

All problem instances are created to be *feasible*, i.e. there are enough machines available, and each activity can be charged up to at least the minimum requested energy amount ($\frac{1}{2}d_a$ for the SingleDemand model; $d_{min,a}$ for the MinMax model). The problem instances were modelled to simulate a *moderate load*, i.e. there exist several vehicles that can not be charged completely, given the available power.

The described problem instances were used in the *offline* (steady-state) mode, i.e. all of the activities were fed into the algorithm at once, making the system have complete knowledge about the entire problem at hand. The schedule is then built ”from scratch”.

4.1.2 Performance Evaluation

As stated earlier (Eq. 2.7), we will measure the *total profit* attained by each of the algorithms, using the following evaluation function:

$$Z = \sum_{i \in \mathcal{I}} p_i x_i \tag{4.1}$$

This translates to simply summarizing the profits of the selected charging plan instances assigned to the activities.

Next, we measure the *runtimes* of the algorithm runs, as this aspect is quite important for a real-time scheduler — the customers expect almost instant response from the routing, reservation and scheduling system.

On top of that, we will register the *number of unscheduled activities*, as our goal is to accomodate all of the customers' requests. This will show to be a valuable indicator of a good solution (e.g. the greedy heuristic and the local ratio technique do not manage to schedule all of the given activities). As stated previously, we call this a *fair ruler policy*, meaning our goal is to rather charge more vehicles with less total energy, instead of charging less cars with more total energy.

We let MCH run for 100 times on each problem instance (since the method is non-deterministic), with the time limit of 10 minutes, and collected average, maximum and minimum values of total profit, and average values of the runtime. For other algorithms (GH, LRT, CPLEX), single values of total profit and runtime are collected from a single run (since those are deterministic methods).

4.2 Parameter Tuning for the Min-conflicts Heuristic

For this purpose we have employed the algorithm variants in the *offline* mode, on the *SingleDemand* vehicle model.

Regarding the profit calculation of the single instance (Eq. 2.1), we have set $k = 2$ and $\alpha = 0.5$, which renders the equation as follows:

$$p_i = \frac{1}{2}\, c_i + (1 - \frac{1}{2})\, \frac{2}{r_i} = \frac{1}{2}\, c_i + \frac{1}{r_i} \tag{4.2}$$

The value of α was chosen according to the experimental results achieved by the CPLEX solver in [7], to make them comparable with those of our heuristics'.

4.2.1 Parameters of the Min-conflicts Heuristic

We varied two parameters of the min-conflicts heuristic:

1. *Random noise probability* p_{noise} — As shown in Algorithm 3.7, in $[p_{noise}]$ percent of cases we choose a random charging plan instance for the currently observed activity.

2. *Sorting strategies for charging plan instances selection* — In subsection 3.3.3 we described the usage of sorting strategies in an attempt to maximize the total profit, i.e. to optimize, not just to satisfy the constraints.

We used the problem instances ranging from 20 to 200 vehicles, and varied:

- p_{noise} in the range $\{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$
- 3 sorting strategy variants:
  - *random* — the initial instances are chosen at random; instances are shuffled before trialling them when searching for best improvement
  - *sortInit* — the initial instances are chosen in the order of the highest profit from the set of sorted feasible instances; instances are shuffled before trialling them for best improvement
  - *sortInitSel* — sorting in the order of highest profit first, both when assigning the initial instances and when trialling for the best improvement

Random Noise Probability

We have found that setting $p_{noise} = 0$, i.e. including no random noise at all, does not work even for the smallest problem instances ($A = 20$) — the algorithm does finish in some of the runs, but not all of them.

At the other extreme, setting $p_{noise} = 1$, i.e. choosing the charging plan instances completely at random, shows to be infeasible for problem instances with $A >= 60$.

Finally, when $p_{noise} = 0.9$ the problem instances having $A >= 80$ could not be solved in all of the runs within the set time limit.

We therefore discarded these results and here present only those for p_{noise} in the subset $\{0.1, 0.3, 0.5, 0.7\}$. Table 4.1 shows how p_{noise} affects the average total profit values when we average the results over the three sorting strategies (*rand*, *sortInit* and *sortInitSel*). Same results are visualized in Figure 4.1.

Table 4.1: Average total profit for various p_{noise} values; *SingleDemand* model, $\alpha = 0.5$

Problem Instance	0.1	0.3	0.5	0.7
20-16-30	**10.747**	10.416	10.163	9.789
40-32-40	**17.128**	16.652	16.500	16.352
60-40-55	**24.679**	24.128	23.977	23.807
80-48-70	**32.567**	31.975	31.731	31.650
100-56-90	**41.490**	40.447	40.144	39.946
120-64-100	**48.480**	47.634	47.313	47.171
160-80-150	**71.014**	68.594	67.051	66.441
200-100-180	**87.295**	84.548	83.012	82.658

Table 4.2 presents the average runtimes for different probability values.

We observe that $p_{noise} = 0.1$ yields best overall results, incurring highest average total profit and low enough average runtimes (less than 100ms over all problem instances). We can notice

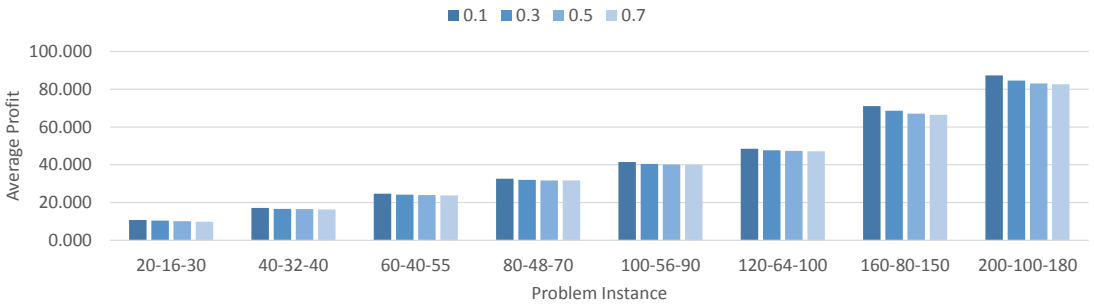


Figure 4.1: Average total profit for various p_{noise} values; *SingleDemand* model, $\alpha = 0.5$

Table 4.2: Average runtimes (ms) for various p_{noise} values; *SingleDemand* model, $\alpha = 0.5$

Problem Instance	0.1	0.3	0.5	0.7
20-16-30	44.610	25.717	16.470	16.697
40-32-40	46.477	23.103	25.017	34.700
60-40-55	59.293	36.267	52.600	215.417
80-48-70	81.137	54.150	153.263	2484.227
100-56-90	62.607	48.953	99.247	628.617
120-64-100	93.743	119.583	494.340	36198.300
160-80-150	60.337	49.653	70.727	177.183
200-100-180	94.697	71.273	115.393	520.517

that setting p_{noise} to higher values significantly increases the runtime (e.g. the runtime ascends to 36s for $p_{noise} = 0.7$ and instance 120-64-100).

