Maximizing the Number of Satisfied Charging Demands in Electric Vehicle Charging Scheduling Problem

Imene Zaidi[1,2(✉)], Ammar Oulamara[2], Lhassane Idoumghar[1], and Michel Basset[1]

[1] Université de Haute-Alsace, IRIMAS UR 7499, 68100 Mulhouse, France
{imene.zaidi,lhassane.idoumghar,michel.basset}@uha.fr
[2] Université de Lorraine, LORIA Laboratory UMR7503, 54506 Vandoeuvre-lès-Nancy, France
ammar.oulamara@loria.fr

Abstract. This paper addresses the electric vehicle charging problem in a charging station with a limited overall power capacity and a fixed number of chargers. Electric vehicle drivers submit their charging demands. Given the limited resources, these charging demands are either accepted or rejected, and an accepted demand must be satisfied. The objective of the scheduler is to maximize the number of satisfied demands. We prove that the problem is NP-hard. Then, we propose a linear programming model, heuristic, and a metaheuristic combining a simulated annealing algorithm with an iterated local search procedure to solve it. We provide computational results to show the efficiency of the proposed methods.

Keywords: Electric Vehicle · Charging Scheduling · Linear Programming · Heuristic · Simulated Annealing

1 Introduction

Electric vehicles have recently gained wide popularity as low-emission vehicles. According to the International Energy Agency [6], the number of electric vehicles reached 16.5 million in 2021. While in 2010, only hundreds of them were on the road. However, the global adoption of electric vehicles is still challenging since charging an electric vehicle is time-consuming and requires considerable electric energy. Moreover, a mass transition to electric vehicles will lead to a saturation of charging stations and a significant increase in electrical power demand that can overload the power grid. Several studies propose smart charging approaches to avoid these negative impacts without expensively upgrading the existing power grid. In smart charging, a management system controls the charging of electric vehicles and optimally schedules the electric vehicle charging load. This paper addresses the electric vehicle charging scheduling problem (EVCSP) in a charging station where drivers submit charging demand reservations before arriving.

P. Legrand et al. (Eds.): EA 2022, LNCS 14091, pp. 89–102, 2023.
https://doi.org/10.1007/978-3-031-42616-2_7

Given the lack of charging stations, the short range of electric vehicles, and the long time required to charge them, drivers of electric vehicles need to carefully plan their trips to ensure that they will have opportunities to recharge their batteries. As a result, it is preferable for them to confirm in advance that the charger they intend to use is available. Moreover, the Open Charge Point Protocol includes the reservation functionality of charging stations [1].

The remainder of this paper is organized as follows. Section 2 briefly reviews the main works on EVCSP. Section 3 describes in detail the investigated problem. Section 4 provides the complexity of the problem. Section 5 formulates it as an integer linear programming (ILP) model. The proposed heuristic is presented in Sect. 6. Section 7 details the developed metaheuristic that combines a simulated annealing algorithm with a iterated local search procedure. Section 8 evaluates the performance of the proposed methods. The paper closes with some conclusions and future research directions in Sect. 9.

2 Related Work

We focus on studies that investigated the problem of optimizing the charging load of electric vehicles from the perspective of charging station operators. The main objectives of these operators are to reduce the total charging cost [5,14,15] or to maximize the satisfaction of their customers. In smart charging stations, a control system builds a charging schedule while considering the arrival and departure times and the amount of energy requested by each vehicle driver. Many studies assume an uncertain arrival time [4,13,15]. Authors in [14] consider that electric vehicles may arrive with or without a reservation. The electric vehicle drivers can provide the departure times [4,14,16], or they can be estimated based on historical behavior [15]. As for the desired energy, [14] assume that the electric vehicle drivers directly specify their desired energy in kWh. Other papers consider charging electric vehicles to the rated battery capacity [10,12,16]. For constraints related to the charging station, authors in [10,12,16] consider a variable charging power where the charging rate varies over time, while in [4], constant power rates were considered. One of the most commonly used constraints is the capacity of the charging infrastructure. This constraint defines the total power limit of the charging infrastructure, expressed in (kW). Limiting the total charging load of electric vehicles is essential to keep the power peaks low and avoid overloading other equipment and transmission lines. Different optimization approaches were adapted and developed to solve EVCSP. A two-stage approximate dynamic programming was proposed in [16]. Some studies have considered stochastic optimization methods as in [13], where the authors proposed a stochastic linear programming model to schedule the electric vehicles charging load in real-time. Metaheuristics were also applied to solve the EVCSP. For example, we can find a particle swarm optimization in [12,14], a genetic algorithm in [5], a GRASP-like algorithm and a memetic algorithm in [4]. Although the studies mentioned above have examined various aspects of the EVCSP, the charging station operating model, the constraints, and the optimization objective are

different from this study. Thus, comparing results between the proposed methods and literature cannot be pertinent.

