MISTA 2013

Scheduling of electric vehicle charging operations

Damir Bucar, · Sandford Bessler, · Nysret Musliu, · Jesper Groenbaek

Abstract In this work we address the optimization problem related to the recharge of electric vehicles, in which parking place availability, parking duration and available power constraints have to be satisfied. We formulate this problem as a resource allocation problem (with time windows) and solve it with exact and heuristic methods. The results obtained with a min-conflicts heuristic are promising. Further, we apply the static problem to an online, real world environment, and shortly discuss additional challenges and the performance of the overall system.

1 Introduction

The electric vehicle (EV) charging is a topic that draws the attention of many scientific communities: their members are power and smart grid experts, signal processing and telecommunications scientists, mobility as well as traffic engineers, or computer scientists specialized in algorithms. Electromobility is indeed at the crossroads of several disciplines and innovation is still possible, as a single standardized solution is not yet deployed or even agreed upon.

As the recharging power doubles the consumption of households, a massive deployment of charging operations will affect in particular the low voltage grids. With today's battery technology, EV charging durations spans still several hours. One way to improve the planning and customer satisfaction at public charging stations is to

Damir Bucar
Vienna University of Technology
E-mail: damir.bucar@gmail.com

Sandford Bessler
FTW Telecommunication Research Center
E-mail: bessler@ftw.at

Nysret Musliu
Vienna Univerity of Technology
E-mail: musliu@dbai.tuwien.ac.at

Jesper Groenbaek
FTW Telecommunication Research Center
E-mail: groenbaek@ftw.at

allow reservation during or before the trip. In this way, both a parking place (including a charging point) and the estimated energy required for charging can be planned well before the arrival of the car. Smart resource allocation algorithms are required in particular at public charging stations, because of the stochastic character of car arrivals and because of possible limitations in power and energy supply.

In the application domain, the problem is easily described: the EV parking lot owner (and charging service provider) has a list of vehicles, (some already arrived, the others on the way) with their planned parking duration and their required amount of energy. In addition, he knows how much power will be available for all the cars, in each time period, in the near future. The question is, when to switch on and off the charging power for every car, so that he can maximize a profit function?

A number of works have already shown that, optimized, grid-aware charging strategies reduce grid congestion and allow a higher penetration of EVs without additional investment in the grid infrastructure ([1], [3],[2], [4]). Most of the works exploit the so called "valleys" in the load curve of the local grid and shift the charging tasks into these periods. For charging at public stations, the model has however to include the estimated arrival at the charging station and the estimated parking time. A first contribution of this work is therefore to identify and formulate the EV charging problem as a resource allocation problem (RAP) with time intervals [10], [5], which is closely related to the bandwidth allocation problem (BAP) [13], and the resource constrained scheduling problem [12]. RAP is NP-hard, since it contains as special case the classical $0-1$ knapsack problem, where all time intervals are identical, and which is itself NP-hard.

Therefore, by using a mixed integer program and the CPLEX solver, we can solve problem instances smaller than 100 cars in reasonable time. For larger instances we investigated first greedy heuristics available in the literature, but with poor results. A more sophisticated heuristic, the local ratio, is investigated and is compared with a min-conflicts based heuristic, which performs best of all. We discuss the solution quality comparing it with the solution of the mixed integer program.

The rest of the paper is organized as follows: in Section 2 we formulate the EV charging problem as an integr program, in Section 3 we describe a number of heuristic solution approaches, in Section 4 we present computational results. In Section 5 we report on the extensions required for the online, real world scheduling version of the EV charging, and present first results. Finally in Section 6 we draw concluding remarks.

