Algorithms for Electric Vehicle Scheduling in Large-Scale Mobility-on-Demand Schemes

Emmanouil S. Rigas[a], Sarvapali D. Ramchurn[b], Nick Bassiliades[a]

[a] *Aristotle University of Thessaloniki, 54124, Thessaloniki, Greece*
{erigas, nbassili}@csd.auth.gr
[b] *Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK,*
sdr1@soton.ac.uk

Abstract

We study a setting where Electric Vehicles (EVs) can be hired to drive from pick-up to drop-off points in a Mobility-on-Demand (MoD) scheme. The goal of the system is, either to maximize the number of customers that are serviced, or the total EV utilization. To do so, we characterise the optimisation problem as a max-flow problem in order to determine the set of feasible trips given the available EVs at each location. We then model and solve the EV-to-trip scheduling problem offline and optimally using Mixed Integer Programming (MIP) techniques and show that the solution scales up to medium sized problems. Given this, we develop two non-optimal algorithms, namely an incremental-MIP algorithm for medium to large problems and a greedy heuristic algorithm for very large problems. Moreover, we develop a tabu search-based local search technique to further improve upon and compare against the solution of the non-optimal algorithms. We study the performance of these algorithms in settings where either battery swap or battery charge at each station is used to cope with the EVs' limited driving range. Moreover, in settings where EVs need to be scheduled *online*, we propose a novel algorithm that accounts for the uncertainty in future trip requests. All algorithms are empirically evaluated using real-world data of locations of shared vehicle pick-up and drop-off stations. In our experiments, we observe that when all EVs carry the same battery which is large enough for the longest trips, the greedy algorithm with battery swap with the max-flow solution as a pre-processing step, provides the optimal solution. At the same time, the greedy algorithm with battery charge is close to the optimal (97% on average) and is further improved when local search is

used. When some EVs do not have a large enough battery to execute some of the longest trips, the incremental-MIP generates solutions slightly better than the greedy, while the optimal algorithm is the best but scales up to medium sized problems only. Moreover, the online algorithm is shown to be on average at least 90% of the optimal. Finally, the greedy algorithm scales to 10-times more tasks than the incremental-MIP and 1000-times more than the static MIP in reasonable time.

Keywords: Mixed Integer Programming, Heuristic search, local search, max-flow, Electric Vehicles, Shared Vehicles, Mobility on Demand

1. Introduction

In a world where over 60% of the total population will be living in, or around, cities the current personal transportation model is not sustainable as it is based almost entirely on privately owned internal combustion engine vehicles. These vehicles cause high pollution (e.g., air and sound), and face low utilization rates[1] [52]. Electric Vehicles (EVs) can be an efficient alternative to those using internal combustion engines when it comes to running costs [17], environmental impact, and quality of driving. However, these advantages come with a trade-off, as EVs have short ranges and long charging times. To address such issues, cities typically resort to building a large number of charging stations with fast chargers, or battery swapping capabilities. Now, such facilities are only worth building if there are enough EVs to use them. However, drivers will not buy EVs if charging stations are not first available, leading to a catch-22 situation.

In order to increase vehicle utilization, Mobility-on-Demand (MoD) schemes have been advocated [30]. MoD involves vehicles that are used by either individuals, or small groups of commuters, thus providing them with an alternative from using their privately owned vehicles. Such systems have the potential to reduce traffic congestion in urban areas, as well as the need for large numbers of parking spots.[2] By doing so, MoD also aims to achieve considerably higher vehicle utilization rates compared to individually owned ones

[1] EVs can be used as energy storage devices when not being driven. In this way (renewable) energy utilization can increase. Thus, in the EVs domain the word *utilization* refers to both the driving and the use of them as energy storage devices.

[2] Demand for travel is not reduced. However, the fact that multiple users end up using the same cars means that there are fewer cars on the road and hence, less congestion (i.e.,

(i.e., few vehicles will cover the transportation needs of many commuters). Moreover, other advantages include the fact that car ownership is reduced, as well as less up-front charges which means low-income people may be better served.

Given the benefits of EVs and MoD schemes, in this paper we explore scenarios within which EVs could be used within MoD schemes, and consider their associated optimisation challenges. By addressing these challenges, the advantages of the two transportation modes would be combined [30, 10]. Moreover, the use of EVs in MoD schemes offer an opportunity to further market EVs to potential car owners as they get to try the technology before buying it. In this way, EV-equipped MoD schemes would help popularise EVs, while at the same time having a positive impact in urban traffic conditions as well as the environment.

To date, a number of MoD schemes, such as ZipCar[3], or CarShare[4] have been proposed, albeit most of them using normal cars. However, EVs present new challenges for MoD schemes. For example, EVs have a limited range that requires them to either charge regularly or have their battery swapped when they stop. Moreover, if such MoD schemes are to become popular, it is important to ensure that charging/swap capacity is managed and scheduled to allow for the maximum number of consumer requests to be serviced across a large geographical area. In addition, in order for MoD schemes to be economically sustainable, and given the higher cost of buying EVs compared to conventional vehicles, it is important to have them working at maximum capacity and servicing the maximum number of customers around the clock.

Against this background, we model the MoD scheme for EVs and develop a number of algorithms to solve the problem of scheduling trips for MoD consumers in order to maximize the number of trip requests serviced while coping with the limited range of EVs. These algorithms attempt to deal with the computational complexity of the scheduling problem in a number of contexts (online v/s offline, with battery swap or battery charge, small-sized or large problems). Thus, we first recast the scheduling problem as a max-flow problem whose solution lets us determine the (upper limit) of trip requests able to be executed given a set of available EVs. Then, we show how

this indirectly can improve congestion as it could reduce the number of cars parked at the sides of the roads. Such parked cars create some congestion).

[3]http://www.zipcar.com/.

[4]http://www.enterprisecarshare.com/.

the scheduling of trips in the MoD scheme is a highly combinatorial problem, for which an optimal offline solution, where all demand is known in advance, scales only up to medium sized problems (tens of EVs and hundreds of trips). Thus, to cope with large problems, we also develop two near-optimal offline solutions, namely an incremental MIP and a greedy heuristic, as well as a tabu search-based local search technique to further improve the solution quality of the non-optimal algorithms. Moreover, to tackle the online version of the problem, where demand is not known in advance, we develop an online scheduling algorithm. In all cases, and given the limited range of EVs, we consider situations where they can either have their battery swapped (with a fully charged one), or charged at the stations. The work presented here has been initiated at [38] with basic versions of the MIP and the greedy algorithms for battery swapping. Specifically, this paper advances the state of the art as follows:

1. We provide a characterisation of the MoD scheme as a max-flow problem. By solving this problem, we are able to determine the set of all feasible trips given a set of available EVs.
2. We propose an optimal Mixed Integer Programming (MIP) formulation of the problem of scheduling EVs in a MoD scheme that maximizes the number of completed tasks (i.e., trip requests from consumers) or the EV utilization (i.e., number of time points each EV is travelling), either using battery swap or battery charging at each station.
3. Given the average scalability of the optimal solution, we develop an incremental-MIP and a greedy heuristic algorithm which are shown to generate near-optimal solutions with considerably lower execution times.
4. We propose a tabu search-based local search technique in order to further improve the solution quality of the non-optimal algorithms.
5. We propose a battery swap optimization algorithm which minimizes the number of necessary battery swaps in order to reduce the need for spare batteries and thus, cost.
6. We propose an online algorithm for scheduling EV trips across the MoD that can cope with uncertainty in the number of future trip requests.
7. Finally, using real-world data of shared vehicle stations in Washington, DC we observe that when all EVs carry the same battery which is large enough for the longest trips, the greedy algorithm with battery swap in combination with the max flow provides the optimal solution.