Charging Plan Instances Sorting Strategy

Similar to trialling the random noise probabilities, we have recorded the average total profits and runtimes for different sorting strategies, while averaging the results over the p_{noise} values.

Table 4.3 shows how the three sorting strategies affect the average total profit values, with results averaged over four p_{noise} values, $\{0.1, 0.3, 0.5, 0.7\}$. Same results are visualized in Figure 4.2.

Table 4.3: Average total profit for the three sorting strategies; *SingleDemand* model, $\alpha = 0.5$

Problem Instance	*rand*	*sortInit*	*sortInitSel*
20-16-30	9.121	10.785	**10.931**
40-32-40	16.367	16.694	**16.913**
60-40-55	23.925	24.141	**24.377**
80-48-70	31.784	31.859	**32.299**
100-56-90	40.143	40.350	**41.028**
120-64-100	47.297	47.535	**48.117**
160-80-150	66.555	68.498	**69.772**
200-100-180	82.676	84.405	**86.054**

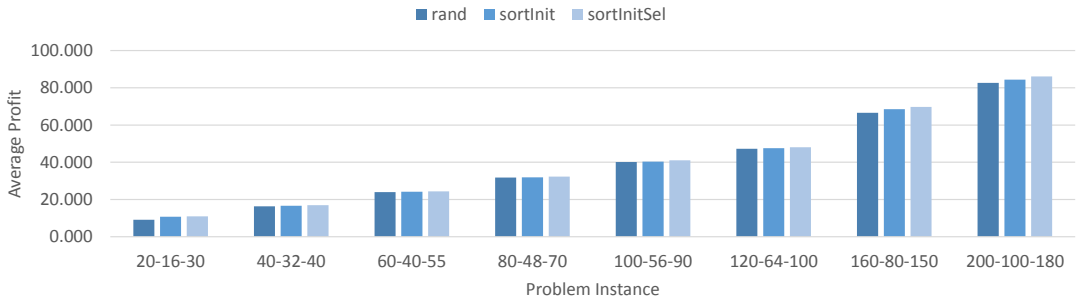


Figure 4.2: Average total profit for the three sorting strategies; *SingleDemand* model, $\alpha = 0.5$

We can observe that the sorting strategy *sortInitSel* yields the overall best average total profit.

Concerning the runtimes, there are no significant trends regarding the sorting strategies to be noted — there is no obvious "winner". The higher random noise probabilities ($p_{noise} = \{0.5, 0.7\}$) skew the results averaged over all p_{noise} values, so we show just the runtimes averaged over $p_{noise} = \{0.1, 0.3\}$ in Table 4.4. The average runtimes for these values are kept low, under 120ms, regardless of the sorting strategy.

Table 4.4: Average runtimes (ms) of the three sorting strategies (averaged over $p_{noise} = \{0.1, 0.3\}$); *SingleDemand* model, $\alpha = 0.5$

Problem Instance	*rand*	*sortInit*	*sortInitSel*
20-16-30	24.110	48.885	32.495
40-32-40	28.815	29.855	45.700
60-40-55	41.580	44.140	57.620
80-48-70	74.980	68.360	59.590
100-56-90	61.030	57.125	49.185
120-64-100	118.925	87.080	113.985
160-80-150	53.960	54.410	56.615
200-100-180	102.870	73.175	72.910

By aggregating the results from the analysis of the two MCH parameters, we can conclude that the best performing MCH variant is the one with **random noise probability** $p_{noise} = 0.1$ and the **sorting strategy** set to *sortInitSel*.

4.2.2 Comparison with Other Techniques

Here we compare the best performing MCH variant ($p_{noise} = 0.1$, *sortInitSel*) to the other solution methods — greedy heuristic (GH), local ratio technique (LRT) and the CPLEX solver (exact method working on the MIP formulation). We are keeping the system in the *offline* mode with the *SingleDemand* vehicle model, and set $\alpha = 0.5$ (to be comparable to CPLEX results from [7]).

Total profit achieved by the four methods (average total profit for MCH) is presented in Table 4.5 and Figure 4.3.

Table 4.5: Total profit values across all problem instances; *SingleDemand* model, $\alpha = 0.5$

Problem Instance	*GH*	*LRT*	*MCH* (Avg)	*CPLEX*
20-16-30	12.760	12.736	11.642	13.526
40-32-40	17.922	18.048	17.569	23.135
60-40-55	26.061	25.424	25.081	32.968
80-48-70	34.054	34.060	33.148	44.029
100-56-90	44.399	41.591	42.412	51.420
120-64-100	52.122	49.374	49.234	61.979
160-80-150	75.622	73.318	73.714	93.962
200-100-180	93.959	87.836	90.258	116.065

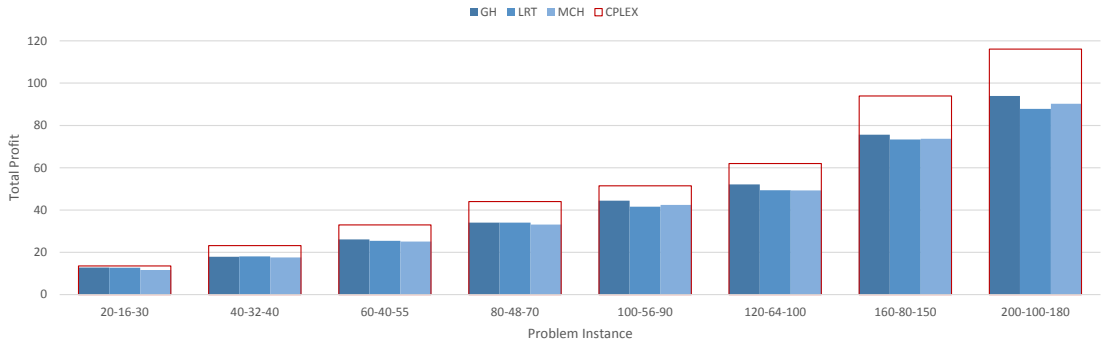


Figure 4.3: Total profit values; *SingleDemand* model, $\alpha = 0.5$

Even though it may seem that GH and LRT perform better than MCH, we must recall another important evaluation criterion we established. While MCH manages to schedule *all* of the given activities, GH and LRT do not succeed to accommodate all of the requests. Table 4.6 shows the number of unscheduled activities for the respective solution methods.