2 The EV charging problem

The EV charging problem which has been studied before [1], [2],[3], [4] is formulated in this work (for the first time, to our knowledge) as a machine scheduling problem with time windows. The machines are the charging points that charge the connected EVs with one of a few predefined charging speeds, and without preemption of the operation. Although preemption would add flexibility in scheduling, it would also add complexity in the handling of charging states and in the payment transactions. The time horizon is discretized in T discrete intervals (for instance 15 minutes each). Using the notation in [5], each EV is associated with an activity $a \in A$ that includes parking and charging at a charging point $m \in M$ (m includes the parking place). The actual charging has to occur within the given time window. The aim is to determine for each activity the charging start time, charging duration and speed, the allocation of activities to charging points,

Notation	Description
T	planning horizon with time slots of $t = 1..T$
A	Set of charging activities
M	set of charging points
S	set of charging speeds
I	instances generated from activities A
e_j, l_j	time window (earliest, latest) $j \in A$
s_i, f_i	start and finish time of charging , $i \in I$
$dmin_j$	minimum energy demand of activity j
$dmax_j$	maximum energy demand of activity j
a_i	activity that generates instance i
w_i	charging speed of instance i
p_i	calculated profit for instance i
P^t	total available power during $t \in T$
v_i^t	timeslots in which parking=true for instance i
x_i	variable: $x_i \in \{0, 1\}, i \in I$.
$k[w_i]$	cost for charging at speed w_i

Table 1 Notation summary

such that the total power consumed in any time period does not exceed a given value, and such that a certain profit objective function to be explained below is maximized.

Note that we have to deal with two types of resources: parking place during the whole time window, and power limitation during the charging period. If we make the assumption that the charging points are identical (machines), we can split the problem into two sub-problems:

- since the machines are identical, we allocate first the activity time windows to the charging points. This problem is similar to scheduling classes to classrooms, and can be solved optimally by a greedy heuristic (increasing starting time rule). The intervals are the activity time windows and cannot be shifted. The resulted allocation of activity to charging point allows us to address the second subproblem as a one machine problem.
- this is a bandwidth or resource allocation problem over time intervals, or RAP which is a NP-hard problem, since it can be reduced to a knapsack problem if the time windows are set to the interval [0,1]. The specific problem which we denote EVRSTW (electric vehicle recharge scheduling with time windows) requires that the charging takes place within the time window of the activity and is limited by total power available in each of the T periods.

In case the machines are not identical, i.e. are divided in two groups: slow and fast charging, the machine allocation can be still made under the following assumption: a fast charging point is an expensive resource, therefore is will not be used for slow charging, which leads to two disjoint activity groups. In the most general case, instances have to be created for each individual machine, making the problem very large.

In the rest of this section we assume the machines are identical, so that the allocation activity-charging point has been done and we focus only on the second subproblem, the RAP.

An important part of the model is the requested energy demand by an EV: a minimum demand *dmin* has to be satisfied and is calculated based on the state of charge on arrival at the charging station and the remaining trip in km until the next recharge. The maximum demand *dmax* is the energy required to fill the battery.

In order to deal with several resources and variables (such as charging speed) in resource allocation problems Bar-Noy et al [5] propose to create an "instance" for each combination of these variables. The original problem is solved when a subset of instances is selected. We use the same method to create an instance $i = i(a, m, w, d, l)$ for each combination of allowed values of the following variables: $a \in A$, $m \in M$, $w \in S$, $d \in [dmin_a/w, dmax_a/w]$, and $l \in [e_a, l_a - dur(d, w)]$, where the charging duration dur is computed from the speed and demand values.

After generating the set of instances I, the problem reduces to finding a set of binary variables x_i, $i \in I$, as we will see below. The profit of an instance p_i is defined as a weighted sum of two objective contributions: the completion degree (the ratio between the energy charged and the maximum energy requested) and the lifetime preservation factor k/w_i of the battery, which we assume to be inversely proportional to the charging speed, (k is a constant, e.g. k=2):

$$p_i = \alpha d_i/dmax_{a_i} + (1 - \alpha)k/w_i, 0 \leq \alpha \leq 1 \tag{1}$$

The profit of an activity is not uniform, making the search for a solution more difficult [10]. In Fig. 1 we depict the profit values of one activity. The function is discrete as the possible demands and the speeds build discrete sets, however some solution points are cut away by power constraints at high charging speeds and by the time window duration at lower speeds. When tuning the function p_i, we take care to assign the most weight to the completion degree, so that the preference for low charging speeds, only slightly affects the solution.