At the same time, the variation with battery charge is at 97% of the optimal without the local search, and at 98.5% when local search is used. Moreover, in case where some EVs do not have a large enough battery to execute some of the longest trips, we observe that the greedy algorithm does not provide the optimal solution anymore and that the incremental MIP is the correct choice as it generates solution 0.5% better than the greedy and with lower execution time, while the optimal algorithm is the best but scales up to medium sized problems only. Moreover, we show that when the objective is the maximization of EV utilization, the utilization increases by 3.6% and the number of completed tasks reduces by 1.5% on average. In addition, the online algorithm is at least 90% of the optimal. In terms of scalability, the greedy algorithm (without local search) scales to 10-times more tasks than the incremental MIP and 1000-times more than the static MIP in reasonable time.

When taken together, our algorithms and results establish the first benchmarks for the study of EV scheduling algorithms in MoD schemes.

The rest of the paper is structured as follows: Section 2 discusses the related work, Section 3 presents the model of a typical MoD scheme and Section 4 presents the formulation of the problem as a max flow one. Section 5 presents the MIP formulation of the problem with battery swap (Section 5.1) and with battery charge (Section 5.2), the incremental MIP scheduling algorithm (Section 5.3), the greedy algorithm with battery swap (Section 5.4) and battery charge (Section 5.5), the local search algorithm (Section 5.6) and the battery swap optimization algorithm (Section 5.7). Moreover, Section 6 presents the online scheduling algorithm and Section 7 describes our empirical evaluation. Finally, Section 8 concludes and presents future work.

2. Related Work

In the problem we study in this paper, we have a set of locations which act as pick-up and drop-off stations and a number of EVs available at each station. Tasks (i.e., trip requests for a specific point in time) are collected by the MoD company and an assignment of EVs to tasks is calculated. EV relocation is not supported, thus the end location of one executed task is always the start location of another. After the execution of each task, the EVs' battery is either replaced with a fully charged one, or charged. In this section, related

work both in terms of the problem structure, as well as problem domain and application is presented:

2.1. Vehicle routing and scheduling problems

Our work shares similarities with other vehicle routing and scheduling problems such as bike sharing, rolling stock, aircraft scheduling, rental car scheduling and ride sharing. For example, Raviv et al. [36], present algorithms for the efficient repositioning of bikes in order to maximize customer satisfaction in a bike sharing system. Agatz et al. [1] summarize a number of optimization techniques for the problem of ride sharing, while Bistaffa et al. [7] apply coalition formation techniques in order to calculate stable (from a game-theoretic point of view) payments for a ride sharing setting. Moreover, Budai et al. [9] present an optimal algorithm for the problem of rolling stock balancing, while Bayen et al. [5] use MIP in order to optimally solve an aircraft scheduling problem. Between these works and ours there are some general similarities, such as the fact that optimization techniques as well as greedy approaches are used, but the nature of our problem, and in particular the need for charging or battery swapping between task execution, makes it quite different from them.

In addition, similarities can also be found with problems such as the capacitated vehicle routing problem [14] (i.e., special case of the Vehicle Routing Problem [15], where each vehicle has a limited carrying capacity), the project scheduling problem [51], and the machine scheduling problem [28]. Regarding the capacitated vehicle routing problem, our case is similar to the extent that each EV has a limited capacity, but here, the EVs must not necessarily serve all the customers, each task must be executed at a specific point in time and the EVs have a limited range leading to charging or battery swapping being necessary between tasks. As far as the project scheduling problem is concerned, for an EV to execute a task, it must be in the right location, the right time and it must have enough range in the battery. These could be considered the skills of the EV (i.e., in the project scheduling problem there are skills). However, in contrast to the project scheduling problem, here exactly one EV must execute one task and the starting time as well as the duration of the task are fixed and they do not depend on the EV that will execute it. Finally, regarding the machine scheduling problem, our problem has a similar structure as, for example, all EVs are assumed to be identical, each of them executing one task at a time and each task is executed by one EV. The main difference, is that the execution of each task

must start at a specific time point, and that after the execution of a task, an EV may not be available to execute another task before it charges or swaps the battery. Overall, the need for battery charging or swapping as well as the strict order of task execution differentiate our problem compared to the three aforementioned ones, and make it harder to find the optimal solution.

2.2. MoD with conventional vehicles

To date, a number of works have dealt with problems related to the management of vehicles in MoD schemes. For example, Pavone et al. [33] study a setting similar to ours where a finite number of customers, vehicles as well as pick-up and drop-off stations exist. Due to the fact that the system can, in some cases, become unbalanced (i.e., high number of vehicles at some stations and low at others), the authors apply robotic rebalancing techniques. In more detail, they assume that empty robotic vehicles can be autonomously driven across stations in order to match the high demand. The authors apply mathematical programming-based techniques to minimize the number of necessary rebalancing trips. In addition, Smith et al. [45] study the same problem but, instead of robotic vehicles, the authors assume that rebalancing drivers exist. In doing so, they found, through empirical studies, that on some occasions these rebalancing drivers themselves become unbalanced. In order to solve this problem, they assume that customers can transport rebalancing drivers across locations and they use mathematical programming to optimally route them across the stations. In a slightly different vein, where robotic autonomous driving vehicles exist, Zhang et al. [46] model a MoD system as a closed Jackson network [44] with passenger loss. They show that an optimal algorithm which minimizes the number of vehicles needing rebalancing while achieving good vehicle availability throughout the network can be found by solving a linear program. The authors evaluate and verify the effectiveness of their approach in a realistic setting and they also discuss the environmental benefits of such approaches, as they lead to a reduced need for vehicles. In contrast to all three papers, here we do not use a-posteriori rebalancing techniques. Instead, the decision making procedure for the selection of trips to be executed also takes into account the proper distribution of the EVs across the locations. In doing so, we have taken into consideration insights from [6], where the authors show that leaving tasks unexecuted at a certain time point, can lead to higher task execution in the future.

Looking at the vehicle sharing problem from a different perspective, Carpenter et al. [12] focus on the problem of sizing vehicle pools (i.e., a number

of vehicles at a single location) for a finite set of customers. The authors aim to minimize the size of these pools, while still achieving high customer satisfaction. They propose three analytical techniques to size a vehicle pool for a finite population of customers, according to the pools' busy period demand. Moreover, they propose an additional heuristic sizing method, which requires no prior data about pool demand. In the same vein, George et al. [22] address the problem of determining the optimal fleet size for a vehicle sharing company and derive analytical results for its relationship to vehicle availability at each station in the company's network of locations. Initially, they formulate a closed queuing network model of the system and later they develop a profit-maximizing optimization problem for determining the optimal fleet size. Similarly, Waserhole et al. [55] formulate the setting as a closed queuing network model of the system with infinite buffer capacity and Markovian demands. However, in this case the authors use pricing techniques to incentivise customers not to choose trips that would unbalance the system. In our case, we do not study the optimal number of vehicles at the stations, but we have considered the case where the initial location of a fixed number of vehicles is optimized in order to maximize customer satisfaction (see Section 7.2).

