An Evolutionary Approach for Scheduling a Fleet of Shared Electric Vehicles

Steffen Limmer, Johannes Varga, Guenther Raidl

2023

An Evolutionary Approach for Scheduling a Fleet of Shared Electric Vehicles

Steffen Limmer[1] $^{[0000-0003-2385-7886]}$, Johannes Varga[2] $^{[0000-0003-1413-7115]}$, and Günther R. Raidl[2] $^{[0000-0002-3293-177X]}$

[1] Honda Research Institute Europe GmbH, 63073 Offenbach, Germany
steffen.limmer@honda-ri.de

[2] Institute of Logic and Computation, TU Wien, 1040 Vienna, Austria
{jvarga,raidl}@ac.tuwien.ac.at

Abstract. In the present paper, we investigate the management of a fleet of electric vehicles. We propose a hybrid evolutionary approach for solving the problem of simultaneously planning the charging of electric vehicles and the assignment of electric vehicles to a set of reservations. The reservation assignment is optimized with an evolutionary algorithm while linear programming is used to compute optimal charging schedules. The evolutionary algorithm uses an indirect encoding and a problem-specific crossover operator. Furthermore, we propose the use of a surrogate fitness function. Experimental results on problem instances with up to 100 vehicles and 1600 reservations show that the proposed approach is able to notably outperform two approaches based on mixed integer linear programming.

Keywords: Electric vehicles · Scheduling · Evolutionary algorithm · Mixed integer linear programming · eMaaS.

1 Introduction

There is an increasing trend towards electric, shared, and multimodal mobility in order to tackle environmental issues and to respond to current and future transportation needs of users [7]. Services, which integrate multiple transportation modes and shared electric mobility are commonly summarized under the term electric Mobility as a Service (eMaaS) [4]. The operation of an eMaaS service, like, for example, a dial-a-ride platform with electric vehicles (EVs), typically requires planning ahead the usage and recharging of a fleet of electric vehicles [2]. This in turn requires to solve an optimization problem. The runtime of the optimization is a critical aspect, especially in a dynamic setting, in which fast responses to newly arriving transportation requests are required.

In the present paper, we investigate the use of evolutionary computation to accelerate the optimization for such type of scheduling problem: The EV Fleet Charging and Allocation Problem (EVFCAP) [9]. In this problem, a fleet of EVs (e.g., a company car fleet) is considered. The EVs can charge energy at a common site (e.g., a company building), where the energy can be either drawn from the power grid for time-varying electricity prices or from photovoltaics (PV) overproduction, if available, for free. Users (e.g., company employees) can make reservations of vehicles. For each reservation, a period of time, when a vehicle is required, and an estimated energy consumption is given. The problem consists in planning the charging of the EVs simultaneously with an assignment of the individual EVs to the reservations with the objective to maximize the usage of EVs for serving reservations while keeping the charging cost low.

The problem can be formulated as a mixed integer linear programming (MILP) problem, which can be reasonably well solved as long as the number of EVs and reservations is not too high. Betz et al. [1] use a MILP approach to solve a variant of the EVFCAP where only a limited number of heterogeneous charging stations are available for charging the EVs. They report that the largest problem instance, which could be solved to proven optimality, comprises eight EVs and about 30 reservations. Sassi and Oulamara [8] consider a problem variant where the total charging power is limited and reservations, which are not assigned to EVs, have to be assigned to a limited number of combustion engine vehicles. They propose a problem-specific heuristic for the solution of that problem and compare it to a MILP approach on problem instances with up to 200 vehicles and 320 reservations. Especially on larger problem instances, the MILP approach is clearly outperformed by the heuristic approach. Varga et al. [9] consider the same problem variant as considered in the present paper. They use a Benders decomposition approach to speed up the optimization with a MILP approach. The optimization is further accelerated by applying in early stages a general variable neighborhood search heuristic to solve the master problem of the Benders decomposition. The authors compare the proposed approach with a standard MILP approach on problem instances with up to 100 EVs and 1600 reservations.

We propose and evaluate an evolutionary approach for the solution of the EVFCAP. The assignment of reservations to EVs is optimized with an evolutionary algorithm (EA), which is hybridized with a linear programming approach in order to compute optimal charging schedules. To ensure feasibility of solution candidates, the EA uses an indirect encoding that does not encode the reservation assignment directly but only the order in which reservations are passed to an insertion operator. The optimization is supported by a problem-specific crossover operator. Furthermore, a surrogate fitness function is employed in order to accelerate the optimization. In experiments, the proposed approach is evaluated and compared to a standard MILP approach and the improved MILP approach proposed in [9] on publicly available problem instances from [9].

The rest of the paper is organized as follows: Section 2 provides a description of the problem. In Section 3, the proposed approach is explained in detail. Section 4 describes the experiments and discusses their results. Finally, Section 5 provides conclusions.