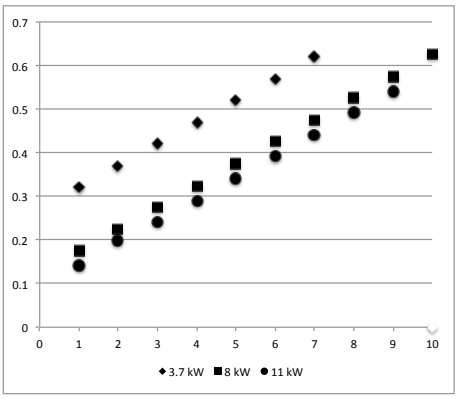

Fig. 1 Profit values for the instances of an activity a_i and $\alpha = 0.5$. The X axis depicts the charged amount relative to the maximum demand $d_i/dmax_{a_i}$

The MIP formulation becomes
Problem EVRSTW:

$$Z = \max \sum_{i \in I} p_i x_i \tag{2}$$

$$s.t. \sum_{i \in I | a_i = j} x_i = 1, j \in A \tag{3}$$

$$\sum_{i \in I | t \in [s_i, f_i]} w_i x_i \leq P^t, t \in T \tag{4}$$

$$\sum_{i \in I | m_i = m} v_i^t x_i \leq 1, m \in M, t \in T \tag{5}$$

$$x_i \in \{0, 1\}, i \in I \tag{6}$$

The objective in (2) is to maximize the total profit. The equality (3) makes sure that exactly one instance of each activity is selected. The inequality (4) limits for each time slot t the total power due to contributed instances. Finally, the contraints (5) avoid that two cars park ($v_i = 1$) on the same parking place m in the same time slot.

3 Solution approaches

Three different heuristics have been implemented and tested: greedy, local ratio and min-conflicts. For reference, exact solutions have been obtained using an AMPL formulation and the CPLEX solver.

3.1 Simple greedy heuristic

The greedy heuristic takes only those instances generated by usage of minimum charging speed (w_{min}) which start at the beginning od the activity's time window (e_j). The steps are shown in Algorithm 1.

Algorithm 1 Greedy Heuristic scheduling

```
for all a ∈ A do                                   ▷ A - activities
    I ← IG.getGreedyInstancesFor(a)
    if I is empty then
        continue
    end if
    I.sort()                                       ▷ according to profit
    for all i ∈ I do
        if i is feasible then                      ▷ doesn't violate the power constraint
            a.setInstance(i)
        end if
    end for
end for
```

As expected, this approach performs poorly. The first issue is the activity's time window - if it is too short, the list generated by the instance generator will be empty. This happens when even the minimum energy demand ($dmin_j$) of the activity can not be satisfied by the charging speed of w_{min} within the parking duration. The second issue is the usual problem of the greedy algorithms: if the current instance can not fit into the schedule, it gets thrown away and never observed afterwards. These two factors cause that many activities are not being scheduled at all.

3.2 Local ratio heuristic

The local ratio algorithm belongs to a class of stack based heuristics [7], and has been proposed in packing problems to maximize bandwidth throughput [8], [5]. In our case the bandwidth corresponds to the charging power. For this type of problems, the algorithm can achieve a 1/3 approximation, which gets better as the ratio between the charging power (speed) of individual activities and the total available charging power is smaller. The algorithm works as follows: the instances I are first sorted in increasing order of their end (charging) times. In the first sweep we select an instance 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. Instances with non-positive profits are removed, and the selected activity is pushed on a stack. When no activities remain unconsidered, in the second sweep activities are popped from the stack and those who violate the total power constraint (4) are deleted. Denote the selected instance $\tilde{i}$, then the profits of conflicting instances p_i are reduced as follows: $p_i = p_i - \beta \bar{w_i} p_{\tilde{i}}$, where $\beta = 2$. Other than in the exact algorithm, the amount of resource has to satisfy $0 \leq \bar{w_i} \leq 1$. The problem arises at the calculation of if $\bar{w_i}$ as w_i varies in the interval $[s_i, f_i]$. The conservative rule in (7) would only waste resources because even a short drop in the capacity (available power) would affect all the charging periods around it.

$$\bar{w_i} = \min_{t \in [s_i, f_i]} w_i \tag{7}$$

The algorithm and its implementation are described in [5].