In all works presented so far, internal combustion engine-based vehicles are assumed, and hence they do not account for the limited range of EVs. Thus, while balancing the load (i.e., number of pending requests across the network) across the network (i.e., by choosing which trips to execute) serving at the same time the maximum number of users, the amount of time spent charging the vehicles is not taken into account.

2.3. Techniques for the management of EVs in MoD schemes

In recent years, there has been a significant interest within the research community in addressing the challenges involved in deploying EVs. In this vein, a significant number of AI-based approaches to solve EV-related problems have been proposed. According to [37], the majority of the existing works consider three main problem categories: 1) *Energy efficient EV routing and range maximization*, where algorithms and mechanisms have been developed to route EVs in order to minimize energy loss and maximize energy harvested[5] during a trip (e.g., [41], [47] and [48]). 2) *Congestion management*,

[5]EVs have the ability to convert heat produced under braking to electricity and recharge their batteries.

where algorithms have been designed to manage and control the charging of EVs, so as to minimize queues at charging points and the discomfort to the drivers (e.g., [34], [16] and [49]). 3) *Integration of EVs into the Smart Grid*, where a number of mechanisms have been developed to schedule and control the charging of EVs (Grid to Vehicle - G2V) so that peaks and possible overloads of the electricity network can be avoided, while minimizing charging cost. Other mechanisms have also been developed to utilize the storage capacity of the EVs (Vehicle to Grid - V2G) in order to balance the electricity demand, or to ease the integration of intermittent renewable energy sources to the grid (e.g., [54], [53], [50], [29], [39] and [40]).

Although AI techniques have been applied to EV-related problems for privately owned vehicles, limited work have considered the management of EV activities in an MoD scheme. For example, Cepolina and Farina [13] study the use of single-sitter compact-sized EVs in a MoD scheme operating in a pedestrian zone. The vehicles are shared throughout the day by different users and similarly to the majority of the works presented so far, one way trips are assumed. However, here the authors also assume open ended reservation to exist (i.e., the drop-off time is not fixed), thus adding one more dimension to the problem. Given this, they propose a methodology to optimize the fleet size and its distribution among the stations so as to maximize the number of serviced customers while minimizing cost using a random search algorithm. Finally, Doppers and Iwanowski [18] aim to map EVs to customers based on a set of criteria such as the priority of the mobility needs, the state of battery of each EV and the charging time. In so doing, they model the problem of assigning EVs to tasks in a MoD scheme with the well known quadratic assignment problem [27] and they solve the problem using an ant-colony optimization algorithm. In our case, the only criterion of selecting which trips to execute is either the maximization of the total number of serviced customers, or the maximization of EV utilization. However, in contrast to these works, we study settings where different EV charging approaches are used considering both traditional charging and battery swapping. Next, we formally define the problem.

3. Problem Definition

We study a MoD setting where customers announce their intentions to drive between pairs of locations at a particular time, a day ahead. After all in-

tentions have been collected by the MoD company,[6] it applies a scheduling algorithm to assign EVs to tasks (i.e., trips across a set of locations). In choosing the tasks it will execute, the MoD company aims to maximize either the number of customers that will be serviced, or the utilization of the EVs. We assume that EVs have a limited driving range which requires them to have their battery either swapped [48], or charged at the stations that are part of the MoD scheme. In a battery swap station the battery is not recharged but instead it is replaced by a fully charged one. Battery swap is an efficient alternative to battery recharging as it significantly reduces the idle time of EVs. Battery swapping with privately owned vehicles raises the problem of battery ownership, as once an EV unloads its battery it may never get the same battery back again. However, in cases of shared EVs, such as the one we study here this is not an issue as a single MoD company exists and all batteries belong to it. Moreover, MoD schemes overcome the high cost and the implementation complexity of battery swapping as they take advantage of economies of scale by using a specific battery type for all EVs (we assume all EVs are of the same model). Thus, the process of battery swapping becomes easier and less expensive.

We denote by $j \in A$ a set of EVs and by $k \in L$ a set of locations which are pick-up and drop-off stations, where each $k \in L$ has a maximum capacity $c_k \in N$. We consider a set of discrete time points $T \subset N$, $t \in T$, where time is global for the system and the same for all EVs. Moreover, we have a set of tasks $i \in \Delta$ where each task is denoted by a tuple $\langle k_i^{start}, k_i^{end}, t_i^{start}, \tau_i, b_i \rangle$ where k_i^{start} and k_i^{end} are the start and end locations of the task, t_i^{start} is the starting time of the task, τ_i is its travel time (each task has also an end time $t_i^{end} = t_i^{start} + \tau_i$), and b_i is the energy cost of the task. We also denote all tasks starting from location k at time point t as $\Delta^{st}(t,k) = \{i \in \Delta\colon t_i^{start} = t, k_i^{start} = k\}$ and all tasks ending at location k and time point t as $\Delta^{end}(t,k) = \{i \in \Delta\colon t_i^{end} = t, k_i^{end} = k\}$. Note here, that one-way rental is assumed, and therefore, start and end locations of a task are different.[7]

[6]We assume a single MoD company to operate all stations of the setting. Note, that having multiple companies may induce some interesting strategic decision making challenges which we aim to study in future.

[7]We assume that customers drive the cars between start and end locations without stopping or parking them during the trip. In case a user would like to set the start and end location of a task to be the same, this would be equivalent to adding specific travel times to each trip as opposed to computing the travel time from pairs of charging locations.

One-way rental introduces significant flexibility for users, but management complexities (e.g., complex decision making in choosing which customers to service, and high importance of the initial location of EVs) [4].

Each EV j has a current location at time point t, denoted as $k_{j,t}$, and this location changes only when the EV executes a task (i.e., an EV cannot change its location without executing a task). Here, we assume that at time point $t = 0$ all EVs are at their initial locations $k_{j,t=0}^{initial} \in L$, (the initial location can either be fixed, or optimized at run time) and that their operation starts at time point $t \geq 1$. Moreover, each j has a current battery level $b_{t,j} \in N$, a maximum range τ_j^{max}, a fixed consumption rate con_j (unit of energy/ time point) and therefore, a remaining driving range in terms of time $\tau_t(j) = [b_{t,j}/con_j] \in N$, as well as a fixed charging rate ch_j. For a task i to be accomplished, at least one EV j must be at location k_i^{start} at time point t_i^{start}. At any t, each EV should either be parked at exactly one location, or traveling between exactly one pair of locations. Henceforth, index j stands for EVs, k for locations, t for time points and i for tasks (see also Table 1). Where required to simplify the explanation, we slightly abuse notation to denote that some quantities are dependent on others (they are functions - e.g., $\tau_t(j)$), whereas others are independent (these can be identified solely by indices - e.g., j).