3 Problem Description

The formulation of an instance of the EVCSP can be defined as follows. We have a set $\mathcal{J} = \{1, \ldots, n\}$ of charging demands to be scheduled on a set of $\mathcal{M} = \{1, \ldots, m\}$ of chargers. Each charger i delivers a constant power of w_i (kW). The total power that can be delivered by all chargers simultaneously must not exceed w_G (kW), which will further be denoted as the power grid capacity. Each electric vehicle j has an arrival time r_j, departure time d_j, and an energy requirement e_j (kWh) that must be satisfied by its departure time d_j. The charging time p_{ij} of each demand j when assigned to charger i is equal to $p_{ij} = \frac{e_j}{w_i}$. Charging demands can either be accepted or rejected. When a charging demand is accepted, it must be satisfied. At each time, a charger can only charge one vehicle, and a vehicle can only be charged by one charger. During the time interval $[r_j, d_j)$, the vehicle is parked and plugged into charger i. The charging scheduling is preemptive, i.e., the charging operation of each vehicle j can be interrupted at any time and resumed later in the interval $[r_j, d_j)$. Even when the vehicle completes charging before d_j, it still occupies the charger i until it departs. Unless otherwise mentioned, we divide the scheduling time horizon H into T time slots of equal length τ. The scheduling objective is to maximize the number of satisfied charging demands.

4 Complexity

Theorem 1. *The problem of maximizing the number of satisfied charging demands is NP-hard.*

Proof. We show that the problem is NP-hard by proving that its special case where all chargers are identical is NP-hard. Let $\overline{m} = \lfloor \frac{w_G}{w} \rfloor$, where w is the charging power rate of each charger. Clearly, at each time, at most $\overline{m}$ chargers can be activated at the same time. Furthermore, maximizing the number of satisfied charging demands is equivalent to minimizing the number of rejected demands. Minimizing the number of rejected charging demands is equivalent to minimizing the number of late jobs in $\overline{m}$ identical parallel machines scheduling problem with release date and preemption of jobs ($P_{\overline{m}}|prmt|\sum U$). In scheduling problem $P_{\overline{m}}|prmpt|\sum U$ if a job is late in an optimal schedule, it is immaterial where it is scheduled. Thus, scheduling on-time jobs is important. In the optimal schedule, the on-time jobs are scheduled in their interval $[r_j, d_j]$, and at most $\overline{m}$ are used to schedule these jobs. Then the on-time jobs correspond to the set of accepted demands in the problem of maximizing the number of satisfied charging demands. In [3] authors showed that the problem $P_{\overline{m}}|prmpt|\sum U$ is NP-Hard even with two identical machines. Then the problem of maximizing the number of satisfied demands is NP-Hard. □

5 Mathematical Formulations

In this section, we formulate the described problem as an integer linear programming (ILP) model. We define binary variables s_{ij} to specify whether or not the charging demand of electric vehicle j is scheduled on charger i. In addition, we define binary variables x_{ijt} specifying whether or not the electric vehicle j is plugged into the charger i at time slot t. Also, we introduce binary variables y_{jt} that specifies whether or not the electric vehicle j is charging at time slot t. The mathematical formulation is as follows.

$$\max \quad \sum_{j=1}^{n} \sum_{i=1}^{m} s_{ij} \tag{1}$$

$$\sum_{i=1}^{m} s_{ij} \leq 1 \qquad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_{j=1}^{n} x_{ijt} \leq 1 \qquad \forall i \in \mathcal{M}, \quad t \in H \tag{3}$$

$$\sum_{t=r_j}^{d_j} x_{ijt} = s_{ij}(d_j - r_j) \qquad \forall i \in \mathcal{M}, \quad j \in \mathcal{J} \tag{4}$$

$$\sum_{t=r_j}^{d_j} y_{jt} = \sum_{i=1}^{m} p_{ij} s_{ij} \qquad \forall j \in \mathcal{J} \tag{5}$$

$$\sum_{j=1}^{n} \sum_{i=1}^{m} w_i \times s_{ij} \times y_{jt} \leq w_G \qquad \forall t \in H \tag{6}$$

Constraints (2) ensure that when a demand j is accepted, it is assigned to one charger. Constraints (3) ensure that each charger i charges one demand at each time slot t. Constraints (4) ensure that if a charging demand j is accepted to be scheduled on charger i, then it will be plugged into this charger from its arrival r_j to its departure d_j. Constraints (5) ensure that if a charging demand j is accepted, the vehicle j will be charged to its requested energy. Constraints (6) ensure that at each time slot t, the total power delivered by all chargers does not exceed w_G. In addition, variables x_{ijt} and y_{jt} are set to zero for all t where $t < r_j$ and $t \geq d_j$.