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-hard nature), followed by the use of greedy and local ratio heuristics (which had the problem of not scheduling all the activities), we investigated the application of min-conflicts heuristic.

3.3 Min-Conflicts Heuristic

This section describes a new solution for our problem based on the main ideas of min-conflicts heuristic. This method was proposed by Minton et al. [14] and it has been proven very useful on solving different constraint satisfaction problems. Min-conflicts heuristic has been applied, among others, for the Hubble Space Telescope scheduling problem [14], Workforce Scheduling [18], Break scheduling [17], N-Queens problem [14], etc. This technique has also been used in combination with noise strategies such as random walk [15]. Furthermore, a similar algorithm to Min-Conflicts heuristic is currently one of best local search strategies for solving Boolean satisfiability problems [16].

Instead of constructing feasible solution gradually, the min-conflicts heuristic generates suboptimal initial solution and then improves it iteratively. The main idea of this method is to concentrate the search in the neighborhood of 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 iteratively improved with the following strategy: in each iteration min-conflicts selects randomly 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 optimum. 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 for our problem, we first formulate it as a constraint satisfaction problem 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 timeslot-wise power constraint violation represents a conflict. The final state with no conflicts is when the power consumption in each timeslot is **smaller than or equal to** the available power, i.e. when the power constraint is not violated.

3.3.1 Internal Solution Representation

The Schedule **S** is internally represented as a two-dimensional MxT matrix. Each timeslot represents a 15 minute interval. In this matrix, the time window of an activity is marked (in the row corresponding to the machine m) with the activity ID. A positive activity ID value denotes, that the vehicle is charging, whereas a negative value means "parking only". An example of a schedule is shown in Figure 2.1. This data structure is significant for the performance of the algorithm, since it allows for rapid access to the activities while traversing the search space. A hashed map is therefore used, with activity IDs as keys. The map leads to the information about the activities and the assigned instances (charging plans).

```
::: 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   3   3   3   3   3 -17 -17  17  17  17   0   0   0   0   0   0   0   0   0   0   0 |  0
  0   0   0   0   4   4   4   4   4   4  -4  -4 -10 -10 -10 -10 -10 -10 -10 -10  10  10  10   0   0   0   0   0   0   0   0 |  1
  0   0   0   0  -7   7   7   7   7   7 -11 -11  11  11  11  11  11  11  11  -8  -8  -8  -8   8   8   8   8   8   0   0   0 |  2
  0   0   0   0 -12 -12 -12  12  12  12  12  12  15  15  15  15  15 -15 -15  20  20  20  20  20   0   0   0   0   0   0   0 |  3
  0   0   0   0   0   1   1   1   1  -1  13  13  13  13  13  13  13 -13 -13   0   0   0   0   0   0   0   0   0   0   0   0 |  4
  0   0   0   0   0  19  19 -19 -19 -19 -19 -19 -19 -19   0   0  -9  -9  -9  -9   9   9   9  -9  -9   0   0   0   0   0   0 |  5
  0   0   0   0   0   0   0   0   0 -14  14  14  14  14  14  14  14 -14  -6  -6  -6  -6  -6   6   6   6   6   0   0   0   0 |  6
  0   0   0   0   0   0   0   0   0   2   2   2   2   2  -2  -2   0   0 -18 -18 -18 -18 -18 -18  18  18  18  18   0   0   0 |  7
```

Fig. 2 Example schedule internal representation

3.3.2 Initial Solution

The generation of an initial solution is done in several different ways. Initial instances (charging plans) have been assigned to activities by one of the following rules:

a) *random*: assign a random instance
b) *greedy max-profit-first*: assign a maximum profit instance, but only among the ones with a low charging speed of 3.7 kW (*greedy*); if such does not exist, turn to a)
c) *max-profit-first*: assign a maximum profit instance among all instances

The technique used depends on the problem size and on the mode of operation of the scheduler: online or offline. For the offline mode, when problem instances are bigger (e.g. more than 40 activities), *max-profit-first* technique creates an infeasible initial solution. In the online mode, when the activities are added into the schedule one by one at different points in time, this method performs well.