In this paper we mainly study this offline setting and then we propose a preliminary solution to an online setting based on our insights from the offline one. In the following section we present a formulation of the problem as a max-flow one.

4. Formulation as a Maximum Flow Problem

In this section, we formulate the EV to task allocation problem as a max-flow problem [24] and we solve it using MIP. By using this approach we aim to remove the tasks that are impossible to be executed, thus reducing the execution time and increasing the solution quality of the scheduling algorithms presented in the following section. All pick up and drop off locations are considered to be the nodes of the network and the execution of tasks is the flow in and out of each node. The initial location of each EV is a source node and the end location of each EV is a sink node. In our case we have many sources and many sinks. Thus, it is a multi-source multi-sink maximum flow problem and can be transformed into a maximum flow problem as follows: Given a network N = (V,E) with a set of sources S = $s_1, ..., s_n$ and a set of

Notation	Explanation
j	EV
k	Location
c_k	Capacity of k
t	Time point
i	Task
k_i^{start}	Start location of task i
k_i^{end}	End location of tasks i
t_i^{start}	Starting time point of task i
t_i^{end}	Ending time point of task i
τ_i	Travel time of task i
b_i	Battery cost of task i
τ_j^{max}	Maximum range of EV j
$b_{t,j}$	Battery level of EV j at time point t
con_j	Consumption rate of EV j
ch_j	Maximum charging rate of EV j
$\tau_t(j)$	Driving range in terms of time
$prk_{j,t,k}$	True if EV j is parked at location k at time t (boolean)
$\epsilon_{j,i,t}$	True if EV j is working on task i at time t (boolean)
λ_i	Task i accomplished (boolean)
$bch_{j,t}$	Charging rate of EV j at time point t

Table 1: Notations used in problem definition and algorithms

sinks T $= t_1, ..., t_m$ instead of a single source and sink node, we are to find the maximum flow across N. We transform the multi-source multi-sink problem into a maximum flow problem by adding a consolidated source connecting to each vertex in S and a consolidated sink connected by each vertex in T (also known as supersource and supersink) with infinite capacity on each edge.

In this formulation, we have one decision variable, namely $\lambda_i \in \{0, 1\}$ denoting whether a task i is accomplished or not and the objective is the maximization of executed tasks (Equation 1). Alternatively, the utilization of the EVs (i.e., total traveling time) can also be maximized (Equation 2). This is achieved under the constraint that for each location, the inflow is always greater or equal to the outflow (Equation 3). Given that an EV changes location only after executing a task, for each location k and time point t, the number of tasks that started from k any time point prior to t,

must be less or equal to the tasks that ended at k any time point prior to t, plus the EVs whose initial location was equal to k (i.e., no task can be executed without the existence of at least one EV).

$$max \sum_{i \in \Delta} (\lambda_i) \tag{1}$$

$$max \sum_{i \in \Delta} (\lambda_i \times \tau_i) \tag{2}$$

$$\sum_{i \in \Delta : t_i^{start} < t, k_i^{start} = k} (\lambda_i) \leq \sum_{i \in \Delta : t_i^{end} < t, k_i^{end} = k} (\lambda_i) + \sum_{j \in A : k_j^{init} = k} 1, \forall t, \forall k \tag{3}$$

The outcome of this procedure is the set of tasks that can be executed assuming that battery swap is being used and that (a) all EVs start with the same battery level and (b) a fully charged battery is capable of executing the longest trip. In fact, if (a) and (b) hold, the max flow algorithm returns the optimal solution in terms of the number of completed tasks. This can easily be verified given the following reasons: every task i has a source location k_i^{start} and a destination k_i^{end}. Moreover, it has a fixed time of departure t_i^{dep} and a fixed duration τ_i. In the max flow algorithm, the duration of a task also contains the time for the battery swap. Given that all EVs carry the same battery which is fully charged at the beginning of each task, any EV can execute any task. Thus, a task can be executed if and only if at least one EV exists at k_i^{start} the time point t_i^{dep}. Equation 3 assures that no task will be considered executed if not enough EVs exist at the source location of the task the time of departure. Thus, given the objective function (Equation 1 or Equation 2), the solver will select to execute the tasks that lead to the maximization of the total number of completed tasks or the EV utilization respectively. In case where either (a), or (b), or both do not hold, then the max flow algorithm will provide the theoretical upper limit in terms of completed tasks. This is true due to the fact that this algorithm assumes all EVs to have the same capabilities. Thus, if an EV is assigned to a task that it cannot actually execute, this task and possibly some future ones will not be executed. The same holds for the case of battery charging, as the charging takes longer time compared to battery swapping. We will refer to this algorithm as *MaxFlow*.

In the case where the initial location of the EVs is not given as input, but it should be optimized, the above formulation is updated as follows: One more

decision variable, namely $initLoc_{j,k} \in \{0, 1\}$, denoting whether location k is the initial location for EV j is added. Moreover, Equation 4 which constrains each EV to have exactly one initial location is added to the formulation, and Equation 3 is updated by using the pre-defined decision variable and becomes Equation 5. We will refer to this algorithm as *MaxFlowInit*

$$\sum_{k \in L} initLoc_{j,k} = 1, \forall j \tag{4}$$

$$\sum_{i \in \Delta : t_i^{start} < t, k_i^{start} = k} (\lambda_i) \leq \sum_{i \in \Delta : t_i^{end} < t, k_i^{end} = k} (\lambda_i) + \sum_{j \in A} \sum_{k \in L} (initLoc_{j,k}), \forall t, \forall k \tag{5}$$

The max flow algorithm calculates the optimal set of tasks to be executed. However, the task execution schedule for each EV is not calculated. Thus, this algorithm is used as a pre-processing step in order to determine the tasks to be executed. Then, any of the scheduling algorithms described in the following section has to be used in order to calculate the task execution schedule for each EV. As can be seen in Sections 7.1 and 7.2 it improves the scalability of the optimal scheduling algorithm and the solution quality of the greedy one.

5. Offline Scheduling Algorithms for MoD Schemes

In this section, we tackle the problem defined in Sections 3 and 4 in the particular context where the demand for trips at each station is known in advance. We term this problem the **offline** problem, as the algorithm is run to pre-determine all trips for the day ahead. In Section 5.1 we present the MIP formulation of the problem with battery swap and in Section 5.2 the one with battery charge. Then, in Section 5.3 the incremental MIP scheduling algorithm and in Section 5.4 the greedy algorithm with battery swap and in Section 5.5 the one with battery charge. Finally, in Section 5.7 we present the battery swap optimization algorithm.