2 Problem Description

We consider a planning horizon $\mathcal{T} = \{1, \dots, T\}$ of T discrete time steps of an equal length of Δt hours. There is a set $\mathcal{N} = \{1, \dots, N\}$ of N EVs, each with a maximum charging power of $P^{\max}$ kW and with a battery capacity of $E^{\max}$ kWh. At the beginning of the planning horizon, each EV $n \in \mathcal{N}$ has a certain initial battery level of E_n^{init} kWh. The EVs can be charged at a common site. It is assumed that at this site, there is a certain electrical base load (consumption) and energy production by a PV system. If the PV production exceeds the base load, there is a surplus energy, which can be used for EV charging. Let Sur_t denote the amount of surplus energy in time step t. For time steps t, in which the base load is higher than the PV production, Sur_t is zero. In addition to the surplus energy, energy from the power grid can be used for EV charging. It is assumed that the price for grid energy varies over time. Let p_t denote the electricity price per kWh in time step t. There is a set $\mathcal{R} = \{1, \dots, R\}$ of R reservations of vehicles by users. For each reservation $r \in \mathcal{R}$, there is a time period, in which a vehicle is required. Let t_r^s and t_r^e denote the first and last time step, respectively, of this period for reservation r. It is assumed that EVs are not charged externally while they are used for a reservation. Thus, it has to be ensured that EVs are sufficiently charged before they are used for a reservation. Let E_r^{res} denote the amount of energy, which is required for serving reservation r.

An operator of the fleet has to decide for each reservation whether it is assigned to an EV or not and if assigned to an EV, to which one exactly. Unassigned reservations might, e.g., be served by a fleet of combustion engine cars inducing additional cost. In addition to the reservation assignment, the charging of the EVs has to be planned. It is assumed that the operator of the fleet is interested in three objectives:

1. Minimizing the amount of energy required for reservations that are not assigned to an EV, which corresponds to maximizing the usage of EVs.
2. Minimizing the electricity cost incurred by EV charging.
3. Minimizing the amount of energy missing in the batteries of the EVs at the end of the planning horizon in order to increase the number of reservations that can be served by EVs in the time after the planning horizon.

We introduce binary variables y_r and $x_{n,r}$, where y_r indicates whether a reservation r is assigned to an EV or not and $x_{n,r}$ indicates whether reservation r is assigned to EV n or not. Furthermore, we introduce variables $P_{n,t}$ for the power by which EV n is charged in time step t. Let E_t^{grid} and E_t^{sur} denote the amount of charging energy consumed from the grid and from surplus energy, respectively, in time step t, and let $E_{n,t}$ denote the battery level of EV n in time step t. The scheduling problem can be expressed as the following MILP problem:

$$\min \; \alpha \sum_{r \in \mathcal{R}} E_r^{\text{res}} \cdot y_r + \sum_{t \in \mathcal{T}} p_t \cdot E_t^{\text{grid}} + \beta \sum_{n \in \mathcal{N}} (E^{\max} - E_{n,T}), \tag{1}$$

subject to:

$$\sum_{n \in \mathcal{N}} x_{n,r} + y_r = 1 \qquad \forall r \in \mathcal{R}, \quad (2)$$

$$P_{n,t} \leq P^{\max} \cdot (1 - \sum_{r \in \mathcal{R} | t_r^s \leq t \leq t_r^e} x_{n,r}) \qquad \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \quad (3)$$

$$E_{n,1} = E_n^{\text{init}} + \Delta t \cdot P_{n,1} - \sum_{r \in \mathcal{R} | t_r^s = 1} x_{n,r} \cdot E_r^{\text{res}} \qquad \forall n \in \mathcal{N}, \quad (4)$$

$$E_{n,t} = E_{n,t-1} + \Delta t \cdot P_{n,t} - \sum_{r \in \mathcal{R} | t_r^s = t} x_{n,r} \cdot E_r^{\text{res}} \qquad \forall n \in \mathcal{N}, \forall t \in \mathcal{T} \setminus \{1\}, \quad (5)$$

$$\sum_{n \in \mathcal{N}} \Delta t \cdot P_{n,t} = E_t^{\text{grid}} + E_t^{\text{sur}} \qquad \forall t \in \mathcal{T}, \quad (6)$$

$$0 \leq P_{n,t} \leq P^{\max} \qquad \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \quad (7)$$

$$0 \leq E_{n,t} \leq E^{\max} \qquad \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \quad (8)$$

$$0 \leq E_t^{\text{grid}} \qquad \forall t \in \mathcal{T}, \quad (9)$$

$$0 \leq E_t^{\text{sur}} \leq Sur_t \qquad \forall t \in \mathcal{T}, \quad (10)$$

$$x_{n,r} \in \{0,1\}, y_r \in \{0,1\} \qquad \forall n \in \mathcal{N}, \forall r \in \mathcal{R}. \quad (11)$$

The objective function is a weighted sum of the three objectives, with weights α and β. The weight α could be, for example, set to the cost arising from traveling the average distance corresponding to the consumption of one kW with a combustion engine vehicle. Constraint (2) ensures that each reservation $r \in \mathcal{R}$ is either assigned to exactly one EV or to no EV. Constraint (3) ensures that an EV is not charged during time steps in which it is used for a reservation. Furthermore, it ensures that an EV is not assigned to two or more temporarily overlapping reservations, because for time steps in which the reservations overlap, the right-hand side of the constraint would be negative, making the problem infeasible. Constraints (4) and (5) set the battery levels of the EVs after the first and the following time steps, respectively. It is assumed that the energy required by a reservation is consumed in the first time step of the reservation. Together with the lower and upper bounds for the battery levels (8), these constraints ensure that an EV has always a sufficient amount of energy before it is used for a reservation and that EVs cannot be charged higher than technically possible. Constraint (6) ensures that the energy charged in a time step t is consumed from the grid and/or from the available surplus energy.