Table 4.6: Number of unscheduled activities; *SingleDemand* model, $\alpha = 0.5$

Problem Instance	*GH*	*LRT*	*MCH*
20-16-30	1	0	0
40-32-40	13	7	0
60-40-55	20	12	0
80-48-70	28	13	0
100-56-90	31	18	0
120-64-100	40	20	0
160-80-150	43	14	0
200-100-180	54	26	0

A smaller part (10-15%) of the unscheduled activities of GH belongs to the too short duration of the parking period, when there is not enough time to fulfill even 50% of the energy demand with the charging rate of 3.7kW. But most of the activities are rejected because of the greedy instances making the consumed power shoot over the available power when trying to add them into the schedule.

Yet another perspective on the quality of the solution method is its runtime. Figure 4.4 shows the runtimes of the three implemented heuristics (average value for MCH).

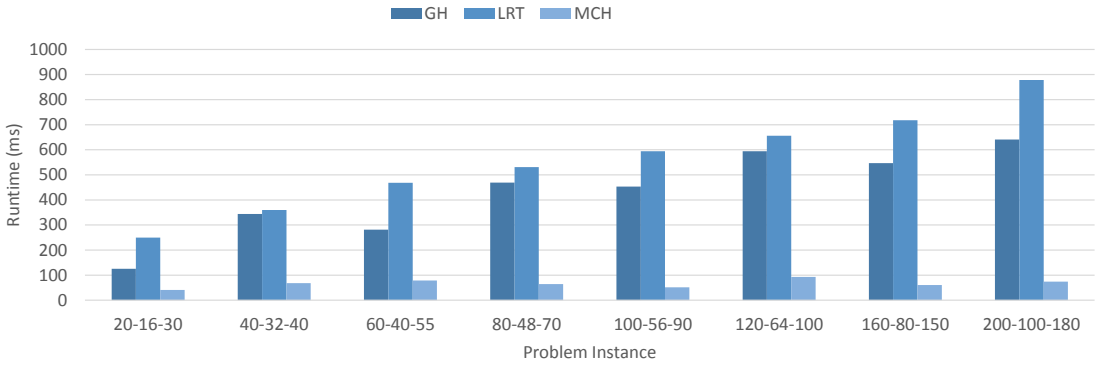


Figure 4.4: Runtime (ms) of the three heuristic methods; *SingleDemand* model, $\alpha = 0.5$

We can observe that the runtimes of MCH are considerably lower (kept under 100ms) than those of the two other heuristics, further adding to the conclusion that MCH is the best-performing method under our evaluation criteria. Its runtimes are not significantly impacted by the size of the problem instances.

We additionally provide the ratio of average and maximum total profit accomplished by MCH compared to the optimal solution (achieved by CPLEX) in Table 4.7. The average profit achieved by MCH is above 75%, and the maximum profit at least 79% of the optimal value (Z_{CPLEX}). The issue with CPLEX (the exact method) approach, however, is the running time for larger instances, which will be further examined in the following section.

Table 4.7: Total profit values of MCH and CPLEX; *SingleDemand* model, $\alpha = 0.5$

	MCH					
Problem Instance	*Min*	*Avg*	*Max*	*CPLEX*	$\frac{Z_{MCH,Avg}}{Z_{CPLEX}}$	$\frac{Z_{MCH,Max}}{Z_{CPLEX}}$
20-16-30	10.286	11.642	12.570	13.526	**0.861**	**0.929**
40-32-40	16.351	17.569	18.737	23.135	0.759	0.810
60-40-55	23.231	25.081	26.462	32.968	0.761	0.803
80-48-70	31.567	33.148	34.802	44.029	**0.753**	**0.790**
100-56-90	40.771	42.412	44.053	51.420	0.825	0.857
120-64-100	46.622	49.234	51.891	61.979	0.794	0.837
160-80-150	71.413	73.714	75.623	93.962	0.785	0.805
200-100-180	87.057	90.258	93.188	116.065	0.778	0.803

4.3 Testing the MinMax Model

This setting should offer more flexibility, since more completion degrees c_i are allowed (instead of just 50%, 75% and 100% with the *SingleDemand* model). On the other hand, allowing more charging plan instances of different completion degrees for a single activity extends the available search space and makes the problem more complex, but also allows for finer-grain scheduling.

When tuning the function p_i (Eq. 2.1) for this purpose, we wanted to assign most of the weight to the charging completion degree, and allow the lower charging rate preference to only slightly affect the solution. This setting is also in line with [9], and is necessary to make the results comparable to those of CPLEX. Thus, we assigned $k = 2$ and $\alpha = 0.95$, which renders the equation as follows:

$$p_i = 0.95\ c_i + (1 - 0.95)\ \frac{2}{r_i} = 0.95\ c_i + \frac{0.1}{r_i} \tag{4.3}$$

4.3.1 Offline (Steady-state) Mode

Recall that the system can work with variable available power constraint (power curve), but for the offline mode we have once again run the experiments with the fixed value of available power.

We compared the results of the three heuristics (MCH, GH, LRT) with the exact solutions of the CPLEX mixed integer programming model (with the stop condition set at an optimality gap of 1%). As for the problem instances, we ran the test on those of size 20 to 120 activities, generated by rules defined in section 4.1.1.

To show once again that the previously selected best-performing variant of the MCH is the one with *sortInitSel* sorting strategy, we set $p_{noise} = 0.1$ and present the average total profit values achieved by the three sorting strategies — *rand, sortInit, sortInitSel* (Figure 4.5). The runtimes are kept low — under 250ms for all three approaches over all the problem instances.