3.3.3 Optimization

After the initial instance has been assigned to each of the activities, the optimization process starts. The method is presented in Algorithm 2.

Algorithm 2 Optimize

```
negativeSlots ← getNegativeTimeslots()
while negativeSlots not empty and timeLimit not reached do
    timeslot ← a random timeslot from negativeSlots
    activity ← a random activity charging at timeslot
    newInstance ← bestImprovement(activity)
    activity.setInstance(newInstance)
    updateSchedule(activity)
    negativeSlots ← getNegativeTimeslots()
end while
```

`negativeSlots` is a list of timeslots in which the difference between available power and power consumption is negative. The optimization stops when there are either no more timeslots with negative power differences or when the time limit is reached. In the `bestImprovement` procedure, the solution is iteratively improved. Its framework is shown in Algorithm 3.

Algorithm 3 Best Improvement

```
bestInstance ← activity.getInstance()
bestPDiff ← calculatePowerDifferences()
instances ← IG.getInstancesFor(activity)
if p ≤ 0.1 of a random number [0,1] then
    return a random instance from instances
end if
sort instances by profit*
for i ∈ instances do
    newPDiff ← updatePowerDiff(i)
    if sumNegatives(newPDiff) > sumNegatives(bestPDiff) then
        bestInstance ← i
        bestPDiff ← newPDiff
        if sumNegatives(newPDiff) == 0 then
            break
        end if
    end if
end for
return bestInstance
```

*The sorting step, depends again on the system mode and the size of the problem instance. In the offline mode it is performed only in case of $|activities| <= 20$, due to feasibility concerns.

The `calculatePowerDifferences` function calculates the difference between the available power and the power consumption in each timeslot. This is also used in the function `getNegativeTimeslots` in Algorithm 1.

The `updatePowerDiff` function is a quick way of determining improvement that the given instance i_{new} offers. It iterates through the timeslots, deduces the power used by the i_{old}, and adds up the power of the newly assigned instance i_{new} in their

"active" timeslots. If the sum of the negative values in `newPDiff` array is greater (i.e. less negative) than the previous best, then i_{new} is assigned as the new `bestInstance`.

The `sumNegatives` function serves as a fitness function here, and it sums up the negative values in the power differences array. This is a fast method of determining the local fitness improvement. It is possible to do so, since all the instances of other activities remain fixed.

4 Computational Results

The described heuristics for the offline scheduling problem EVRSTW have been implemented and solved a set of synthetic generated problem sets. The problems were solved in an *offline mode*, e.g. all the activities were fed into the algorithm at once. The schedule is then built "from scratch", and the algorithm works with the whole problem.

The problems were selected to simulate a moderate load, meaning that the solution contains several EVs that cannot charged with the maximum requested energy. The problems are characterized by [A]-[M]-[P]: the number of activities (A), charging points (M) and power limit P, which is kept constant during the time horizon T. All the problems are feasible, i.e. there are enough machines, and each charging activity can get the minimum requested energy amount. Along with the list of activities, for each problem instance the number of machines and a power limit are given. The former is an integer value and depicts how many charging spots we have at hand, while the latter is either a single real value or an array of values,e.g. the profile of the available power which varies throughout the day.

The problems were generated using the uniform distribution U(a,b): e_j= U(0,20), l_j= e_j+ U(4,10), $dmin_j$= U(3,5), $dmax_j$= $dmin_j$+U(3,5).