3 Evolutionary Algorithm

We propose an evolutionary algorithm for the solution of the described EV fleet scheduling problem. More precisely, we apply a hybrid approach, where the assignment of reservations to EVs is optimized with an EA and the charging scheduling is determined via linear programming. Furthermore, a surrogate fitness function is used in order to accelerate the optimization. The following subsections provide a detailed description of the approach.

3.1 Encoding

An individual has to encode the assignment of reservations to EVs. An obvious encoding would be a list of R integer variables $i_1, \dots, i_R$ between zero and N, where reservation r is assigned to no EV if i_r is zero and to EV i_r, otherwise. However, with this encoding it is hard to ensure that the encoded assignment is feasible. Thus, we use an indirect encoding, where the genotype is a permutation of the numbers $1, \dots, R$. To compute the phenotype (the actual reservation assignment), the reservations are passed in the encoded order to an insertion operator. The insertion operator computes a (feasible) reservation assignment in the form of a list $A = [A_1, \dots, A_N]$ of N lists, where the list A_n contains the reservations assigned to EV $n = 1, \dots, N$. The EA makes use of two insertion operators: *basic insertion* and *random insertion*. The basic insertion is outlined in Algorithm 1. It starts with a list of empty lists and then iterates over the

Algorithm 1: Basic insertion operator.

```
Input: list $P$ of reservations
Output: reservation assignment $A$
$A = [[\,], \dots, [\,]]$;
for $r$ in $P$ do
    for $n = 1, \dots, N$ do
        $A' = A[n] + [r]$;
        if feasible($A'$) then
            $A[n] = A'$;
            break; // go to next reservation
        end
    end
end
return $A$;
```

reservations in the order in which they were passed to the operator. For each reservation r it goes through the EVs and inserts r in the list belonging to the first EV for which the insertion does not lead to an infeasible reservation assignment. The assignment of reservations to an EV is infeasible if two or more of the assigned reservations overlap or if it is not possible to satisfy the energy requirements of the reservations with the EV. The latter can be easily determined by checking if the energy requirements are satisfied in the case of uncontrolled charging (i.e., charging the EV in all time steps in which it is not used for a reservation with the maximum possible power until the battery is full). The random insertion operator works similar to the basic insertion operator with the difference that it does not assign a reservation to the first possible EV it finds but to an EV which is randomly selected from all possible EVs.

It is obvious that the order in which the reservations are passed to the insertion operators has a big influence on the resulting reservation assignment.

Reservations at the beginning of the list have a high chance of being inserted, while reservations at the end of the list often cannot be inserted since they overlap with already inserted reservations or lead to infeasible energy requirements. The basic insertion operator is the main insertion operator used in the EA, since it is deterministic. The random insertion operator is only used as part of the initialization process (see Section 3.2). As already stated, the used encoding in form of permutations, which are passed to the basic insertion operator in order to compute the phenotype, has the advantage that an individual always encodes a feasible reservation assignment. However, it has also a disadvantage: It cannot encode all possible feasible reservation assignments. For example, if there are two EVs and three reservations $r1$, $r2$, $r3$, which can be all assigned to the first EV, then the basic insertion operator will always assign the reservations to the first EV, no matter in which order they are passed. However, assigning one or more of the reservations to the second EV or to no EV at all might yield a better objective value. But as we will see from the experimental results, the encoding's advantage seems to outweigh its disadvantage.

3.2 Initialization

As already described in the previous subsection, reservations at the beginning of the list, which is passed to the insertion operator, have a higher chance of being inserted than reservations at the end of the list. Thus, it is preferable to have promising reservations at the beginning of the list. Reservations with high energy requirements and low durations can be considered to be promising. They contribute much to the reduction of the first term of the objective function while retaining a high flexibility in the charging schedule and/or in the insertion of further reservations. Hence, in the initialization of the EA's population, we first sort the reservations in decreasing order of required energy per time step of duration. The resulting list is then used to initialize the individuals as exemplary illustrated in Figure 1 for a population of two individuals. For each individual,

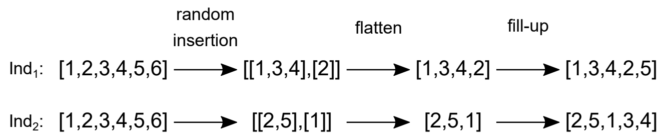

random insertion
flatten
fill-up
Ind1: [1,2,3,4,5,6] → [[1,3,4],[2]] → [1,3,4,2] → [1,3,4,2,5]
Ind2: [1,2,3,4,5,6] → [[2,5],[1]] → [2,5,1] → [2,5,1,3,4]

Fig. 1. Example of the initialization of a population of two individuals Ind_1 and Ind_2.

the sorted list of reservations is passed to the random insertion operator. The resulting reservation assignment is then flattened. The resulting flattened list is then filled up with the reservations that are not already in the list in decreasing order of energy requirement per time step of duration. However, in the fill-up, not all reservations are considered but only reservations which were inserted for at least one individual. In the example, reservation 6 is not considered in the individuals. This means, the optimization will never assign this reservation to an EV. It can be assumed that reservations which were not inserted at least once are likely to be also not assigned to an EV in the optimal solution. By excluding them from the optimization (and considering them as unassigned from the beginning), a notable reduction of the search space can be achieved. For the sake of simplicity, we assume in the rest of the paper that always the whole set $\mathcal{R}$ of reservations is considered in the individuals.