5.1. Optimal Scheduling with Battery Swap

The aim of the MoD scheme is to maximize either the number of tasks that are completed (a.k.a. customer satisfaction) (Equation 6) or the total number of time points that EVs are traveling (a.k.a. EV utilization) (Equation 7) (i.e., the objective functions can be used alternately). To achieve this, we

present an optimal solution based on Mixed Integer Programming (MIP)[8] (solved using IBM ILOG CPLEX 12.6.2), where we use battery swapping to cope with the EVs' limited range. MIP techniques have been particularly useful to solve such large combinatorial problems (e.g., combinatorial auctions [42], [3], travelling salesman problem [19]). Also, such a solution can be used as a benchmark for more customized algorithms (as we do in this paper in Section 5.6 and Section 6). We will refer to this algorithm as *Off-Opt-Swap*. Note that in case the max flow algorithm returns the optimal set of tasks to be executed, this algorithm simply calculates the EV to task execution schedule. However, if the max flow returns the upper limit in terms of executed tasks, this algorithm also decides which of these tasks can actually be executed. In more detail, we define three binary decision variables: 1) $\lambda_i \in \{0,1\}$ denoting whether a task i is accomplished or not, 2) $\epsilon_{j,i,t} \in \{0,1\}$ denoting whether EV j is executing task i at time t or not, and 3) $prk_{j,t,k} \in \{0,1\}$ denoting whether j is parked at time point t at location k or not. Moreover, a set of constraints is used:

Objective functions:

$$max \sum_{i \in \Delta} (\lambda_i) \tag{6}$$

$$max \sum_{j \in A} \sum_{i \in \Delta} \sum_{t \in T} (\epsilon_{j,i,t}) \tag{7}$$

Subject to:

- *Completion constraints:*

$$\sum_{j \in A} \sum_{t_i^{start} \leq t < t_i^{end}} \epsilon_{j,i,t} = \tau_i \times \lambda_i, \forall i \tag{8}$$

$$\sum_{j \in A} \sum_{t_i^{start} > t, t \geq t_i^{end}} \epsilon_{j,i,t} = 0, \forall i \tag{9}$$

$$\epsilon_{j,i,t+1} = \epsilon_{j,i,t} \forall j, \forall i, \forall t : t_i^{start} \leq t < t_i^{end} - 1 \tag{10}$$

[8]Other optimization techniques could also be applied here. However, the aim of this paper is not to study different optimization techniques, but to establish the first benchmarks to a problem that has not been previously solved.

$$\sum_{t_i^{start} \leq t < t_i^{end}} \epsilon_{j,i,t} \leq \tau_{t_i^{start}}(j), \forall i, \forall j \tag{11}$$

- *Temporal, spatial, and routing constraints:*

$$\sum_{k \in L} prk_{j,t,k} = 1 - \sum_{i \in \Delta} \epsilon_{j,i,t}, \forall j, \forall t \tag{12}$$

$$2 \times \sum_{i \in \Delta} \epsilon_{j,i,t_i^{start}} = \sum_{k \in L} \sum_{t \in T} \left| prk_{j,t+1,k} - prk_{j,t,k} \right|, \forall j \tag{13}$$

$$prk_{j,t_i^{start}-1,k_i^{start}} \geq \epsilon_{j,i,t_i^{start}}, \forall i, \forall j \tag{14}$$

$$prk_{j,t_i^{end},k_i^{end}} \geq \epsilon_{j,i,t_i^{end}}, \forall i, \forall j \tag{15}$$

$$\sum_{j \in A} (prk_{j,t,k}) \leq c(k), \forall k, \forall t \tag{16}$$

$$prk_{j,t=0,k} = k_a^{init}, \forall j, \forall k \tag{17}$$

$$\epsilon_{j,i,t=0} = 0, \forall j, \forall i \tag{18}$$

- *Cut constraints:*

$$\sum_{j \in A} prk_{j,t,k} = \sum_{j \in A} prk_{j,t-1,k} + \sum_{\Delta^{st}(t,k)} \lambda_i - \sum_{\Delta^{end}(t,k)} \lambda_i, \forall t, \forall k \tag{19}$$

The *completion* constraints ensure the proper execution of tasks. Thus, for each executed task, the time travelled must be equal to the duration of the trip concerned[9] (Equation 8), while, at the same time no traveling must take place when a task is not executed (Equation 9). Moreover, each task is executed by only one EV at a time (Equation 10 together with Equation 13). Now, for an EV to execute a task, its full range, calculated based

[9]In case a user would like to set the start and end location of a task to be the same, this would be equivalent to adding specific travel times to each trip as opposed to computing the travel time from pairs of charging locations. Otherwise Equation 3 of the MIP formulation would not work properly.

on the battery level at the starting time of the task, must not be violated (Equation 11) (i.e., in case trip requests are not within the range of a single battery load, the solver will not schedule them). We assume all EVs to have a fixed average consumption, and that each time an EV reaches a parking station a fully charged battery is swapped into it, while the number of swaps is minimized a posteriori (Section 5.7 presents a battery swap minimization algorithm).

The *temporal, spatial and routing* constraints ensure the proper placement of the EVs over time. Equation 12 requires that for each time point at which an EV is executing a task, this EV cannot be parked at any location and also assures (together with Equation 10) that at each time point, each EV executes at most one task. Moreover, Equation 13 ensures that no EV changes location without executing a task (the sum of all changes of EVs' locations as denoted in prk decision variable, must be double the total number of tasks that are executed). Note that, this constraint is linearized at run time by CPLEX[10].

Now, whenever a task is to be executed, the EV that will execute this task must be at the task's starting location one time point before the task begins (Equation 14), and similarly, whenever a task has been executed, the EV that has executed this task must be at the task's end location the time point the task ends (Equation 15). Moreover, at every time point, the maximum capacity of each location must not be violated (Equation 16). Finally, at time point $t = 0$, all EVs must be at their initial locations (Equation 17), which also means that no tasks are executed at $t = 0$. Note that, in case Equation 17 is removed, the solver will decide on the optimal initial location of each EV given a specific set of tasks waiting to be executed.

Equation 19 ensures that for every location, the total number of EVs at charging stations changes only when EVs depart or arrive to execute a task, or after executing tasks. Despite the fact that this constraint is covered by Equation 13, when added to the formulation, it significantly speeds up the execution time. For example, for a setting with 8 locations, 15 EVs, 60 time points and 70 tasks, constraint 13 reduces the average execution time from 450 seconds to less than 200 seconds. In fact, it is known that the introduction of additional *cut constraints* into a MIP problem may cut off infeasible solutions at an early stage of the branch and bound searching

[10]This is usually done by adding two extra decision variables and two extra constraints.

process and thus reduce the time to solve the problem [20].

By using battery swapping, an EV can have full range in just a few minutes. However, such battery swap schemes impose high costs and implementation complexity. Thus, in the next section, a variation of the optimal solution where, instead of battery swap, a more traditional battery charge scheme is used is presented.