Config.	GH	LRH	MCH (min-avg-max)	CPLEX
20-16-30	10	15.16	14.32 15.59 17.15	19.20
40-32-40	11.47	23.68	22.30 21.59 23.59	29.60
60-40-55	12.96	34.16	32.11 32.68 33.53	41.48
80-48-70	17.28	45.78	41.85 43.34 44.74	53.47
100-56-90	22.52	59.81	53.91 55.15 56.38	67.93
120-64-100	28.76	69.09	64.91 66.34 73.56	78.52

Table 2 Objective function for different algorithms, $\alpha = .95$

Config.	GH	LRH	MCH (min-avg-max)	CPLEX
20-16-30	.006	0.16	1.35	24
40-32-40	.066	0.34	0.14	26
60-40-55	.078	0.36	1.10	192
80-48-70	.186	0.46	5.98	527
100-56-90	.202	0.51	8.88	1500+
120- 64-100	.155	0.73	111.70	468

Table 3 Runtime of algorithms (s) (average over 50 runs)

Config.	GH	LRH	MCH (50 run avg.)
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	1

Table 4 Refused charging activities (unscheduled)

We compared the heuristic results with an exact solution from a AMPL/CPLEX mixed integer model, setting the stop condition at an optimality gap of 1%. The test results are showed in the Tables 2, where the objective function was computed according to definition (1), Table 3 in which the runtime is seconds is given for all the algorithms, and Table 4 in which certain activities are removed in order to achieve feasibility of GH and LRH. The use of CPLEX has the only disadvantage that, for harder or larger problem instances, it will not terminate with a slightly infeasible solution in due time, although such a solution would be acceptable in the environment.

We observe that MCH performs better than the two other heuristics, achieving up to 80% of the optimal solution. The execution times are moderate, one single instance of the largest problem remained infeasible after the time limit of 300s. The main advantage of MCH upon greedy and local ratio heuristics is the feasibility, expressed here by the number of refused (unscheduled) activities. The objective (fitness) is not significantly better, which is due to the fact that the min-conflicts algorithm is a method to solve constraint satisfaction problems (CSP), and not constraint optimization problems (COP). Therefore, there is definitely room for further improvement in the sense of optimization. The heuristic presented here stops immediately after it reaches feasibility, i.e. the moment it finds the schedule which does not violate the power constraint in any of the timeslots. Fitness calculation could be further modified towards prefering better solutions among the feasible ones.

Another aspect that created less problems than were expected was the memory requirement for the large number of instances generated. This number depends on combination of variables, especially the size of the time windows.

5 Real world scheduling

The scheduling algorithm using MCH was integrated in a controller that can react to events generated from cars and from the power grid distribution system. The main differences to the static setting are:

1. the schedule is rolling a timeslot at a time, and the system is now able to process different events such as reservations, arrivals within or outside of the planned period, leave and timeout events. The scheduler checks of course whether a machine is still busy, when new activities are allocated.
2. at any time the scheduler deals with a subset of the activities (only those within the time horizon), reducing the number of problem instances. Additionally, the same activity can be rescheduled as long as the selected instance has not started yet. This allows a continuous improvement of the schedule.
3. the sum-power constraint in Eq. (4) is the result of the available power prediction and may take different values during the duration of an activity (instance) as shown

in Fig. 3. This subtile constraint is believed to make the online problem more difficult to solve. A good schedule algorithm would allocate the activities tight around the power drop at 4 p.m. as shown in Fig. 3.

4. the online problem has to deal with prediction errors.

Typical prediction errors occur at the estimation of the leave time of a vehicle: early leaving leads eventually to interruption of charging, but also leaving later than planned leads to a suboptimal schedule. In the following experiments we compare the quality of the online solution to that of an offline optimal decision. Therefore, we first run the online scheduler for one day simulation, using a realistic vehicle charging load distribution, see Fig. 3. 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 with their actual leave times are scheduled by an offline scheduler for comparison. The experiment should show, how much better is the offline scheduler for different levels of the prediction error (which we define by the standard deviation) in the real versus planned duration.