3.3 Fitness Evaluation

In order to evaluate an individual, the encoded permutation of reservations is passed to the basic insertion operator as described in Section 3.1. Given the resulting reservation assignment, the charging of the EVs is optimized with respect to the second and third term of the objective function (1). Optimizing the charging for a fixed reservation assignment is a purely continuous problem, which can be efficiently solved with linear programming. The resulting charging schedule and the reservation assignment are then used to compute the overall objective.

3.4 Crossover

The crossover operator produces an offspring individual from two parent individuals. There are different standard crossover operators for permutations, like OX, PMX or alternating position crossover, which are popular for applications like the traveling salesman problem [6]. However, in preliminary experiments we noticed that such operators do not work well for the given EV fleet scheduling problem. Instead, we use the following approach for the crossover: We select four random integers s_1, e_1, s_2, e_2 with $1 \leq s_1 < e_1 < s_2 < e_2 \leq R$. The reservations between the positions s_1 and e_1 in the first parent are copied into the offspring. Then, the reservations between the positions s_2 and e_2 in the second parent, which are not already in the offspring, are copied into the offspring. Finally, the gaps in the offspring are filled up with the reservations which are not already in the offspring in decreasing order of energy requirement per time step of duration. This is illustrated in Figure 2, where it is assumed that the energy requirement per time step of duration of a reservation r_1 is greater than or equal to that of an reservation r_2 iff $r_1 < r_2$. First, the reservations 4, 6, and 5 are copied from

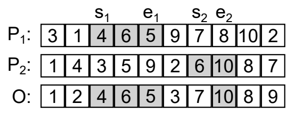


Fig. 2. Example for the crossover of two parents P_1, P_2 to an offspring O. It is assumed that the energy requirement per time step of duration of a reservation r_1 is greater than or equal to that of an reservation r_2 iff $r_1 < r_2$.

the first parent into the offspring. Then reservation 10 is copied from the second parent into the offspring. Reservation 6 is not copied from the second parent since it is already in the offspring. Then the first so far unset position of the offspring is set to reservation 1, since from the reservations, which are not already in the offspring, this is the reservation with the highest energy requirement per time step of duration. Finally, analogously, the remaining unset positions are set to reservations 2, 3, 7, 8, and 9.

3.5 Mutation

As mutation, two basic operations are applied: An exchange of two random reservations in the permutation and a shift of a random reservation to another position in the permutation. The complete mutation operator is outlined in Algorithm 2. With a probability of p_{swap}, reservations are swapped and otherwise

Algorithm 2: Mutation operator.

```
Input: list $P$ of reservations
Output: modified list $P'$ of reservations
1  $P' = P$;
2  $p = $ uniform_rand(0.0,1.0);
3  if $p < p_{\text{swap}}$ then
4      $n_{\text{swap}} = n_{\text{swap\_start}}$;
5      $p = $ uniform_rand(0.0,1.0);
6      while $p < p_{\text{n\_swap}}$ do
7          $n_{\text{swap}} = n_{\text{swap}} + 1$;
8          $p = $ uniform_rand(0.0,1.0);
9      end
10     perform $n_{swap}$ swaps of random pairs of reservations in $P'$;
11 else
12     $n_{\text{shift}} = n_{\text{shift\_start}}$;
13     $p = $ uniform_rand(0.0,1.0);
14     while $p < p_{\text{n\_shift}}$ do
15         $n_{\text{shift}} = n_{\text{shift}} + 1$;
16         $p = $ uniform_rand(0.0,1.0);
17     end
18     perform $n_{\text{shifts}}$ shifts of random reservations to random positions in $P'$;
19 end
20 return $P'$;
```

reservations are shifted. If reservations are swapped, the number of swaps is determined randomly. A certain minimum number $n_{\text{swap\_start}}$ of swaps is always executed. With probability $p_{\text{n\_swap}}$, $n_{\text{swap\_start}} + 1$ or more swaps are executed, with probability $p^2_{\text{n\_swap}}$, $n_{\text{swap\_start}} + 2$ or more swaps are executed, and so on. Analogously, the number of shifts is randomly determined based on parameters $n_{\text{shift\_start}}$ and $p_{\text{n\_shift}}$.

3.6 Optimization Process

We apply a steady-state generational scheme, where in each generation one offspring is generated and is considered for insertion in the population. With a certain crossover probability *pc*, the offspring is produced by crossover of two parent individuals and with probability $1 - pc$ the offspring is produced by copying a parent individual. Parent individuals are selected with binary tournament selection without replacement. The offspring is then mutated. If one or both of the parameters $n_{\text{swap\_start}}$ and $n_{\text{shift\_start}}$ of the mutation operator are set to zero, it might happen that the offspring is not changed by the mutation operator. Hence, if an offspring was produced by copying a parent, we repeat its mutation until at least one swap or shift was executed. After the mutated offspring is evaluated, it replaces the so far worst individual in the population, if the offspring is better and if there is not already another individual with the same fitness in the population (to increase the diversity in the population).