5.2. Optimal Scheduling with Battery Charging

Here, the problem of scheduling EVs in a MoD scheme is formulated using fast battery charging (we will refer to this algorithm as *Off-Opt-Charge*). In addition to the decision variables and the constraints presented in the previous section, we define one continuous variable namely $bch_{j,t} \in [0, ch(j)]$ which denotes whether an EV j is charging at time point t and at which charging rate (i.e., the charging rate can be any between 0 and the maximum charging rate - $ch(j)$[11], as well as 2 *completion* constraints.

$$bch_{j,t} \leq \sum_{k \in L} prk_{j,k,t} \times ch(j), \forall j, \forall t \tag{20}$$

$$0 \leq b_{j,t=0} + \sum_{t'=0}^{t} bch_{j,t'} - \sum_{i \in \Delta} \sum_{t''=t_i^{start}}^{t} \epsilon_{i,j,t''} \times con(j) \leq 100, \forall j, \forall t \tag{21}$$

Equation 20 ensures that each EV j can charge only while being parked. When an EV is parked, it can charge with a charging rate up to its maximum one. However, when it is driving and $prk_{j,k,t} = 0$ it cannot charge. The charging takes place at any time point chosen by the solver as long as the available range will not compromise the task execution ability. At the same time, Equation 21 ensures that the battery level of an EV j never exceeds 100% and never goes below 0% (i.e., $b_{j,t=0}$ is the initial battery level, $\sum_{t'=0}^{t} bch_{j,t'}$ is the amount of battery charged and $\sum_{i \in \Delta} \sum_{t''=t_i^{start}}^{t} \epsilon_{i,j,t''} \times con(j)$ is amount of energy consumed). Thus, no EV j will execute a task i for which it does not have enough range, nor will it charge more than its battery capacity. Note that Equation 21 replaces Equation 11.

[11] Not using the maximum charging rate can prolong the life time of the battery. Moreover, the available energy could be limited and the highest charging rate could not be available, as for example limited energy from renewable sources in used (however we do not study this case in this paper)).

The solutions presented so far calculate the optimal schedule for the EVs. However, as we show later in Sections 7.1 and 7.7, they have quadratic time complexity and are mainly usable for small and medium sized problems (less than 300 tasks and 20 EVs). For this reason, algorithms that can calculate solutions close to the optimal, but with a low computational complexity, are essential. In the next section, an algorithm which incrementally calls the MIP for each EV is presented.

5.3. Incremental MIP Scheduling Algorithm

MIP problems are known to be NP-Hard in the worst case [25]. A widely employed strategy to overcome the computational difficulty for the solution of large MIP models is based on the idea of *decomposition* [20]. The decomposition approach divides a large and complex problem, which may be computationally expensive or even intractable when formulated and solved directly as a single MIP model, to smaller sub-problems, which can be solved much more efficiently. Usually, decomposition approaches only lead to suboptimal solutions. However, they substantially reduce the problem complexity and the solution time, which makes it possible to apply MIP based techniques to large real-world problems.

Algorithm 1 Incremental MIP Scheduling Algorithm.

Require: Δ and A and L and T and $\forall i \in \Delta$, τ_i, and $\forall j \in A$, $k_j^{initial}, \tau_j^{max}$, and $\forall k \in L$, c_k.

1: $completedTasks \leftarrow \emptyset$
2: **for** $\forall j \in A$ **do**
3: Call $Optimal(\Delta, j)$
4: {Optimal: Off-Opt-Swap or Off-Opt-Charge}
5: **Return** Schedule for j and set of latest completed tasks
6: {Once the MIP has been solved, update the set of completed tasks with the new completed tasks }
7: $completedTasks \leftarrow completedTasks + newTasks$
8: $\Delta \leftarrow \Delta - newTasks$
9: **end for**
10: **Return** $completedTasks$

Based on the idea of decomposition, we present an offline scheduling algorithm, which incrementally calls and solves the MIP formulation of the problem. Note that some insights were also taken from [8] and in particular

from the discussion on the planning of completely independent agents. The intuition behind this algorithm is the following: Given that the dimension of the problem that affects the execution time the most is the number of EVs (see Section 7.1), we solve the MIP problem for each EV separately. Thus, the MIP formulation is solved sequentially (see Algorithm 1), for each EV within the list of available EVs. In more detail, after the set containing all completed tasks is initialized to the empty one (line 1), the optimal algorithm is called for *each EV*[12] (lines 2-9): Every time the schedule for one EV is calculated (line 6), the sets for the completed tasks and the remaining tasks are updated accordingly (lines 7, 8). At the end of this procedure, the set that contains the completed tasks is returned (line 10). Note that this algorithm works both with battery swap and battery charge. We will refer to the one with battery swap as *Off-Incr-Swap* and to the one with battery charge as *Off-Incr-Charge*.

The incremental MIP algorithm achieves near optimal solutions with relatively small execution time (see Section 7.1). However, for problems with more than a few thousands EVs and tasks, the execution time increases.[13] Given that greedy algorithms based on heuristic search have proven to be effective in similar highly combinatorial problems [35], in the next section, such greedy algorithms are presented.

5.4. Greedy Scheduling with Battery Swap

Given that EVs change locations only when being driven by customers, the tasks that an EV will be able to execute in the future are directly related to the ones it has already executed in the past (i.e., the end location of one task will be the start location for the next one). In large settings, normally not all tasks can be executed. Thus the selection of the ones to execute is of great importance, since each decision can affect future task execution. In the case of our MIP formulation, the solver finds the optimal schedule for EVs that maximises the number of tasks executed, or the EV utilization. However, the scalability of that algorithm is average. For this reason, here we present a greedy algorithm which applies a one-step look ahead heuristic search mechanism. This algorithm achieves, as we show later, near optimal

[12] We have experimented with various orderings of the EVs, but without significant effect on the execution time or the solution quality.

[13] This algorithm can consume a lot of memory. Thus, efficient memory management in the implementation of the algorithm is crucial. We used methods provided by CPLEX.

performance and scales to thousands of EVs and tasks. We will refer to this algorithm as *Off-Greedy-Swap*.

For each time point t and for each station k, we consider the number of the available EVs, as well as the number of the remaining tasks to be executed. Given that all tasks to be executed are known in advance, if the number of available EVs is greater than, or equal to the number of tasks remaining to be executed, we can safely assume that enough EVs are available to execute all tasks (note that more EVs may arrive later after executing tasks). In this case, each task is executed according to its starting time.

In the case where the number of EVs is lower than the number of the remaining tasks, we need to carefully choose which of the tasks need to be executed in the current time point, so that more tasks can be executed in the future. To this end, we employ a one-step look ahead heuristic criterion. The intuition behind this heuristic is to give priority to the tasks that lead to destinations where either 1) high number of tasks remain to be executed, or 2) many tasks are about to be executed soon, or 3) both. In this way, the EVs do not remain idle for long time (i.e., it combines customer satisfaction and EV utilization).