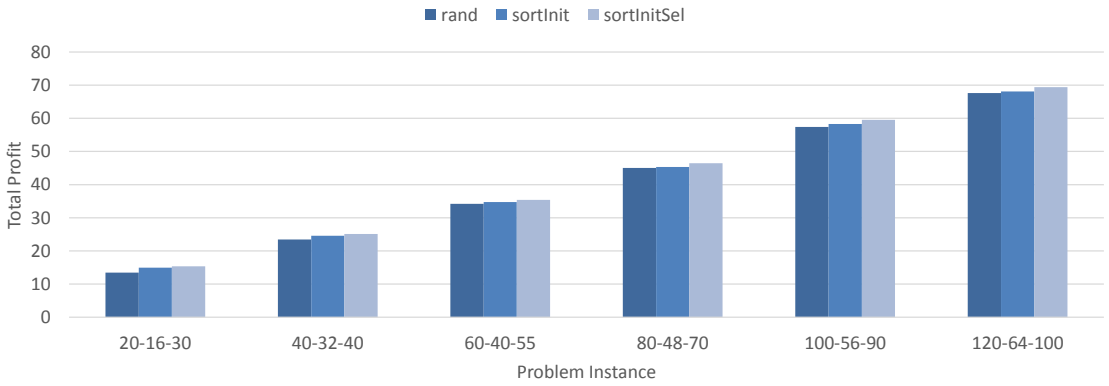


Figure 4.5: Average total profit for various MCH sorting strategies; *MinMax* model, $\alpha = 0.95$

Next, we line up MCH against greedy heuristic (GH), local ratio technique (LRT) and the CPLEX solver. Total profit values achieved by the four solution methods are presented in Table 4.8 and Figure 4.6. Once again, the values for MCH are averaged (over 100 runs).

Table 4.8: Total profit values; *MinMax* model, $\alpha = 0.95$

Problem Instance	*GH*	*LRT*	*MCH (Avg)*	*CPLEX*
20-16-30	16.233	15.166	15.343	19.200
40-32-40	27.671	23.672	25.113	29.600
60-40-55	40.849	34.159	35.395	41.480
80-48-70	53.825	45.780	46.425	53.470
100-56-90	67.677	59.815	59.542	67.930
120-64-100	81.953	69.092	69.433	78.520

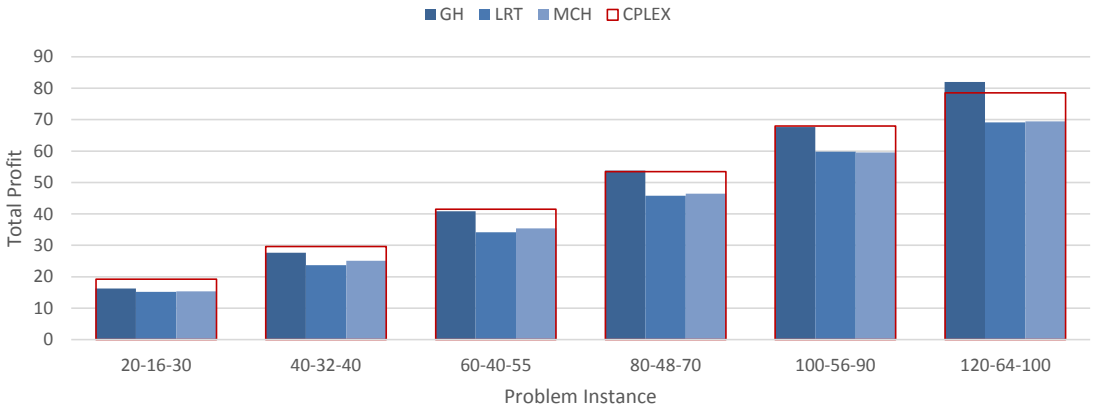


Figure 4.6: Total profit values across all problem instances; *MinMax* model, $\alpha = 0.95$

We can notice that GH profits are unexpectedly high, reaching even over the optimal CPLEX results. The reason for that is, once again, the above-zero number of unscheduled activities, which are rejected to make the solutions of GH and LRT feasible (Table 4.9). LRT performs nearly as well as MCH on the profit scale, but the solutions contain quite a high number of unscheduled activities (about 30% for larger problem instances).

Table 4.9: Number of unscheduled activities; *MinMax* model, $\alpha = 0.95$

Problem Instance	*GH*	*LRT*	*MCH*
20-16-30	1	2	0
40-32-40	14	11	0
60-40-55	22	18	0
80-48-70	30	25	0
100-56-90	35	30	0
120-64-100	45	39	0

Here again, the reason for about 10% of the unscheduled activities of GH is the too short parking period ($d_{min,a}$ can not be satisfied with $r = 3.7$kW), while the rest is due to not enough available power.

Comparing the runtimes of the algorithms, we can see that MCH dominates (Table 4.10), keeping the average runtime below 160ms even for the largest tested problem instance.

Table 4.10: Runtime (s) of the four solution methods; *MinMax* model, $\alpha = 0.95$

Problem Instance	*GH*	*LRT*	*MCH (Avg)*	*CPLEX*
20-16-30	0.219	0.158	0.080	24
40-32-40	0.343	0.343	0.077	26
60-40-55	0.453	0.358	0.103	192
80-48-70	0.859	0.467	0.101	527
100-56-90	0.719	0.512	0.100	1500+
120-64-100	0.735	0.735	0.156	468

Finally, let us see how close MCH gets to the optimal CPLEX solutions in Table 4.11.

Table 4.11: Total profit values of MCH and CPLEX; *MinMax* model, $\alpha = 0.95$

	MCH					
Problem Instance	*Min*	*Avg*	*Max*	*CPLEX*	$\frac{Z_{MCH,Avg}}{Z_{CPLEX}}$	$\frac{Z_{MCH,Max}}{Z_{CPLEX}}$
20-16-30	13.787	15.343	16.787	19.200	**0.799**	**0.874**
40-32-40	23.611	25.113	26.407	29.600	0.848	0.892
60-40-55	33.517	35.395	36.610	41.480	0.853	0.883
80-48-70	44.752	46.425	47.941	53.470	0.868	0.897
100-56-90	57.759	59.542	60.970	67.930	0.877	0.898
120-64-100	67.920	69.433	71.198	78.520	**0.884**	**0.907**

We can conclude that MCH is the absolute winner among the three tested heuristic methods, with the main advantage of *feasibility* — it managed to schedule **all** of the activities provided and satisfy at least their minimum demands. At the same time, the runtime is kept fairly low for even the greatest problem instances (under 160ms), and it achieves the total profit (objective/fitness value) of **80%** (average profit) and even **up to 90%** (maximum profit) **of the optimal solution**, according to our evaluation function 4.1.