3.7 Surrogate-assisted Optimization

As outlined in Section 3.3, in the fitness evaluation the EV charging is optimized. Although this optimization can be done very efficiently with the help of linear programming, the fitness evaluation is responsible for a large fraction of the runtime of the whole optimization process. We apply a surrogate model – i.e., a fast approximation of the fitness function to accelerate the optimization. Popular surrogate models used in the context of evolutionary optimization are data-driven models like the Kriging model, radial basis functions or support vector machines [3]. However, in our case we do not have to rely on a data-driven approach. Instead, we can use a fast heuristic to set the charging powers and use the objective with the resulting charging schedule as a surrogate for the real fitness with globally optimal charging scheduling. We apply the following simple heuristic: We assume that the EVs are charged uncontrolled, i.e. with maximum possible power, when they are not used for reservations and are not fully charged already. The objective value with uncontrolled charging should be already a reasonable indicator for the real objective value with optimized charging. More precisely, it provides an upper bound for the real objective value.

There are different options for integrating the surrogate in the evolutionary optimization. Popular strategies are the generation-based strategy and the individual-based strategy [5]. In the generation-based strategy, the surrogate is used in some generations and in the rest of the generations, the real fitness function is used. In the individual-based strategy, in each generation it is determined with help of the surrogate, which offspring individuals are evaluated with the real fitness function. In the experiments described later, we compare the standard version of the EA without any surrogate (*EA*) with two surrogate-assisted versions (*EA-SI* and *EA-SG*). The *EA-SI* variant uses an individual-based strategy. In each generation, N^{inter} intermediate offspring individuals are generated and are evaluated with the surrogate fitness function. Only the best of these individuals (in terms of the surrogate fitness) is then evaluated with the real fitness function and is considered for insertion in the population. In the *EA-SG* variant, an extreme case of the generation-based strategy is applied: The whole optimization works only on the surrogate fitness and only the best individual (in terms of the surrogate fitness) of the final population is evaluated with the real fitness.

4 Experiments

4.1 Experimental Setup

We executed the experiments on a compute cluster. Each process was run on a separate node with an Intel(R) Xeon(R) E5-2623@3.00GHz 8-core CPU and 64 GB RAM. We use a single-threaded C/C++ implementation of the evolutionary algorithm. We compare the EA to a standard MILP approach, which solves the complete model (1–11), and to the improved MILP approach proposed in [9]. The improved approach (denoted as BDH) applies a Benders decomposition to split the problem into a master problem and a subproblem, which are iteratively solved in an alternating manner. In early stages of the optimization process, the master problem is solved with a general variable neighborhood search heuristic. For the solution of the subproblem as well as the master problem in later stages, MILP is used. For all MILP optimizations and for the charging optimization in the fitness evaluation of the EA, version 9.1 of the Gurobi solver is used and the number of threads for the solver is set to one. For the BDH approach, we use the same Julia implementation as in [9]. The parameters of the BDH approach are set to the same values used in the experiments in [9]. To achieve a fair comparison between the EA and the MILP approaches, which do not exhibit a number of fitness evaluations, we use a time limit as termination condition for the optimizations.

Depending on the concrete use case, there might be different runtime requirements. Hence, we observe the optimization results after 5 min, 15 min and 1 hour of runtime. We evaluate the different approaches on artificially created problem instances from [9][3]. The instances have the naming scheme tmax$T$_nN_rmax$R$_I, where T is the number of time steps of the planning horizon, N is the number of EVs, R is the number of reservations and I is the index of the problem instance between 1 and 30. The length Δt of a time step is assumed to be 15 min. EVs can be charged with a maximum power of 3.3 kW and have a battery capacity of 20 kWh. See [9] for more details to the problem instances. For different problem sizes, i.e. different values of T, N, and R, there are 30 instances per size. In the experiments we use the first instance, i.e. with index $I = 1$, for each problem size with $T = 768$ time steps. These are the largest problem sizes considered in [9]. We are not interested in instances of small size, since these can be efficiently solved with a standard MILP approach. In the experiments we execute 21 optimization trials per problem instance with each EA variant. The setting of the parameters of the EA is discussed in the following subsection.

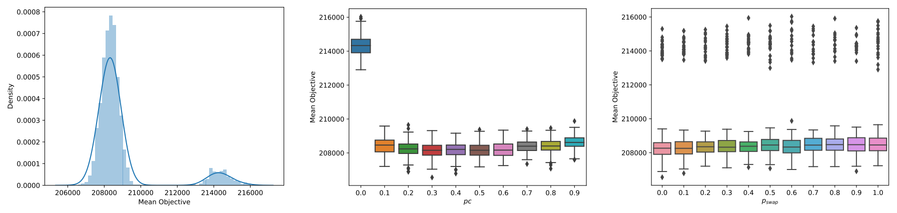
[3] The problem instances are publicly available at `https://www.ac.tuwien.ac.at/research/problem-instances/#evfcap`