Constraints (6) can be linearized by using a new binary variables z_{ijt}. a variable z_{ijt} equals to 1 if variables y_{jt} and s_{ij} equal to 1. Constraints (6) are replaced by the following constraints:

$$z_{ijt} \geq y_{jt} + s_{ij} - 1 \qquad \forall i \in \mathcal{M}, \quad j \in \mathcal{J}, \quad t \in H \tag{7}$$

$$z_{ijt} \leq y_{jt} \qquad \forall i \in \mathcal{M}, \quad j \in \mathcal{J}, \quad t \in H \tag{8}$$

$$z_{ijt} \leq s_{ij} \qquad \forall i \in \mathcal{M}, \quad j \in \mathcal{J}, \quad t \in H \tag{9}$$

$$\sum_{i=1}^{k} \sum_{j=1}^{n} w_i z_{ijt} \leq w_G \qquad t \in H \tag{10}$$

6 Greedy Constructive Heuristic

Since maximizing the number of satisfied demands is NP-Hard, it is hard to find optimal solutions for large-size instances in a reasonable time. Moreover, using a commercial linear programming solver may incur additional costs for charging station operators. Hence, we propose heuristics and metaheuristics. The proposed heuristic, detailed in Algorithm 1, builds a charging schedule by considering vehicles in the non-decreasing order of their arrival time r_j and

breaking ties first by the non-decreasing order of their departure time d_j, then by the non-decreasing order of their energy request e_j. Let $(w_G^t)_{t\in H}$ be a vector of reals that stores the power allocated at each time slot t, which is initialized to 0. For each vehicle j, if at least a charger is available at r_j, the heuristic begins by seeking an available charger with the smallest charging power to charge j without exceeding the current grid capacity (lines 7-10). If such a charger exists, it is selected to charge vehicle j (line 11). Otherwise, the heuristic calculates the value $e(j', r_j, d_j)$ that represents the amount of energy allocated to each scheduled charging demand j' ($j' \neq j$) in the interval $[r_j, d_j)$. The charging demand with the greatest value of $e(j', r_j, d_j)$ will be rejected if $e(j', r_j, d_j)$ is greater than the requested energy e_j (line 15). Otherwise, the vehicle j is rejected (line 16). When no charger is available at r_j (lines 18-22), the charging demand with the maximum departure time is rejected.

Algorithm 1: Constructive greedy heuristic

Input : The set of charging demands $\mathcal{J}$, the set of chargers $\mathcal{M}$, the grid capacity w_G
Output: The assignment of vehicles to chargers, the set of rejected demand

1 Sort $\mathcal{J}$ by non-decreasing order of r_j. Then, in non-decreasing order of d_j. Then, in non-decreasing order of e_j ;
2 Sort $\mathcal{M}$ by non-decreasing order of charging power w_i ;
3 $(w_G^t) \leftarrow (0)_{t\in H}$;
4 **while** $\mathcal{J} \neq \emptyset$ **do**
5 Let j be the first demand in $\mathcal{J}$;
6 **if** *at least a charger is available at* r_j **then**
7 w_j^a be the first available charger in $\mathcal{M}$;
8 Let b be the number of time slots in $[r_j, d_j)$ where $w_G^t + w_j^a \leq w_G$;
9 $E_j \leftarrow e_j/(b \times \tau)$;
10 **if** *the vehicle* j *can be scheduled on an available charger* i *with a charging power* $w_i \geq w_j$ *without exceeding* w_G **then**
11 Schedule j on charger i and remove it from $\mathcal{J}$;
12 **else**
13 Let $e(j', r_j, d_j)$ be the allocated energy to charging demand $j' \neq j$ in the interval $[r_j, d_j)$;
14 Let k be the scheduled demand with $\max_{j'} e(j', r_j, d_j)$;
15 **if** $e(k, r_j, d_j) > e_j$ **then** Reject k ;
16 **else** Reject j and remove it from $\mathcal{J}$
17 **end**
18 **else**
19 Let j' be the scheduled charging demand with maximum $d_{j'}$;
20 **if** $d_{j'} > d_j$ **then** Reject j' ;
21 **else** Reject j and remove it from $\mathcal{J}$
22 **end**
23 Update w_G^t;
24 **end**

7 Simulated Annealing Metaheuristic

7.1 Solution Representation

Solving the scheduling problem consists of two main decisions: first, selecting electric vehicles to be plugged into chargers; then, selecting vehicles to charge by choosing the appropriate time slots for charging without exceeding w_G. Therefore, a solution to the charging scheduling problem consists of the assignment solution and the power allocation solution. We represent the assignment of charging demands to chargers as a vector $(\pi_1, .., \pi_m)$ where π_i is the sequence of vehicles assigned to a charger i and we place the unassigned demands in a list of rejected demands. The power allocation solution is represented with a vector $(T_j)_{j \in \mathcal{J}}$ where $T_j \subseteq H$ stores, for each vehicle j, the time slots chosen for charging process. For convenience, we define the vector $(w_G^t)_{t \in H}$, which stores the minimum grid capacity at each time slot t.