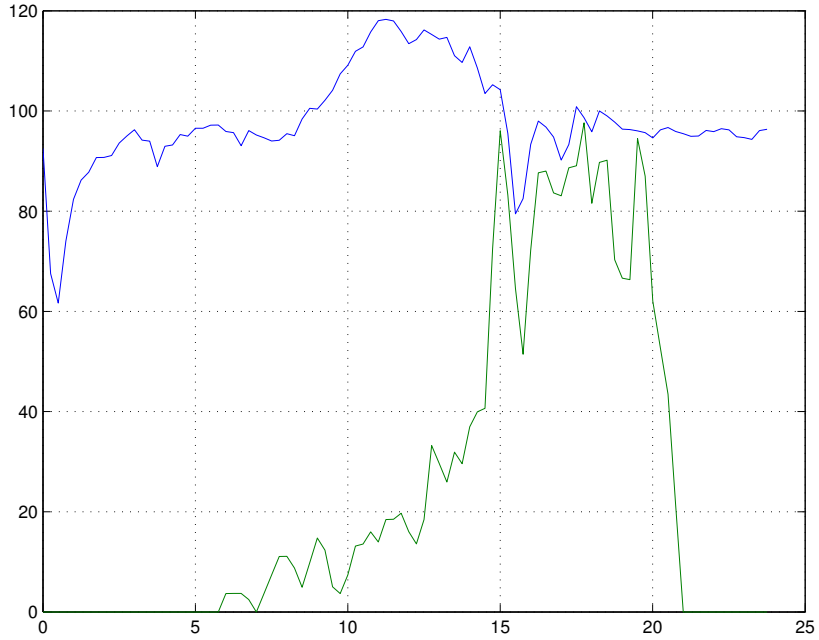

Fig. 3 Charging profile at a shopping mall: Total power limit P_t and total load as a function of time of the day (hour).

The results in Table 5 show as expected that the performance of the scheduler in online mode looses another 9% when the average planned duration of 3.5 hours varies with a standard deviation of half an hour. This result is of course specific to the "shopping mall" car arrival distribution as illustrated by the load curve in Figure 3.

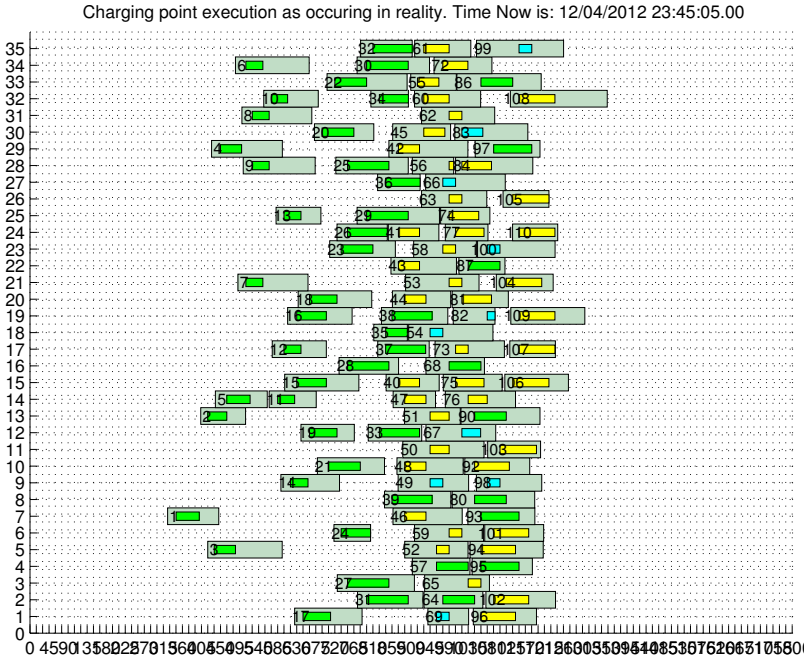

Fig. 4 Online schedule at the end of the day, each tick on the x axis is 15 minutes, the machines 1-35 are on the y axis; the charging speeds are 3.7 (green), 8kW (yellow), 11kW (turquois), the gray rectangles are the time windows.

duration error σ [minutes]	objective ratio online/offline [%]	total charged ratio online/offline [%]
0	88%	83%
30	79%	75%
60	76%	72%

Table 5 Online Scheduling performance compared to offline optimum, under parking duration prediction error (80 cars)

6 Conclusions and Future Work

In this work we analyzed a scheduling problem that occurs when several electric vehicles are recharged at a public charging station. The specific application requirements (time windows, discrete interval, minimum and maximum energy demand) led to the integer program formulation of a resource allocation problem. We showed that a min-conflicts heuristics produces promising results and achieves during a limited runtime (of 10 seconds) and in practical cases around 80% of the optimum obtained by solving exactly the integer program. The heuristics has been integrated in an online environment. The uncertainity in the parking duration has a significant impact on the solution quality, especially in the high load case, where the charging can only be scheduled towards the end of the parking period.