4.2 Parameter Setting and Analysis

Based on preliminary experimental results, we set the population size to 100 and the number N^{inter} of intermediate offspring per generation in the *EA-SI* variant to 20. In order to investigate the influence of different settings of the crossover rate pc and of the parameters of the mutation operator on the optimization performance, we drew 2200 random settings of these parameters and performed five optimization trials on the problem instance tmax768_n020_rmax0320_30 with each of these parameter settings. Please note that this problem instance is not part of the evaluation instances used in the experiments described later. The optimizations were done with the standard *EA* variant with a time limit of 1 min per trial. The minimum and maximum of the considered parameter ranges can be seen in Table 1. The ranges of continuous parameters are equidistantly discretized with a step size of 0.1. For each of the random parameter settings we computed the mean objective value over the five trials. The lowest mean objective value was obtained with the parameter setting show in Table 1. It yielded a mean objective value of 206,553.86. We applied this parameter setting in all

Table 1. Parameter ranges considered in the sampling and parameter setting yielding the lowest mean objective value.

Parameter	pc	$n_{\text{swap\_start}}$	$n_{\text{shift\_start}}$	$p_{\text{n\_swap}}$	$p_{\text{n\_shift}}$	p_{swap}
Range (min,max)	0.1, 0.9	0, 4	0, 4	0.2, 0.8	0.2, 0.8	0.1, 1.0
Setting	0.3	2	0	0.3	0.4	0.0



(a) (b) (c)

Fig. 3. (a) Distribution of mean objective with evaluated parameter settings. (b) Boxplot of mean objective values with different settings for the crossover rate pc. (c) Boxplot of mean objective value with different settings for the probability p_{swap}.

EA variants in the subsequent experiments. The distribution of the mean objective over all evaluated parameter settings can be seen in Figure 3(a). There are two peaks, one around 208,000 and one around 214,000. The first arises from parameter settings with a crossover rate $pc > 0$ and the second from parameter settings with $pc = 0$. This can be seen from the boxplot of the mean objective with different values for pc in Figure 3(b). Thus, the crossover has a clearly positive effect on the optimization. The best results are achieved with crossover rates between 0.3 and 0.5. The probability p_{swap} is zero in the best found parameter setting. That means, only shifts and no swaps are performed in the mutation operator. However, in the results of the parameter tuning there is no clear trend towards any best setting for p_{swap} as can be seen from the boxplot in Figure 3(c). The same holds for the other parameters of the mutation operator. Thus, the EA seems not very sensitive regarding the setting of these parameters.

4.3 Experimental Results

Table 2 shows the objective values achieved with the standard MILP approach (*MILP*), the improved MILP approach from [9] (*BDH*) and the EA variants (*EA*, *EA-SI*, and *EA-SG*) on the different problem instances after a runtime of 5 min, 15 min, and 60 min.

Table 2: Objective values obtained by the different approaches on different problem instances after 5 min, 15 min, and 60 min runtime. The results of the EA variants are averages over 21 trials.

Time Limit [min]	***MILP***	***BDH***	***EA***	***EA-SI***	***EA-SG***
tmax768_n020_rmax0080_01					
5	**11,365.78**	11,580.85	11,445.63^{2,3}	11,521.20^{3}	11,990.64
15	**11,358.29**	11,417.99	11,402.05^{2,3}	11,480.91^{3}	12,003.68
60	**11,358.29**	11,396.97	11,381.07^{2,3}	11,441.68^{3}	12,009.36
tmax768_n020_rmax0160_01					
5	55,066.20	45,734.72	44,794.47	**43,685.21**^{1}	43,749.39^{1}
15	50,240.80	44,684.07	43,964.40	**43,414.14**^{1,3}	43,627.42^{1}
60	43,716.90	44,462.71	43,495.71	**43,192.74**^{1,3}	43,568.40
tmax768_n020_rmax0320_01					
5	N/A	215,110.85	208,850.95	205,906.86^{1}	**205,210.12**^{1,2}
15	223,243.76	210,163.65	206,598.62	205,109.22^{1}	**204,693.99**^{1}
60	217,205.15	207,150.76	205,208.35	**204,219.44**^{1}	204,301.36^{1}
tmax768_n050_rmax0200_01					
5	N/A	30,193.53	**28,007.51**^{2,3}	28,138.48^{3}	28,762.28
15	**27,222.70**	30,131.23	27,778.85^{2,3}	27,967.79^{3}	28,735.96
60	**27,069.16**	27,913.11	27,506.53^{2,3}	27,727.19^{3}	28,789.72
tmax768_n050_rmax0400_01					
5	N/A	N/A	113,234.86	110,101.28^{1}	**109,974.88**^{1}
15	N/A	110,061.76	111,412.08	**108,826.50**^{1}	109,407.65^{1}