The use of CPLEX, even though always achieving optimal results while scheduling all of the activities, has the disadvantage of a *too long running time* (especially for our problem instance 100-56-90). This can not be tolerated in a real world scheduling environment as we described it, since the KOFLA customers expect almost immediate response (in terms of a few seconds at most) from the system. If, however, the running time is not of the essence, CPLEX should be used.

MCH, on the other hand, succesfully solves the problem instances in a fraction of time of the CPLEX solver execution, up to the order 10^{-4} (100ms vs. 1500+s for the problem instance 100-56-90).

Figure 4.7 shows an example schedule generated by MCH for the 40-32-40 problem instance, while Figure 4.8 shows the corresponding power load profile, in comparison with the two other heuristics.

We can see that the area under the MCH consumed power load curve is the largest, i.e. it satisfies the most energy demand among the three heuristics. For this specific problem instance 40-32-40, GH satisfies 53%, LRT 59%, and the MCH 67% of the total energy demand. Table 4.12 shows total energy charged (demand satisfied) per heuristic over all problem instances.

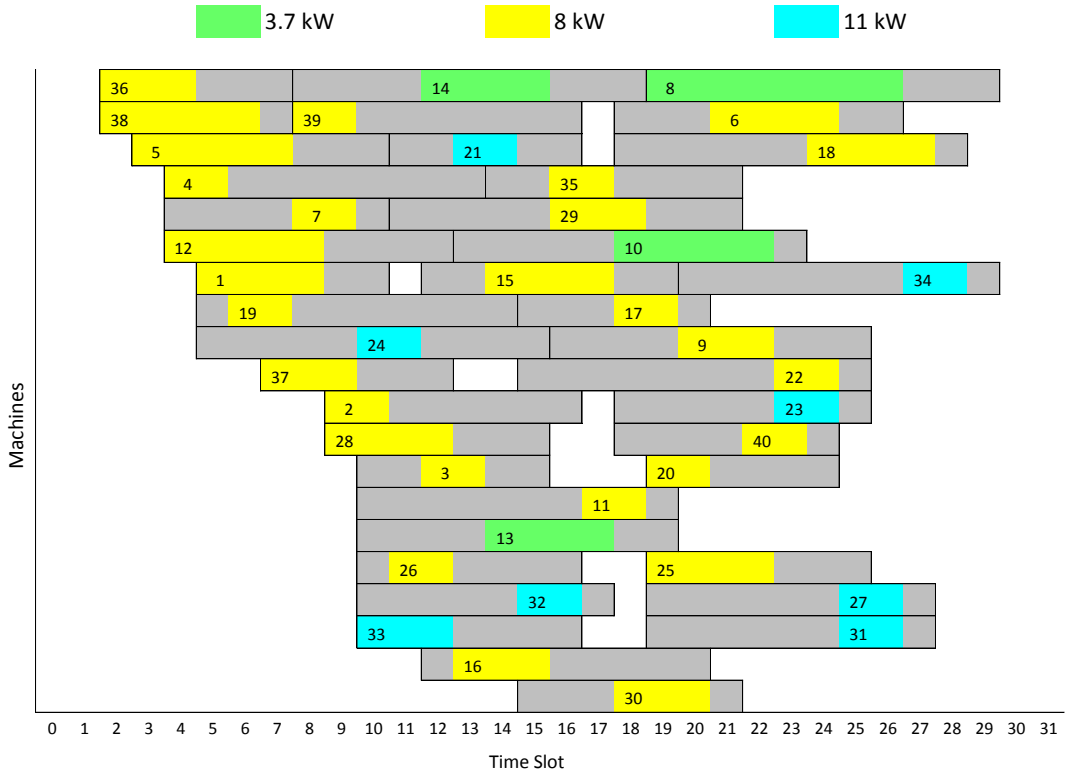


Figure 4.7: Schedule generated by MCH in offline mode

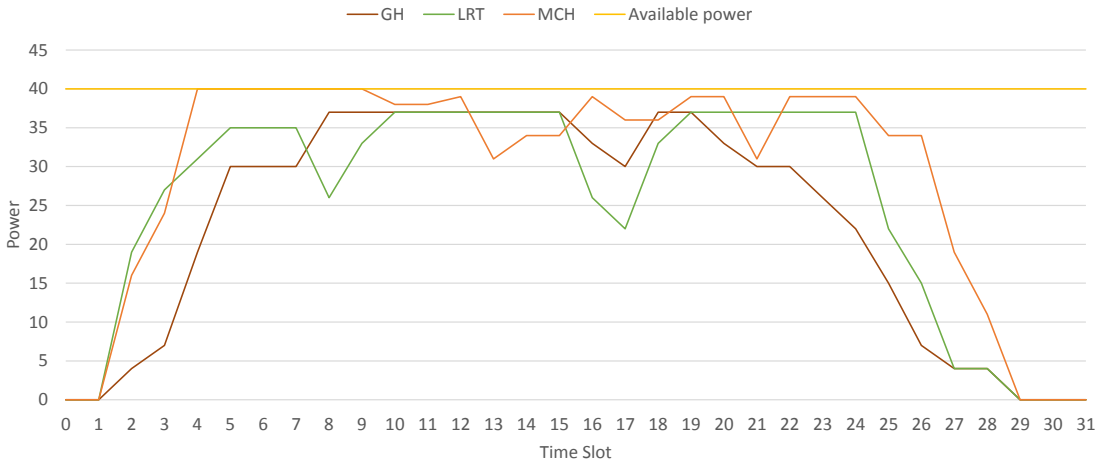


Figure 4.8: Consumed power (charging load) vs. available power; $P(t) = 40$ (constant)

Table 4.12: Total energy charged; *MinMax* model, $\alpha = 0.95$

		GH		*LRT*		*MCH (Max)*	
Problem Instance	$Demand$	$Charged$	%	$Charged$	%	$Charged$	%
20-16-30	160	124.027	0.775	118.014	0.738	137.944	0.862
40-32-40	329	173.262	0.527	195.343	0.594	219.929	0.668
60-40-55	493	251.275	0.510	286.476	0.581	302.813	0.614
80-48-70	651	327.728	0.503	366.899	0.564	392.511	0.603
100-56-90	816	423.654	0.519	472.835	0.579	501.300	0.614
120-64-100	974	496.560	0.510	547.498	0.562	583.454	0.599

4.3.2 Online (Dynamic) Mode

Finally, we have integrated the scheduling algorithm using MCH into a charging station controller of the KOFLA simulator, to simulate real-world scheduling.