7.2 Simulated Annealing

The simulated annealing (SA) algorithm, initially proposed by [7], is a stochastic local search metaheuristic successfully adapted to address several scheduling problems. In this paper, a candidate solution for the SA algorithm represents the assignment of charging demands to chargers, on which an iterated local search (ILS), described in Sect. 7.4, is applied to get the power allocation vector and the objective function value of each generated solution. The detailed procedure of the implemented SA is presented in Algorithm 2. It starts by taking as input an initial solution (S_0), generated using the heuristic detailed in Sect. 6, and five parameters: the maximum number of generated neighbors at each iteration ($maxGenerated$), the acceptation ratio at each iteration ($acceptanceRatio$), the final temperature (T_f), the maximum global number of generated solutions ($maxTrials$), and the parameter for initializing the value of the temperature (μ). First, the initial solution S_0 is set as the current solution S and as the global best solution S_{best} (line 1). The temperature parameter T is initially set to a value proportional to the objective function value of the initial solution $T = \mu f(S_0)$. The maximum number of accepted solutions at each iteration ($maxAccepted$) is initially set proportionally to the parameter ($maxGenerated$) (line 2). At each iteration (lines 3-19), SA generates neighbors of the current solution S until reaching either ($maxGenerated$) or ($maxAccepted$). We detail the neighborhood generation in Sect. 7.3. For each new solution S', the global number of generated solutions ($trial$) and the number of generated neighbors of S' ($generated$) are incremented (lines 8-9). The objective function value of each solution, i.e., the number of scheduled demands, is referred to by $f(S)$, and it is calculated by the ILS procedure given in Sect. 7.4. The gap between the objective values of the new solution S' and the current solution S is calculated as $\Delta f = f(S') - f(S)$. The neighbor S' is accepted and replaces the current solution based on the Metropolis criteria (lines 10-16); the new solution S' replaces the current solution if there is an improvement, i.e., $\Delta f > 0$. If S' improves the

best solution found so far, it will become the new global best solution S_{best}. Otherwise, a random number is generated following the uniform distribution $U[0,1]$ and the neighbor S' will become the current solution if $U(0,1) \leq e^{\Delta f/T}$ where T is the temperature parameter that controls the probability of accepting worse solutions. For each accepted solution, the parameter *accepted* is incremented (line 12). Finally, a cooling scheme gradually decreases the temperature at each iteration (line 16). We consider the Lundy-Mees cooling scheme proposed by [9]. It updates the temperature T at each iteration l as $T_{l+1} = \frac{T_l}{a+bT_l}$. Connolly in [2] develops a variant of the Lundy-Mees scheme that set the parameter a to 1 and b in function of the initial temperature T_0, the final temperature T_f and the size of the neighborhood M as $b = \frac{T_0 - T_f}{MT_0T_f}$. Here, the number of iterations is not fixed directly. In fact, if we omit the condition on $maxAccepted$, the number of iterations will be equal to $\frac{maxTrials}{maxGenerated}$. Thus, we set M to this value (line 1). After updating the temperature, the number of generated neighbors (*generated*) and the number of accepted solutions (*accepted*) are reset to zero (line 4). The algorithm will stop if the number of generated solutions (*trial*) reaches its maximum ($maxTrials$), or after generating ($maxGenerated$) solutions that did not result in accepted solutions, i.e. $accepted = 0$ (line 17). When the stopping criterion is met, the algorithm terminates and returns the best solution S_{best} found so far.

Algorithm 2: Simulated annealing

input : S_0, $maxGenerated$, $acceptanceRatio$, T_f, $maxTrials$, μ
output: Best solution found S_{best}

1 $S_{best} \leftarrow S_0$, $S \leftarrow S_0$, $T \leftarrow \mu f(S_0)$, $M \leftarrow \frac{maxTrials}{maxGenerated}$, $trial \leftarrow 0$;
2 $maxAccepted \leftarrow acceptanceRatio \times maxGenerated$;
3 **repeat**
4 $accepted \leftarrow 0$; $generated \leftarrow 0$;
5 **while** $generated \leq maxGenerated$ ***and*** $accepted \leq maxAccepted$ **do**
6 $S' \leftarrow Neighbor(S)$;
7 $\Delta f \leftarrow f(S') - f(S)$;
8 $generated \leftarrow generated + 1$;
9 $trial \leftarrow trial + 1$;
10 **if** $\Delta f > 0$ *or* $U(0,1) \leq e^{\Delta f/T}$ **then**
11 $S \leftarrow S'$;
12 $accepted \leftarrow accepted + 1$;
13 **if** $f(S) > f(S_{best})$ **then** $S_{best} \leftarrow S$;
14 **end**
15 **end**
16 $T \leftarrow \frac{T}{1+bT}$;
17 **until** $trial \leq maxTrials$ ***and*** $accepted > 0$;
18 **return** S_{best}

7.3 Neighborhood Operators

The SA algorithm randomly chooses one of three operators to generate a new solution:

- **Change assignment:** this operator chooses a charging demand j on a charger i_1 and moves it to another charger i_2. The chargers and the charging demand are randomly selected. If a charging demand in charger i_2 overlaps with j, the move is discarded.
- **Assign a rejected charging demand:** this operator chooses a charging demand j from the rejected list and inserts it on a charger i. The charger and the charging demand in the rejected list are randomly selected. The move is discarded if at least one charging demand in charger i overlaps with j.
- **Reject a charging demand:** this operator moves a charging demand from a charger to the rejected list. The charger and the charging demand are randomly selected.

When a move is discarded, the SA algorithm randomly selects another operator. After each successful move, the SA algorithm applies an ILS procedure to construct and improve the power allocation solution.