At each time point, we select all tasks i that should start being executed at this time point. Based on the initial location k of a task i, we calculate the score in Equation 22. In calculating the sum, we take into consideration the number of EVs that already exist at the destination. However, we remove the ones that will depart before the arrival of the EV at k. Thus, we come up with a score for each location (i.e., $sc_k = \{k, score_k\} \in \mathbb{R}$) and we select to execute the tasks that lead to locations with the highest score. The sum guarantees that both locations with tasks which are about to be executed soon and locations with high numbers of remaining tasks will get a high score. Example (see Figure 1): In the destination of a given task, there are five tasks remaining to be executed with departure times: 5, 7, 8, 9, 12. If the task is executed, the EV will arrive at the destination at time point 7. Thus, it may also be able to execute one of tasks 8, 9, 12. Moreover, 3 EVs already exist at the destination. If these EVs take care of 5,7, 8, then the new EV could be able to execute tasks 9 and 12. For this reason the score is calculated only using 9 and 12.

$$score_k = \sum(1/\Delta t), \Delta t = t_i^{dep} - t_j^{arr}, (\forall j, \forall i \text{ at the destination after } t_j^{arr}) \tag{22}$$

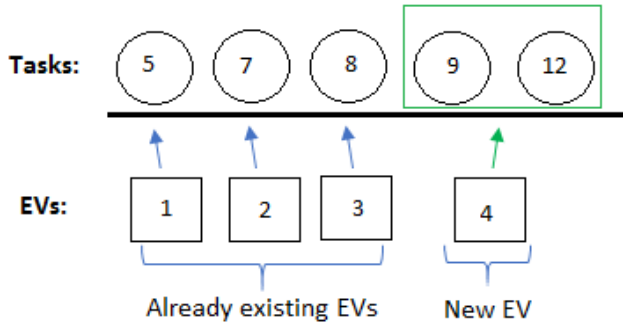


Figure 1: Greedy heuristic- $score_k$ calculated for remaining tasks only.

The greedy scheduling formulation that is presented here, consists 1) of a pre-processing phase and 2) of the main scheduling algorithm (Algorithm 4) which applies a task execution algorithm.

Algorithm 2 Initialization of sets and variables.

Require: Δ and A and L and T and $\forall j \in A$, l_{init}^{j}.

for all ($k \in L$) **do** {Initialize to the empty sets. }
    $\Delta_k \leftarrow \emptyset$, $CT_{k,t} \leftarrow \emptyset$
end for
for all ($i \in \Delta$) **do** {Populate Δ_k. }
    $\Delta_{k_i^{start}} \leftarrow \Delta_{k_i^{start}} + i$
end for
for all ($k \in L$ and $t \in T$ and $i \in \Delta$) **do** {Populate $CT_{k,t}$. }
    if ($k_i^{start} == k$ AND $t_i^{start} == t$) **then**
        $CT_{k,t} \leftarrow CT_{k,t} + i$
    end if
end for
for all ($k \in L$ and $t \in T$) **do** {Populate $R_{t,k_i^{start}}$ }
    $R_{t,k_i^{start}} = |CT_{k,t'}|$ for $t' \geq t$
end for
for all ($j \in A$ and $k \in L$ and $t \in T$) **do** {Initialize $prk_{j,t,k}$ }
    $prk_{j,t,k} = k_j^{init}$
end for
for all ($k \in L$ and $t \in T$) **do** {Initialize $evs_{t,k}$ }
    $evs_{t,k} = \sum_j (prk_{j,t,k})$
end for

Pre-processing

During the pre-processing phase, the initialization of the sets and the variables takes place (see Algorithm 2). In more detail, sets $\Delta_k \subseteq \Delta$ and $CT_{k,t} \subseteq \Delta_k$ are created and initialized to the empty set (line 2). Then, all tasks starting at location $k \in L$ are assigned to set Δ_k (line 5), and for each location k and time point t, $CT_{k,t}$ (line 9) is populated with all tasks to be executed at t. Moreover, the set $R_{t,k_i^{start}}$ (line 13) containing the number of tasks remaining to be executed from each location and each time point t, is created and initialized. Finally, variable $prk_{j,t,k}$ (line 16) which holds the initial location of each EV, and variable $evs_{t,k}$ (line 19) which counts the number of EVs parked at each location, are initialized. In the next section, the steps for the assignment of an EV to a task are presented.

Task Execution

Each time an EV j is assigned to a task i, Algorithm 3 is called and a number of actions take place: Initially, EV j is set to be working on task i, by changing the value of variable $\epsilon_{j,i,t}$ from 0 to 1 for the duration of the trip (line 1) (variables $\epsilon_{j,i,t}$, $prk_{j,t,l}$ and λ_i are also used here). Then, the parked location k of j is updated based on the end location k_i^{end} of the task and the arrival time $t_{current} + \tau_i$ (lines 3, 4). Finally, the total number of completed tasks is increased by one (line 6), and the total number of EVs parked at the start and end location of the trip are updated accordingly (lines 8, 9). Note that following the same modeling of the problem as in the MIP formulation (Equation 15), once an EV arrives at a destination, it stays there for at least one time point which is used for the necessary battery swap.

Algorithm 3 Task execution.

1: $\epsilon_{j,i,t'} = 1\ ,\ \forall t' \geq t, t' \leq t + \tau_i$ {j is assigned to i.}
2: {Parked location of agent j is updated.}
3: $prk_{j,t',k_i^{start}} = 0\ ,\ \forall t' \geq t, t' < t + \tau_i$
4: $prk_{j,t',k_i^{end}} = 1\ ,\ \forall t' > t + \tau_i$
5: {The number of completed tasks is increased by one.}
6: $taskSum \leftarrow taskSum + 1$
7: {Number of EVs at start and end locations is updated.}
8: $evs_{t,k_i^{start}} \leftarrow evs_{t,k_i^{start}} - 1\ ,\ \forall t' \geq t$
9: $evs_{t,k_i^{end}} \leftarrow evs_{t,k_i^{end}} + 1\ ,\ \forall t' > t + \tau_i$
10: **Return** ϵ , prk , $taskSum$, $evs_{t,k_i^{end}}$, $evs_{t,k_i^{start}}$

Algorithm 4 Greedy Scheduling Algorithm - Battery Swap.

Require: Δ and A and L and T and $\forall i \in \Delta$, τ_i, and $\forall j \in A$, $k_j^{initial}, \tau_j^{max}$, and $\forall k \in L$, c_k.

for all $(t \in T)$ **do**
  for all $(k \in L)$ **do** {If available EVs at start, $\geq$ total task number.}
    if $(evs_{t,k_i^{start}} \geq R_{t_i^{start},k_i^{end}})$ **then**
      for all $(i \in CT_{k,t})$ **do**
        If $(evs_{t+\tau_i,k_i^{end}} < c_{k_i^{end}})$, search for an j to execute i:
        $(prk_{j,t,k} = 1$ AND $\tau_{j,t_i^{start}} \geq \tau_i)$
        If an EV is found, execute task.
      end for
{If available EVs at start $<$ total task number.}
    else
{Calculate $score_k$ using Equation 22 and assign to sc_k.}
      for all $(k \in L)$ **do**
        $sc_k = \{k, score_k\}$.
      end for
      Sort sc_k in descending order based on $score_k$.
      for all $(k \in sc_k)$ **do**
        for all $(i \in CT_{k,t})$ **do**
{If the end location of the task is equal to the location at the sorted score array, then this task should be executed.}
          if $(k_i^{end} == k)$ **then**
            for all $(j \in A)$ **do**
              If $evs_{t+\tau_i,k_i^{end}} < c_{k_i^{end}}$, Search for an j:
              $(prk_{j,t,k} = 1$ AND $\tau_{j,t_i^{start}} \geq \tau_i)$
              to execute the first task in $CT_{k,t}$ (then remove tasks from list).
              If an EV is found, call Algorithm 3.
            end for
          end if
        end for
      end for
    end if
  end for
end for
Return ϵ , prk , $taskSum$