60	N/A	110,028.34	109,559.97	**107,634.57**^{1,3}	108,990.14^{1}
tmax768_n050_rmax0800_01					
5	N/A	N/A	514,038.25	508,647.80^{1}	**506,261.09**^{1,2}
15	N/A	N/A	510,606.16	505,195.21^{1}	**503,542.10**^{1,2}
60	N/A	510,663.74	505,904.61	502,171.33^{1}	**501,239.37**^{1,2}
tmax768_n100_rmax0400_01					
5	N/A	N/A	56,426.29	**56,208.45**^{1,3}	56,451.73
15	N/A	59,518.77	55,974.64	**55,765.65**^{1,3}	56,314.65
60	N/A	57,847.22	55,394.08^{3}	**55,150.19**^{1,3}	56,296.16
tmax768_n100_rmax0800_01					
5	N/A	N/A	242,928.36	237,856.31^{1}	**235,740.16**^{1,2}
15	N/A	N/A	239,679.03	234,452.57^{1}	**233,053.13**^{1,2}
60	N/A	232,561.19	236,538.11	230,798.55^{1}	**230,196.62**^{1}
tmax768_n100_rmax1600_01					
5	N/A	N/A	1,014,829.75	1,009,492.52^{1}	**1,005,598.19**^{1,2}
15	N/A	N/A	1,010,564.52	1,004,557.84^{1}	**1,001,765.96**^{1,2}
60	N/A	N/A	1,005,903.36	998,538.39^{1}	**996,443.36**^{1,2}

For the EA variants, the shown objectives are averages over the 21 trials per problem instance. The best results per time limit and problem instance are highlighted in bold. An "N/A" denotes that no feasible solution was found. The superscripts 1, 2, and 3 at the results of the EA variants indicate that the results are statistically significantly lower (better) than the corresponding results of the *EA*, *EA-SI*, and *EA-SG* variant, respectively. This was determined with pairwise Wilcoxon rank sum tests with a significance level of 0.05.

On the smallest problem instance, the MILP approach performs very well. In 5 min, it achieves an objective value, which is not achieved by the EA variants in 60 min. On the instance with 50 EVs and 200 reservations, the MILP is not able to find a feasible solution in 5 min, but it yields better results than the EA variants after 15 min and 60 min. On the other instances, the MILP approach is outperformed by the EA. On the five largest instances the MILP approach is not even able to find a feasible solution within 60 min[4]. The BDH approach scales better than the standard MILP approach and finds feasible solutions within one hour for all problem instances except the largest one. However, in most cases it is also outperformed by the EA variants.

On most of the problem instances, the standard EA variant is outperformed by at least one of the surrogate-assisted variants. With the *EA-SG* variant, the

[4] Please note that we use a slightly different MILP problem formulation than [9] (we use helper variables for the battery levels), since we noticed that this yields a better performance. With the formulation from [9] the MILP approach is able to find feasible solutions for the larger instances within 60 min, but only the trivial solutions, where no reservation is assigned to an EV.

results might get worse with a higher runtime since it uses only the surrogate fitness to determine the quality of a solution. One can see a certain trend in the results: On instances with a high total number of reservations or a high number of reservations per EV, the *EA-SG* variant performs best. On instances with a low total number of reservations or a low number of reservations per EV, the standard EA variant yields better results than the other EA variants. However, on these small instances, the MILP approach is also still efficient. On instances with a medium total number of reservations or a medium number of reservations per EV, the *EA-SI* variant performs best. An explanation for this can be derived from Table 3, which shows detailed results on three problem instances computed with the three EA variants with a runtime of one hour. The

Table 3. Detailed results on three problem instances after a runtime of one hour. The results are averages over 21 trials.

EA Variant	**Unserved Cost**	**Energy Cost**	**Battery Cost**	**#Served**	**Generations**
		tmax768_n020_rmax0080_01			
EA	62.3	11,318.8	0.0	78.0	33,956.7
EA-SI	62.3	11,379.4	0.0	78.0	51,032.6
EA-SG	62.3	11,947.1	0.0	78.0	4,445,403.3
		tmax768_n050_rmax0400_01			
EA	41,456.8	65,164.8	2938.4	329.1	20,585.3
EA-SI	39,454.3	65,312.1	2868.2	333.6	16,440.7
EA-SG	38,717.8	67,376.4	2896.0	336.3	476,646.6
		tmax768_n100_rmax1600_01			
EA	790,889.7	198,101.4	16,912.3	887.9	6,516.4
EA-SI	782,347.2	200,970.6	15,220.6	897.4	2,828.3
EA-SG	779,589.8	201,540.0	15,313.6	897.8	72,332.8

table lists the three parts of the objective separately (cost for reservations not assigned to EVs, energy cost, and cost for missing energy in the batteries at the end of the planning horizon), the number of reservations assigned to EVs and the number of executed generations. The results are averages over the 21 trials. One can see that with an increasing number of reservations per EV, the rate of unassigned reservations increases and thus the cost for unassigned reservations contributes more to the objective value. These cost are computed exactly in the surrogate fitness and thus the surrogate becomes more accurate with an increasing number of reservations per EV. Furthermore, the larger the instance, the lower the number of generations, which can be executed and thus the higher the benefit from using a surrogate. This explains why the standard EA is better on smaller instances while the surrogate-assisted variants are better on the larger instances.