7.4 Iterated Local Search

Given an assignment solution, the iterated local search (ILS) procedure will solve the power allocation problem by selecting the maximum subset of scheduled demands from the assignment solution that can be satisfied without exceeding the grid capacity w_G. The assignment solution may or not be feasible, i.e., the grid capacity w_G may not be sufficient. Let $\mathcal{J}'$ be the set of assigned charging demands. Let $\tilde{w}_G$ be the minimum grid capacity required to satisfy all charging demands in the set $\mathcal{J}'$. The basic idea is to obtain a charging schedule with the minimum value of $\tilde{w}_G$. When $\tilde{w}_G > w_G$, we insert and reject charging demands until $\tilde{w}_G$ reaches w_G. Note that we can only insert the demands rejected by the ILS procedure. Moreover, each demand can only be reinserted in their previously assigned charger, meaning that we cannot move a charging demand to another charger. Therefore, we keep a list L_{LS} of rejected demands by ILS along with their previous chargers. The implemented ILS algorithm (Algorithm 3) starts by building the power allocation vectors of an assignment solution S_0 using a heuristic described in Algorithm 4 (line 1). The current solution S is set to S_0. At each iteration (line 3-23), the ILS procedure generates $maxGeneratedLS$ neighbors of the current solution S (line 5-12). For each generated neighbor, it applies a procedure to minimize $\tilde{w}_G$ (line 7) that will be described below. The best feasible solution S' in the neighborhood of S is selected. A solution is feasible if its grid capacity $\tilde{w}_G$ is less than or equal to w_G. If the best neighborhood S' is better than the best solution found so far S^*, it will replace the current solution S and the best solution S^*. Otherwise, the number of non-improving iterations $iter$ is incremented (line 21). In this case, the current solution S is set to either the best solution in the neighborhood S' or to S^*. The best solution in the neighborhood S' may replace the current solution S if

a randomly generated number u is less than the probability p_{iter} (line 17-22). p_{iter} decreases in a geometric way [11] and is calculated as follows. $p_{iter} = p_0 \times r^{iter-1}$, Where p_0 is the initial acceptance probability, $r < 1$ is the reducing factor, and $iter$ is the number of iterations. When the number of non-improving iterations $iter$ exceeds $maxNonImproving$, the search is considered as stagnating on a local optimum and is subsequently terminated.

Algorithm 3: Iterated local search procedure

Input : The assignment solution S_0, $maxNonImproving$, $maxGeneratedLS$, r, p_0
Output: Best feasible solution found S^*

```
1  Initialize the power allocation for S_0 according to Algorithm 4;
2  iter ← 0; S ← S_0; S* ← empty solution;
3  while iter < maxNonImproving do
4      S' ← empty solution;
5      for k = 1 to maxGeneratedLS do
6          N ← Local Neighbor(S);
7          Apply minimizing grid capacity procedure to N;
8          if w̃_G(N) ≤ w_G and f(N) > f(S') then
9              if S* is empty then S* ← N ;
10             S' ← N;
11         end
12     end
13     if f(S') > f(S*) then
14         S ← S';
15         S* ← S';
16         iter ← 0;
17     else
18         Generate a random number u ~ U(0, 1);
19         if u < p_0 × r^(iter−1) then S ← S' ;
20         else S ← S* ;
21         iter ← iter + 1
22     end
23 end
24 return S*
```

Initial Solution for Power Allocation. Let $\mathcal{J}'$ be the set of vehicles in the assignment solution vector. Let w_j be the charging power of each vehicle $j \in \mathcal{J}'$. Then, the charging time p_j of each demand j can be calculated as $\lceil e_j/w_j \rceil$. The proposed heuristic, detailed in Algorithm 4, builds the power allocation solution for the set $\mathcal{J}'$ by considering the assigned vehicles in the non-decreasing order of their departure time d_j, and break ties first by non-increasing order of their energy request e_j, then by non-increasing order of their arrival time r_j (line 1). The grid capacity $\tilde{w}_G$ and power allocation vectors are initialized to 0 (line 2). The power allocation heuristic starts by charging vehicle j at time slots without exceeding $\tilde{w}_G$ in chronological order (lines 4, 6–15). Then, on time slots with the minimum w_G^t value (lines 5, 6–15).

Algorithm 4: Power allocation heuristic

Input : The set of charging demands $\mathcal{J}'$, the selected charging power w_j and the charging time p_j for each vehicle
Output: The vectors $(T_j)_{j\in\mathcal{J}}$, $(w_G^t)_{t\in H}$, the grid capacity $\tilde{w}_G$