Future work will explore multi-objective functions that reflect different stakeholder interests, such as equipment costs or time varying energy costs. Furthermore, ways to make the schedule more robust to metering data errors in general will be investigated.

Acknowledgements This work has been conducted within the Austrian project KOFLA, which is funded by the programme "ways2go" and the Austrian Federal Ministry for Transport, Innovation and Technology. Nysret Musliu was supported by the Austrian Science Fund (FWF): P24814-N23.

References

1. J. A. Pecas Lopes, F. J. Soares, P. M. Almeida, M. Moreira da Silva, Smart Charging Strategies for Electric Vehicles: Enhancing Grid Performance and Maximizing the Use of Variable Renewable Energy Resources, EVS-24, Stavanger, Norway, May 13-16, 2009.
2. K. Clement, E. Haesen, and J. Driesen, Coordinated Charging of Multiple Plug-In Hybrid Electric Vehicles in Residential Distribution Grids, PES Power Systems Conference and Exposition, 2009.
3. O. Sundstrom and C. Binding, Planning Electric-Drive Vehicle Charging under Constrained Grid Conditions, International Conference on Power System Technology, 2010.
4. S. Sojoudi and S. H. Low, Optimal charging of plug-in hybrid electric vehicles in smart grids, IEEE Power and Energy Society (PES) General Meeting, Jul. 2011.
5. Bar-Noy, Amotz, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber, A unified approach to approximating resource allocation and scheduling, Journal of the ACM (JACM) 48, no. 5 (2001): 1069-1090.
6. Bo Chen, R. Hassim, M. Tzsur, Allocation of bandwidth and storage, IIE Transactions, Vol. 34, Iss. 5, 2002.
7. Allan Borodin, David Cashman, and Avner Magen, How well can primal-dual and local-ratio algorithms perform?, ACM Trans. Algorithms 7, 3, Article 29 (July 2011).
8. Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz, Local ratio: A unified framework for approximation algorithms, in Memoriam: Shimon Even 1935-2004, ACM Comput. Surv. 36, 4 (December 2004), pg. 422-463.
9. Sandford Bessler, Jesper Grønbæk, Max Herry, Andreas Schuster, Rupert Tomschy, Supporting E-mobility Users and Utilities towards an Optimized Charging Service, European Electric Vehicle Congress, Brussels, 26-28 october 2011.
10. Andreas Darmann, Ulrich Pferschy, Joachim Schauer, Resource allocation with time intervals, Theoretical Computer Science, Volume 411, Issue 49, Pages 4217-4234, 5 November 2010.
11. G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, Improved approximation algorithms for resource allocation, in 9th International Integer Programming and Combinatorial Optimization Conference, volume 2337 of LNCS, pages 401414, Springer, 2002.
12. Phillips, Cynthia A., R. N. Uma, and Joel Wein, Offline admission control for general scheduling problems, in Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, pp. 879-888, Society for Industrial and Applied Mathematics, 2000.
13. B. Chen, R. Hassin, and M. Tzur, Allocation of bandwidth and storage. IIE Transactions, 34:501507, 2002.
14. Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird, Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, 58:161–205, 1992.
15. Richard J Wallace and Eugene C Freuder, Heuristic methods for over-constrained constraint satisfaction problems, in *Overconstrained Systems, Springer 1106*, 1996.
16. Bart Selman and Henry A. Kautz and Bram Cohen, Local Search Strategies for Satisfiability Testing, Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability, 1993.
17. Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, Wolfgang Slany, An AI-based break-scheduling system for supervisory personnel, IEEE Intelligent Systems, 25(2):60-73, 2010.
18. Nysret Musliu, Heuristic Methods for Automatic Rotating Workforce Scheduling, International Journal of Computational Intelligence Research, Volume 2, Issue 4, pp. 309-326, 2006.