There are several key differences in this online (dynamic) setting compared to the offline mode (which we described in detail in section 3.4):

- *time transformation* from real time into discrete time slots
- the *rolling schedule* with NOW marker simulating the flow of time
- possible *rescheduling* of activities for which the charging has not yet commenced, allowing for continuous improvement
- *prediction errors*, ...

As opposed to the *offline* mode, when the scheduler has complete and deterministic knowledge of all of the activities in advance (and builds the schedule from scratch), in the *online* mode the schedule is built *incrementally*, and the system operates only with the *subset of activities* that fall into the upcoming planning time horizon. On one hand, there are vehicles that already left and do not need to be planned for any more; on the other hand, there are vehicles for which no activity has been registered with the system yet (no *reserve* event occured yet).

Here it makes even more sense to use the strategy of *sorting the charging plan instances* before assigning one of them as initial, since the activities are considered *one-by-one*. The previous schedule was valid, so the only conflicts that can occur are the ones including the new activity, and here we attempt to include the ones of highest profit first.

The online system has to deal with *prediction errors*, which typically involve vehicles arriving or leaving earlier or sooner than planned, leading to suboptimal charging schedules. *Leaving too early* possibly leads to interruption of the activity's charging plan, and consequently to less than planned satisfied energy demand; *leaving later* than planned unnecessarily occupies the parking space resource.

In the following experiment we simulate a shopping mall parking (and EV charging) lot, with 80 arriving EVs, and compare the quality of the online MCH solution to that of an offline optimal solution. We first run the *online scheduler* for a 24-hour simulation, using a realistic vehicle charging load distribution (Figure 4.9). The actual parking duration with an average of 3.5 hours is statistically modified to simulate prediction errors, according to a normal distribution. The

generated activities are then fed into the *offline scheduler* (optimal, CPLEX), with their actual leaving times, for comparison.

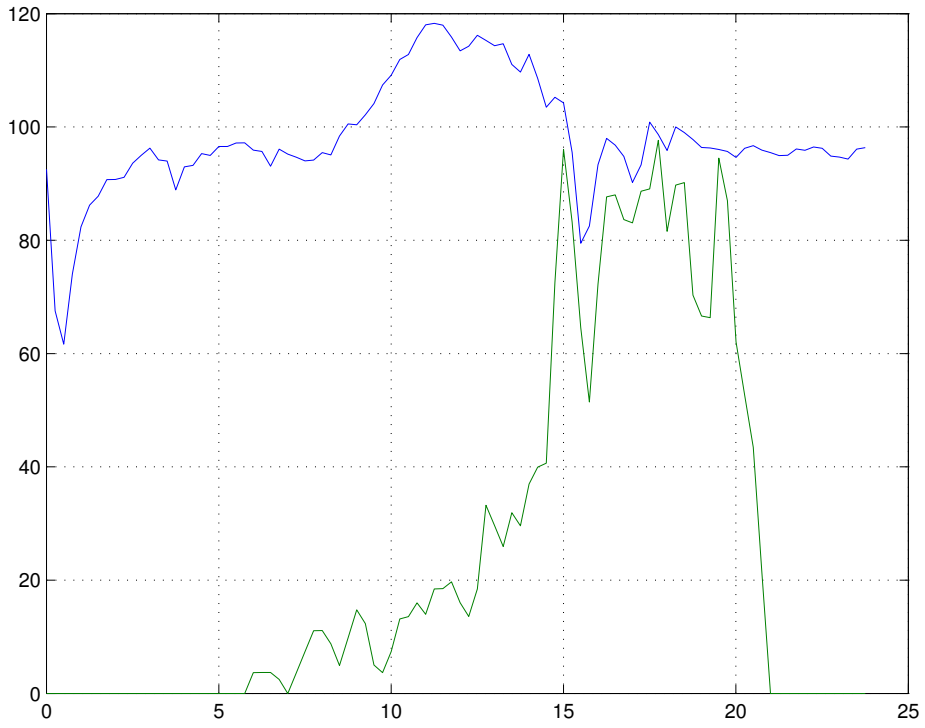


Figure 4.9: Available power and charging load profile at the shopping mall; total power limit P_t (*blue*) and total charging load (green) (source: [9])

We would like to show how much better is the offline scheduler for different levels of prediction error (defined by standard deviation) in the real versus planned parking duration, i.e. how much does the prediction error, present in the online mode, affect the solution quality.

The results in Table 4.13 confirm, as expected, that the performance of the MCH scheduler in the online mode degrades for additional 9% when we let the average planned parking duration of 3.5 hours vary with a standard deviation of 30 minutes. We must note, however, that this result is specific to our model of the "shopping mall" car arrival distribution, as illustrated by the load curve in Figure 4.9.

Figure 4.10 shows the entire dynamically generated schedule at the end of the day, after the simulation was run.