1 Sort $\mathcal{J}'$ by non-decreasing order of d_j. Then, in non-increasing order of e_j. Then, in non-increasing order of r_j ;
2 $(w_G^t) \leftarrow (0)_{t\in H}$; $\tilde{w}_G \leftarrow 0$; $(T_j) \leftarrow (\emptyset)_{j\in\mathcal{J}'}$;
3 **for** $j \in \mathcal{J}'$ **do**
4 $H_1 \leftarrow$ the set of time slots t where $t \in [r_j, d_j)$ and $\tilde{w}_G \geq w_j + w_G^t$ sorted in chronological order;
5 $H_2 \leftarrow$ the set of time slots t where $t \in [r_j, d_j)$ and $t \notin H_1$ sorted in non decreasing order of w_G^t;
6 **while** $p_j > 0$ **do**
7 **if** $H_1 \neq \emptyset$ **then** $H_i \leftarrow H_1$;
8 **else** $H_i \leftarrow H_2$;
9 Let t be the first time slot of H_i;
10 Remove t from H_i ;
11 $T_j \leftarrow T_j \cup \{t\}$;
12 $w_G^t \leftarrow w_G^t + w_j$;
13 $p_j \leftarrow p_j - 1$;
14 **if** $w_G^t > \tilde{w}_G$ **then** $\tilde{w}_G \leftarrow w_G^t$
15 **end**
16 **end**
17 **return** $\tilde{w}_G$, $(T_j)_{j\in\mathcal{J}}$, $(w_G^t)_{t\in H}$

Local Neighbor Structure. In the ILS procedure, a neighbor is generated by one of the following operators:

- **Reject** this operator removes one or multiple charging demands from a charger to the rejected list L_{LS}. We implements three methods to select a vehicle to reject. First, a randomly chosen vehicle. Second, reject the vehicle j with the greatest value v_j where $v_j = \sum_{t\in T'} w_G^t$ where $T' = \{t \in T_j \text{ and } w_G^t > w_G\}$. Third, calculate the value v_j for all scheduled vehicles and then a roulette wheel selection [8] is performed i.e., a vehicle j with a higher value v_j has a higher probability to be chosen. After rejecting a vehicle, the w_G^t is updated.
- **Reinsert** this operator randomly chooses one or more vehicles from L_{LS} to be assigned back to its charger. The power allocation for inserted vehicle is obtained using the same procedure in Algorithm 4 (lines 4-21).

Minimizing Grid Capacity Procedure. We use a SA algorithm similar to Algorithm 2 but with different objective function $f(S)$ and different neighbor structure. This second SA is denoted by MINWG-SA. The objective of MINWG-SA is to try to reschedule the charging operations so that the minimum grid capacity $\tilde{w}_G$ is minimized. Since Algorithm 2 is a maximization algorithm, in the MINWG-SA, we replace line 10 by $\Delta f < 0$ or $U(0,1) \leq e^{-\Delta f/T}$. Also,

line 13 is replaced by $f(S) < f(S_{best})$. A neighbor structure in the minimizing grid capacity local search method moves a charging operation of a vehicle j from a time slot $t_1 \in T_j$ to another time slot $t_2 \notin T_j$. Let $\mathcal{J}'$ be the set of scheduled charging demands where $d_j - r_j - p_{ij} > 0$, where p_{ij} is the charging time of vehicle j on its assigned charger type i. First, we randomly select an electric vehicle $j \in \mathcal{J}'$ and two time slots t_1 and t_2, where t_1 is a time slot with $w_G^{t_1} = \min_{\{t \in T_j\}} w_G^t$, and t_2 is a time slot with $w_G^{t_2} = \max_{\{t \notin T_j\}} w_G^t$. Then, the charging operation of vehicle j is moved from time t_1 to t_2 by deleting t_1 from T_j and adding t_2 to T_j. This procedure is repeated k times for the same vehicle, where k is randomly selected in $\{1, \ldots, p_{ij}\}$. After each move, the vector w_G^t is updated as well as the objective value $\tilde{w}_G$.

8 Simulation Results

The proposed algorithms are implemented in C++, and run on a desktop computer with an Intel Core i5, 2.9 GHZ CPU and 8 GB RAM. The ILP model is solved using CPLEX 12.8. In the following, we present our experimental results on randomly generated instances.

We consider five groups of instances with different number of charging demands $n \in \{10, 40, 50, 100\}$, different number of chargers $m \in \{15, 24, 27, 30\}$, and different power grid capacities $w_G \in \{50, 75, 100, 125\}$. For each group, one third of chargers deliver a power $w_1 = 11$(kW), one third of chargers deliver a power $w_2 = 22$ (kW), and the remaining third of chargers deliver a power $w_3 = 43$ (kW). For each group, we generate 10 different random instances as follows. The arrival times of vehicles are generated from the uniform distribution in the interval $[0, 0.2n]$ (in hours). The required energy are generated from the uniform distribution $[5.5, 66]$ (in kWh). To generate the departure times of vehicles, we first calculate the charging times p_{1j} (in hours) for each vehicle j $\in \mathcal{J}$ assuming that it can be charged with chargers of type 1 (11 kW). Then, the departure time of each vehicle j is calculated as $d_j = r_j + (1 + \alpha)p_{1j}$, where α is randomly chosen according to the value p_{1j} as follows. For p_{1j} in [0.5, 1], [1, 2], [2,3], [3,4], [4,5], and [5,6] α is randomly chosen in [0.1, 1], [0.1, 0.9], [0.1, 0.8], [0.1, 0.7], [0.1, 0.6], and [0.1, 0.5], respectively. On the basis of preliminary experiments, we set the parameters μ, $maxGenerated$, $maxTrials$, $acceptanceRatio$, and T_f to 0.12, 50, 100, 0.1, and 0.001 respectively. For the LS procedure, we set the parameters $maxNonImproving$, $maxGeneratedLS$, the reducing factor r and the initial acceptance p_0 to 5, 5, 0.75, and 0.2 respectively.