One can see in Table 3 that on the instance with 20 EVs and 80 reservations, more generations are executed with the *EA-SI* variant than with the *EA* variant. This appears counter-intuitive. However, we found that in the *EA-SI* variant the (real) fitness evaluation tends to be faster than in the *EA* variant. This is probably because the *EA-SI* variant tends to evaluate individuals with a higher number of assigned reservations, which makes the charging optimization easier since there are less time steps in which the EVs can charge. At the same time, the computation of the surrogate fitness costs nearly no time on the smaller problem instances. Figure 4 shows for the different EA variants the average and standard deviation of the optimization progress of the 21 trials on the different problem instances. From this one can see again the advantage of the surrogate-

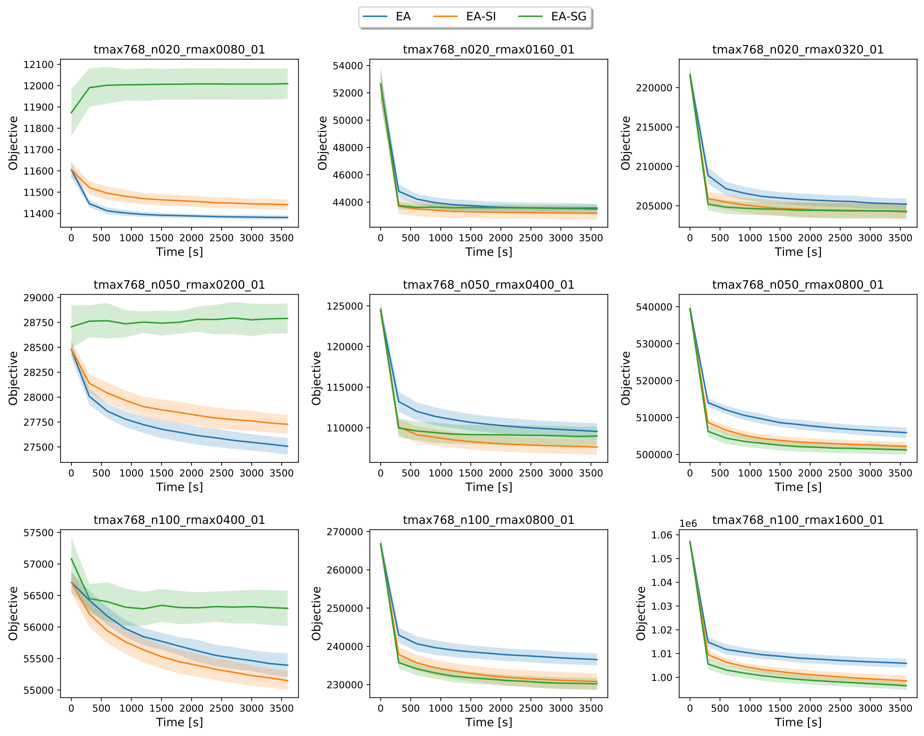


Fig. 4. Average and standard deviation of optimization progress of the EA variants on different problem instances.

assisted variants on the larger instances and the poor performance of the *EA-SG* variant on instances with a small number of reservations per EV. There is also a comparatively high variance in the results of *EA-SG* on such instances.

5 Summary and Conclusion

We proposed a hybrid evolutionary algorithm (EA) for the electric vehicle fleet scheduling problem, which employs an indirect encoding in order to improve the handling of constraints. Furthermore, a problem-specific crossover operator is used. An analysis of the influence of the parameters of the EA revealed that this crossover is clearly beneficial for the optimization. We further propose the use of a problem-specific surrogate fitness. In experiments we evaluated two variants, *EA-SI* and *EA-SG*, of the surrogate-assisted EA and compared it to the standard EA and to two MILP-based approaches. The MILP approaches are clearly outperformed by the EA variants on the considered problem instances. Furthermore, the experimental results show that the use of the surrogate fitness is beneficial. There is no clear winner among the *EA-SI* and *EA-SG* variants, but the *EA-SI* variant appears to be a good compromise between the standard EA without surrogate and the completely surrogate-based *EA-SG* variant – it yields a reasonable performance on the smaller as well as on the larger problem instances.

References

1. Betz, J., Werner, D., Lienkamp, M.: Fleet disposition modeling to maximize utilization of battery electric vehicles in companies with on-site energy generation. Transportation Research Procedia **19**, 241–257 (2016)
2. Bongiovanni, C., Kaspi, M., Geroliminis, N.: The electric autonomous dial-a-ride problem. Transportation Research Part B: Methodological **122**, 436–456 (2019)
3. Díaz-Manríquez, A., Toscano Pulido, G., Barron-Zambrano, J., Tello, E.: A review of surrogate assisted multiobjective evolutionary algorithms. Computational Intelligence and Neuroscience **2016**, 1–14 (2016)
4. Haveman, S., et al.: eMaaS project public summary report. Tech. rep., University of Twente (2020)
5. Jin, Y.: Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation **1**(2), 61–70 (2011)
6. Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial intelligence review: An international survey and tutorial journal **13**(2), 129–170 (1999)
7. Reyes García, J.R., Lenz, G., Haveman, S.P., Bonnema, G.M.: State of the art of mobility as a service (MaaS) ecosystems and architectures — An overview of, and a definition, ecosystem and system architecture for electric mobility as a service (eMaaS). World Electric Vehicle Journal **11**(1) (2020)
8. Sassi, O., Oulamara, A.: Electric vehicle scheduling and optimal charging problem: Complexity, exact and heuristic approaches. International Journal of Production Research **55**(2), 519–535 (2017)
9. Varga, J., Raidl, G.R., Limmer, S.: Computational methods for scheduling the charging and assignment of an on-site shared electric vehicle fleet. IEEE Access **10**, 105786–105806 (2022)