Table 4.13: Online MCH scheduling performance compared to the offline optimum, under parking duration prediction error; 80 vehicles, *MinMax* model, $\alpha = 0.95$ (source: [9])

Parking duration prediction error σ [min]	Objective ratio $\frac{Z_{MCH,\ online}}{Z_{CPLEX,\ offline}}$ [%]	Total power charged ratio online/offline [%]
0	88%	83%
30	79%	75%
60	76%	72%

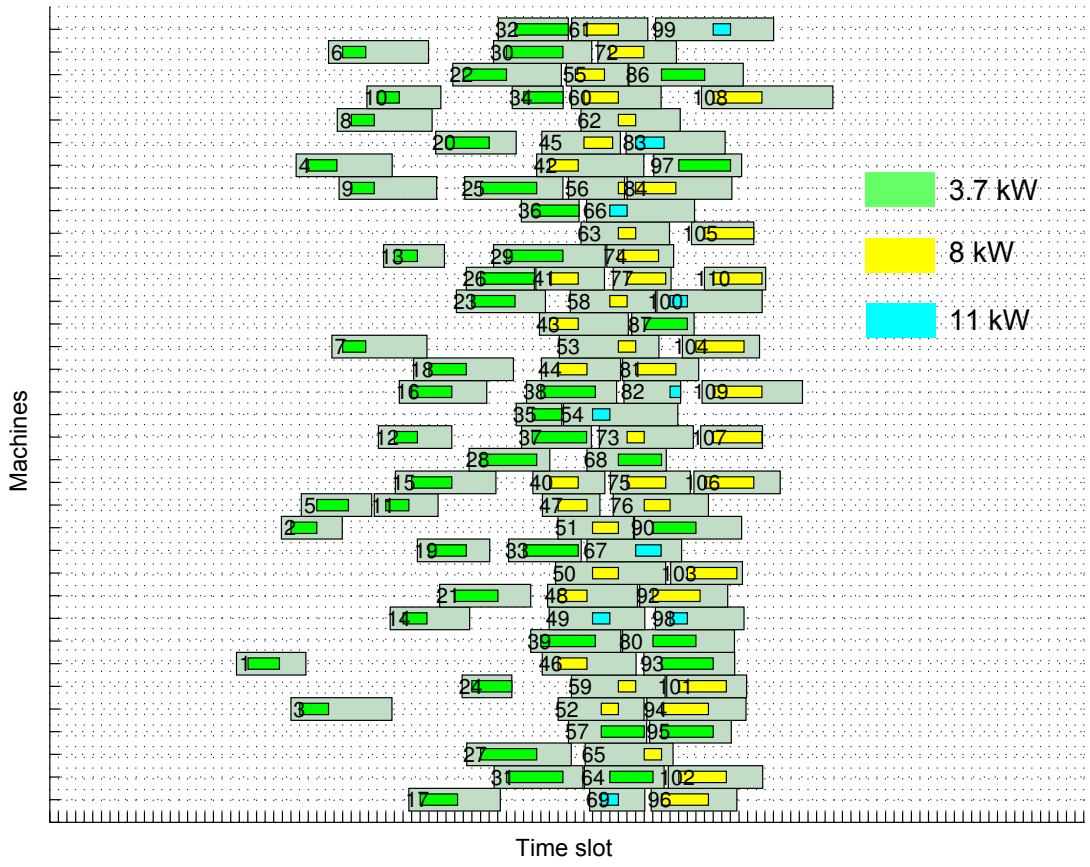


Figure 4.10: Schedule generated in the online mode (source: [9])

CHAPTER 5

Conclusions and Future Work

With the massive deployment of electric vehicles (EVs), public charging will become necessary in addition to home charging overnight. Under this assumption, in this work we described and analyzed the efficient charging of EVs arriving to a public charging station, and contending for a limited amount of the electric power resource during the planning time horizon. We denoted this problem as Electric Vehicles Recharge Scheduling with Time Windows (EVRSTW). Our model incorporates some specific application requirements — such as time windows, minimum and maximum energy demands, different charging rates, discretization of time etc. — which make the problem highly combinatorial (the decision variant of its special case is NP-complete).

EVRSTW can be split (under the assumption of identical machines, i.e. charging points) into two subproblems. The first subproblem — the one of allocating the activities to machines — can be optimally solved by a greedy method. Thus, all of the hardness of the EVRSTW problem lies in the second subproblem — the one of identifying the start time and duration of charging (defined by the charging rate) within the parking time window, while maximizing the profit.

Since the problem belongs to the family of resource allocation problems, we searched for heuristic approaches to replace the exact methods, which solve the mixed integer programming formulation. Using the exact method (CPLEX solver, MIP) entails *unpredictable runtimes* — for the same problem size the execution can take from few seconds to tens of minutes [7]. In order to build a real world dynamic system based on requests and almost immediate responses, we needed to employ methods that obtain reasonably good solutions within a *bounded execution time*.

We showed that the min-conflicts heuristic, as designed in this work, can produce promising results in a fraction of the CPLEX solver's execution time (up to the order of 10^{-4}, less than 200 milliseconds versus 25+ minutes in the extreme case). It manages to achieve from 75% up to 93% for the *SingleDemand* model, and from 80% up to 91% for the *MinMax* model, of the optimal solutions recorded by the exact CPLEX solver (offline mode). In an online environment, where the response time is of large significance (such as KOFLA framework), MCH would be

an immediate choice over CPLEX. If, on the other hand, we are looking for optimal solutions and do not care for the execution times, the exact method should be favored.

The min-conflicts heuristic confidently outperforms the other two compared heuristics — greedy heuristic and local ratio technique — by always producing feasible solutions (on the tested problem instances), scheduling all of the activities in a consistently short time.

Further on, the min-conflicts heuristic has been integrated into the online environment by means of a charging station controller, which is designed to handle various framework events and provide signals for a reliable system performance. We have shown that, in this setting, the parking period uncertainty represents the major issue, and has a significant impact on the solution quality. In this online setting, the response time is very important, and MCH has proven its potential with running times in terms of hundreds of milliseconds, while producing solutions of acceptable quality.

5.1 Future work

Future work should investigate the possible improvements of the min-conflicts heuristic as designed here, and perhaps even its hybridization with other local search techniques, e.g. tabu search. One such area, where our version of MCH could be improved, is the fitness evaluation function of the neighborhood.

Another research direction would be to incorporate the *electricity price* into the objective (profit) function, as seen in works [29], [18], [11], [27], [21], since it is usually different on and off peak.

A generalization to *non-identical* machines, when the preprocessing step as described here is not possible, would be yet another logical step. This would restrict certain activities to be scheduled only on a subset of machines, which would further increase the complexity of the problem.

Bibliography

[1] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1 – 8, 1987.

[2] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. (S.) Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *Journal of the ACM*, pages 735–744, 2000.

[3] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: A unified framework for approximation algorithms. In memoriam: Shimon Even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, December 2004.