We set the maximum computation time of CPLEX to 30 min. Table 1 provides a comparison of results obtained for the four groups of instances. The first column denotes the instance number in the group. For CPLEX and the heuristic, column "scheduled" reports the objective value found, and column "time" displays the total running time in seconds. Due to the stochastic nature of the SA algorithm, ten independent executions were done for each instance. We report the best, the worst, and the average objective function value over the ten runs in columns "best", "worst", and "average", respectively. Also, we report the standard deviation of the mean objective function value in column "std" and the average running time in column "time".

Table 1. Comparison results between CPLEX, the heuristic, and the SA algorithm.

instance	**CPLEX**		**Heuristic**		**SA**				
	scheduled	time (s)	scheduled	time (s)	best	worst	average	std	time (s)
group 1 with $n = 10$, $m = 15$, and $w_G = 50$									
1	10	6.07	7	5.45E-05	10	10	10	0.00	3.56
2	9	1800.66	7	3.89E-05	9	9	9	0.00	23.12
3	9	1800.38	7	2.67E-05	9	9	9	0.00	19.54
4	10	3.98	7	8.70E-05	10	9	9.4	0.52	17.49
5	9	537.35	9	1.96E-05	9	9	9	0.00	21.66
6	10	18.17	6	4.74E-05	10	8	9.6	0.70	12.94
7	9	1800.48	7	3.17E-05	9	8	8.3	0.48	24.85
8	9	742.27	7	3.50E-05	9	8	8.7	0.48	24.43
9	9	46.03	6	5.14E-05	9	8	8.1	0.32	22.92
10	9	1800.66	7	3.21E-05	9	8	8.6	0.52	23.82
Average	9.30	855.60	7.00	4.24E-05	9.30	8.60	8.97	0.30	19.43
group 2 with $n = 40$, $m = 24$, and $w_G = 75$									
1	26	1802.38	27	1.37E-04	30	29	29.1	0.32	27.60
2	26	1802.27	25	2.19E-04	31	29	29.6	0.70	26.04
3	11	1801.35	23	2.08E-04	26	24	25.2	0.63	29.64
4	23	1802.33	19	4.86E-04	24	22	23.3	0.67	31.95
5	25	1801.64	21	2.17E-04	26	24	25.1	0.74	30.58
6	26	1802.17	25	1.34E-04	29	28	28.7	0.48	29.68
7	26	1801.81	26	1.02E-04	30	28	29.7	0.67	29.79
8	24	1801.51	23	1.66E-04	28	27	27.3	0.48	29.87
9	25	1801.89	23	1.38E-04	27	26	26.6	0.52	32.25
10	30	1802.85	30	9.83E-05	33	31	32.1	0.57	28.48
Average	24.20	1802.02	24.20	1.91E-04	28.40	26.80	27.67	0.58	29.59
group 3 with $n = 50$, $m = 27$, and $w_G = 100$									
1	21	1802.28	36	1.88E-04	41	39	40.20	0.79	34.45
2	31	1802.03	37	1.16E-04	44	42	42.70	0.67	34.62
3	16	1801.76	35	1.06E-04	41	38	39.50	0.85	32.63
4	34	1802.02	38	1.45E-04	44	42	42.80	0.63	30.70
5	22	1802.04	41	1.35E-04	45	43	44.10	0.57	33.17
6	24	1801.96	38	1.69E-04	41	40	40.80	0.42	34.59
7	23	1801.92	37	1.99E-04	42	40	41.10	0.74	32.79
8	31	1801.90	34	4.25E-04	39	37	38.00	0.67	35.74
9	33	1801.90	34	1.53E-04	37	34	36.10	0.88	35.74
10	15	1802.46	40	1.71E-04	45	43	44.00	0.67	33.12
Average	25	1802.03	37	1.81E-04	41.9	39.8	40.93	0.69	33.75

(*continued*)

Table 1. (*continued*)

instance	**CPLEX**		**Heuristic**		**SA**				
	scheduled	time (s)	scheduled	time (s)	best	worst	average	std	time (s)
group 4 with $n = 100$, $m = 30$, and $w_G = 125$									
1	18	1806.49	76	6.08E-04	77	76	76.20	0.42	50.57
2	11	1806.12	81	2.39E-03	86	83	84.90	0.88	49.28
3	13	1805.92	76	2.32E-04	80	78	78.90	0.74	55.75
4	14	1806.03	75	3.54E-04	78	76	77.40	0.70	55.62
5	14	1806.01	75	6.24E-04	82	78	80.50	1.18	52.36
6	13	1805.99	77	2.65E-04	82	79	81.00	0.94	55.05
7	13	1805.57	78	2.43E-04	83	81	81.40	0.70	51.40
8	14	1806.07	73	6.58E-04	80	77	78.70	1.16	50.72
9	12	1806.21	74	3.89E-04	82	80	80.40	0.84	51.84
10	14	1806.14	77	3.88E-04	81	77	79.40	1.26	55.16
Average	13.60	1806.06	76.20	6.15E-04	81.10	78.50	79.88	0.88	52.78

First, we can notice that CPLEX found six optimal solutions out of 40, all in group one instances with ten vehicles (instances 1,4, 5, 6, 8, and 9 in group 1). All remaining instances were hard to solve for CPLEX within 30 min. The SA also achieved six optimal solutions. However, it took an average time of 17.16 s, while CPLEX took an average time of 225.64 s. As expected, the SA algorithm outperforms the heuristic since it is set to the initial solution for the SA algorithm. The SA algorithm achieved the best solutions in all instances. We calculate the average gap $Gap_{best}(\%)$ (resp. $Gap_{mean}(\%)$) between the objective values found by CPLEX S_{CPLEX} and the best (resp. mean) objective values found by the SA algorithm S_{SA} as $Gap_{best}(\%) = \frac{S_{CPLEX} - S_{SA}}{S_{SA}}$. The $Gap_{best}(\%)$ values were 0%, −14.75%, −40.33%, and −83.23% for groups 1, 2, 3, and 4, respectively. The $Gap_{mean}(\%)$ values were 3.68%, −12.54%, −38.92%, and −82.97% for groups 1, 2, 3, and 4, respectively. The gap between the SA algorithm and solutions found by CPLEX increases significantly with the size of instances. The heuristic starts performing better than CPLEX in groups 3 and 4. Finally, we compare the proposed methods in terms of running time. As expected, the heuristic is faster than the SA algorithm. The heuristic took less than one millisecond, whereas the SA algorithm took an average running time of 32.44 s. In summary, the SA algorithm outperformed CPLEX in significantly less time.

9 Conclusion

This paper addressed the EVCSP in a charging station with different charging types and limited overall power. We proved that the problem is NP-Hard and we formulate it as an ILP model. It was hard for CPLEX to solve the ILP model within 30 min. Therefore, we designed a heuristic and a SA algorithm

combined with an ILS procedure. We generated different instances to evaluate the performance of the proposed methods. The experimental results underline the efficiency of the proposed methods. We assumed that the data related to vehicle charging demands were known in advance. In future research, we can study the scheduling problem in real-time to handle charging demands with or without reservations. Another challenge is considering multi-objective optimization to add the objective of minimizing the charging costs.

References

1. Ocpp 2.0 - 2.0.1 specification. Tech. rep., the Open Charge Alliance (OCA). https://www.openchargealliance.org (2020)
2. Connolly, D.T.: An improved annealing scheme for the qap. Eur. J. Oper. Res. **46**(1), 93–100 (1990)
3. Du, J., Leung, J.Y., Wong, C.S.: Minimizing the number of late jobs with release time constraint. J. Comb. Math. Comb. Comput. **11**(97), 107 (1992)
4. García-Álvarez, J., González, M.A., Vela, C.R.: Metaheuristics for solving a real-world electric vehicle charging scheduling problem. Appl. Soft Comput. **65**, 292–306 (2018)
5. Gong, L., Cao, W., Liu, K., Zhao, J.: Optimal charging strategy for electric vehicles in residential charging station under dynamic spike pricing policy. Sustain. Urban Areas **63**, 102474 (2020)
6. IEA: Global EV Outlook 2021: Accelerating ambitions despite the pandemic. International Energy Agency (IEA) (2021). www.iea.org
7. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
8. Lipowski, A., Lipowska, D.: Roulette-wheel selection via stochastic acceptance. Phys. A **391**(6), 2193–2196 (2012)
9. Lundy, M., Mees, A.: Convergence of an annealing algorithm. Math. Program. **34**(1), 111–124 (1986)
10. Niu, L., Zhang, P., Wang, X.: Hierarchical power control strategy on small-scale electric vehicle fast charging station. J. Clean. Prod. **199**, 1043–1049 (2018)
11. Ogbu, F., Smith, D.K.: The application of the simulated annealing algorithm to the solution of the n/m/cmax flowshop problem. Comput. Operat. Res. **17**(3), 243–253 (1990)
12. Rahman, I., Vasant, P.M., Singh, B.S.M., Abdullah-Al-Wadud, M.: On the performance of accelerated particle swarm optimization for charging plug-in hybrid electric vehicles. Alex. Eng. J. **55**(1), 419–426 (2016)
13. Wang, Z., Jochem, P., Fichtner, W.: A scenario-based stochastic optimization model for charging scheduling of electric vehicles under uncertainties of vehicle availability and charging demand. J. Clean. Prod. **254**, 119886 (2020)
14. Wu, H., Pang, G.K.H., Choy, K.L., Lam, H.Y.: Dynamic resource allocation for parking lot electric vehicle recharging using heuristic fuzzy particle swarm optimization algorithm. Appl. Soft Comput. **71**, 538–552 (2018)
15. Yang, S.: Price-responsive early charging control based on data mining for electric vehicle online scheduling. Electric Power Syst. Res. **167**, 113–121 (2019)
16. Zhang, L., Li, Y.: Optimal management for parking-lot electric vehicle charging by two-stage approximate dynamic programming. IEEE Trans. Smart Grid **8**(4), 1722–1730 (2015)