[4] M. Barth and M. Todd. Simulation model performance analysis of a multiple station shared vehicle system. *Transportation Research Part C: Emerging Technologies*, 7(4):237–259, 1999.

[5] BeCharged. BeCharged - Market leader in charging stations for electric vehicles. `http://www.becharged.eu/en/`, January 2014.

[6] A. Beer, J. Gartner, N. Musliu, W. Schafhauser, and W. Slany. An AI-based break-scheduling system for supervisory personnel. *Intelligent Systems, IEEE*, 25(2):60–73, March 2010.

[7] S. Bessler and J. Groenbaek. Routing EV users towards an optimal charging plan. *EVS26 International Battery, Hybrid and Fuel Cell Electric Vehicle Symposium*, 2012.

[8] A. Borodin, D. Cashman, and A. Magen. How well can primal-dual and local-ratio algorithms perform? In *Proceedings of the 32Nd International Conference on Automata, Languages and Programming*, ICALP'05, pages 943–955, Berlin, Heidelberg, 2005. Springer-Verlag.

[9] D. Bucar, S. Bessler, N. Musliu, and J. Groenbaek. Scheduling of electric vehicle charging operations. Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA), 2013.

[10] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. An improved approximation algorithm for resource allocation. *ACM Trans. Algorithms*, 7(4):48:1–48:7, September 2011.

[11] K. Clement, E. Haesen, and J. Driesen. Coordinated charging of multiple plug-in hybrid electric vehicles in residential distribution grids. In *Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES*, pages 1–7, 2009.

[12] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theor. Comput. Sci.*, 411(49):4217–4234, November 2010.

[13] U. Eberle and R. von Helmolt. Sustainable transportation based on electric vehicle concepts: a brief overview. *Energy Environ. Sci.*, 3:689–699, 2010.

[14] FTW. KOFLA – Kooperatives Fahrerunterstützungssystem für optimiertes Lademanagement von elektrischen Fahrzeugen. `http://www.ftw.at/research-innovation/projects/kofla`, 2012.

[15] V. Gabrel. Scheduling jobs within time windows on identical parallel machines: New model and algorithms. *European Journal of Operational Research*, 83(2):320–329, June 1995.

[16] R.-J. He. Parallel machine scheduling problem with time windows: a constraint programming and tabu search hybrid approach. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 5, pages 2939–2944 Vol. 5, 2005.

[17] F.S. Hillier and G.J. Lieberman. *Introduction to operations research*. McGraw-Hill, 9 edition, 2009.

[18] C. Hutson, G. K. Venayagamoorthy, and K. A. Corzine. Intelligent scheduling of hybrid and electric vehicle storage capacity in a parking lot for profit maximization in grid power transactions. In *Energy 2030 Conference, 2008. ENERGY 2008. IEEE*, pages 1–8, 2008.

[19] A. W. J. Kolen, J. K. Lenstra, C. H. Papadimitriou, and F. C. Spieksma. Interval scheduling: A survey. *Naval Research Logistics (NRL)*, 54(5):530–543, 2007.

[20] J. A. P. Lopes, F. J. Soares, and P. M. R. Almeida. Integration of electric vehicles in the electric power system. *Proceedings of the IEEE*, 99(1):168–183, 2011.

[21] S. Mal, A. Chattopadhyay, A. Yang, and R. Gadh. Electric vehicle smart charging and vehicle-to-grid operation. *International Journal of Parallel, Emergent and Distributed Systems*, 28(3):249–265, June 2013.

[22] S. Minton, A. Philips, M. D. Johnston, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1993.

[23] N. Musliu. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4), 2006.

[24] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *Journal of Scheduling*, pages 879–888, 2000.

[25] H. Qin and W. Zhang. Charging scheduling with minimal waiting in a network of electric vehicles and charging stations. In *Proceedings of the Eighth ACM International Workshop on Vehicular Inter-networking*, VANET '11, pages 51–60, 2011.

[26] IHS Press Room. Number of fast-charging stations for electric vehicles set to rise to nearly 200,000 in 2020. `http://press.ihs.com/press-release/design-supply-chain-media/number-fast-charging-stations-electric-vehicles-set-rise-nea`, August 2013.

[27] P. Sanchez-Martin and G. Sanchez. Optimal electric vehicles consumption management at parking garages. In *PowerTech, 2011 IEEE Trondheim*, pages 1–7, 2011.

[28] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *DIMACS SERIES IN DISCRETE MATHEMATICS AND THEORETICAL COMPUTER SCIENCE*, pages 521–532, 1995.

[29] Y. Sugii, K. Tsujino, and T. Nagano. A genetic-algorithm based scheduling method of charging electric vehicles. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 4, pages 435–440, 1999.

[30] O. Sundström and C. Binding. Optimization methods to plan the charging of electric vehicle fleets. *International Conference on Control, Communication and Power Engineering CCPE2010, Chennai, India*, pages 323–328, 2010.

[31] O. Sundström and C. Binding. Planning electric-drive vehicle charging under constrained grid conditions. In *Power System Technology (POWERCON), 2010 International Conference on*, pages 1–6, 2010.

[32] R. A. Verzijlbergh, Z. Lukszo, J. G. Slootweg, and M. D. Ilic. The impact of controlled electric vehicle charging on residential low voltage networks. In *Networking, Sensing and Control (ICNSC), 2011 IEEE International Conference on*, pages 14–19, 2011.

[33] R. J. Wallace and E. C. Freuder. Heuristic methods for over-constrained constraint satisfaction problems. In *In Proc. CP'95 Workshop on Overconstrained Systems*, pages 97–101. Springer, 1995.

[34] Y. Xu and F. Pan. Scheduling for charging plug-in hybrid electric vehicles. In *CDC*, pages 2495–2501. IEEE, 2012.

[35] S.-H. Yang, W.-S. Cheng, Y.-C. Hsu, C.-H. Gan, and Y.-B. Lin. Charge scheduling of electric vehicles in highways through mobile computing. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 692–698, 2011.

[36] T. Zhang, W. Chen, Z. Han, and Z. Cao. Charging scheduling of electric vehicles with local renewable energy under uncertain electric vehicle arrival and grid power price. *IEEE Transactions on Vehicular Technology*, 63(6):2600 – 2612, 2014.