The Off-Greedy-Swap Algorithm

Here, we elaborate on the key steps of the greedy algorithm (Algorithm 4). For all $i \in CT_{k,t}$ awaiting to be executed at the current time point, the following steps are executed repetitively for each task:

1. If the number of available EVs with enough range is greater than the number of tasks remaining to be executed, starting from this location, then all tasks are set to be executed sequentially (lines 3-8). For each task, if its execution does not lead to a violation of the maximum capacity of the destination location (waiting queues do not exist), then, once there exists an EV that is parked at the current location, it is assigned to the task and the latter is executed by calling Algorithm 3. If no such EV exists, the task is not completed.
2. If the number of the EVs parked at the current location and have enough range is less than the tasks remaining to be executed (lines 9-21), the scores for the end location of each task are calculated. Then, sc_k in descending order based on $score_k$ (lines 10-14). If the execution of the task does not violate the maximum capacity of the destination location, the tasks with the higher scores are executed based on EV availability. Once there exists an EV that is parked at the current location, it is assigned to the task and the latter is executed by calling Algorithm 3. Otherwise, the task is not completed.

The execution of Algorithm 4 generates a schedule for all EVs as well as for the total number of completed tasks. In case where after the max flow algorithm (Section 4), the Greedy algorithm gets as input a set of tasks that all of them can be executed, the problem of scheduling EVs to tasks becomes a simple resource allocation problem. In this case, the heuristic function is not used as enough EVs to execute all tasks always exist (i.e., lines 9-21 of Algorithm 4 are never executed). Thus, at every time point EVs are simply assigned to the tasks.

Next, the analysis of the average case time complexity of the greedy algorithm is presented.

Complexity Analysis – Battery Swap

While the algorithm is greedy, some steps such as the sorting of $score_k$, can be computationally costly. Hence, in what follows, we elaborate on the complexity of key steps of this algorithm.

Theorem 1. (Complexity class of Algorithm 4)

The average case complexity of Algorithm 4 is $|\Delta| \times |L| \times |A| + |T| \times |L|^2 \times log|L| + |L|^2 \times |T|$.

Proof. For lines 1 and 2 of Algorithm 4, the number of iterations is fixed to $|T| \times |L|$. Now, for each t and l_k the iterations for line 4 depend on $|CT_{k,t}|$. However, for all T and L they are always equal to $|\Delta|$. The calculation of the scores (lines 10-13), depend on $|L| \times |CT_{k,t}|$ (line 12), and the sorting of $score_k$ (line 14) on $|L| \times log|L| + |L|$ (i.e., $|L| \times log|L|$ for the sorting of $score_k$ and $|L|$ of the update of the order of locations). The selection of the tasks and the selection of the EVs to execute the selected tasks (lines 15-25) depend on $|L| \times CT_{k,t} \times |A|$. Thus, the computational cost is given by the following equation:

$$\text{Cost} = |T| \times |L| \times (CT_{k,t} + |L| \times CT_{k,t} + |L| \times log|L| + |L| + |L| \times CT_{k,t} \times A) \quad (23)$$

which is equal to:

$$\text{Cost} = |T| \times |L| \times CT_{k,t}(1 + |L| + |L| \times |A|) + |T| \times |L| \times (|L| \times log|L| + |L|) \quad (24)$$

In the average case, where tasks and EVs are equally distributed across locations and time points, $CT_{k,t} = \frac{|\Delta|}{|T| \times |L|}$ and Equation 29 becomes:

$$Cost = |T| \times |L| \times \frac{|\Delta|}{|T| \times |L|} \times (1 + |L| + |L| \times |A|) + |T| \times |L|^2 \times log|L| + |L|^2 \times |T| \quad (25)$$

which is equal to:

$$Cost = |\Delta| \times (1 + |L| + |L| \times |A|) + |T| \times |L|^2 \times log|L| + |L|^2 \times |T| \quad (26)$$

Finally, as $|A| >> 1$ and $|L| >> 1$ Equation 26 becomes:

$$Cost = |\Delta| \times |L| \times |A| + |T| \times |L|^2 \times log|L| + |L|^2 \times |T| \quad (27)$$

Therefore, the complexity increases proportionally (linearly) with the number of tasks and EVs, and quadratically with the number of locations times $log|L|$ times the time points and quadratically with the number of location times the number of time points. Therefore, the average case complexity of Algorithm 4 is $O(|\Delta| \times |L| \times |A| + |T| \times |L|^2 \times log|L||L|^2 \times |T|)$. These results are also verified experimentally (see Section 7.1).

In the next section, a variation of the greedy scheduling algorithm, where battery charge, instead of battery swap is used, is presented.

5.5. Greedy Scheduling with Battery Charging

In this section, we present Algorithm 5 which uses battery charge instead of battery swap. Algorithm 5 works similarly to Algorithm 4, but there is one main difference: instead of swapping batteries after each task is completed, the battery level is monitored and updated for each EV based on its status (i.e., parked or traveling) (lines 27-36). In other words, as an EV travels, the battery discharges, while when an EV is parked the battery level may increase if charging takes place. For this reason, variable $b_{j,t} \in \{0, 100\}$ is used in order to capture the level of battery charge at each EV and each time point. In case energy is needed and when being parked at any location, energy is charged into the EV (lines 29-31), whereas when traveling, the battery is discharged (lines 31-33). In order to decide on the charging schedule of an EV, we use one of the algorithms presented in [39]. According to this algorithm, and assuming that an EV needs n units of energy, the charging takes place in the first n time points after the arrival of the EV to the station, during which enough available chargers exist and enough energy can be provided from the grid. In contrast to the *Off-Opt-Charge*, here the charging rate is fixed to the maximum one, however this does not affect the task execution ability of the algorithm (i.e., the greedy is always using the maximum rate, while the optimal is using it only when is needed. Given that it is optimal and has full knowledge of future demand, if there is no need to use maximum charging rate, it does not do so). The problem of efficient load balancing across the electricity network is not studied in this paper, and therefore the use of more advanced charging scheduling algorithms will be considered in future work. Also, in contrast to Algorithm 4, in order to decide whether an EV can execute a task or not, its current range and not its maximum range (as was happening in the case of battery swap where a fully charged battery was swapped into the EV) is compared with the energy demand of the upcoming task (lines 5 and 20). We will refer to this algorithm as *Off-Greedy-Charge*.

Complexity Analysis – Battery Charge

Here the complexity class of Algorithm 5 is calculated.

Theorem 2. (Complexity class of Algorithm 5)

The average case complexity of Algorithm 5 is $O(|\Delta| \times |L| \times |A| + |T| \times |L| \times (|L| \times log|L| + |L| + |A|))$.

Algorithm 5 Greedy Scheduling Algorithm - Battery Charge.

Require: Δ and A and L and T and $\forall i \in \Delta$, τ_i, and $\forall j \in A$, $k_j^{initial}, \tau_j^{max}$, and $\forall k \in L$, c_k.

for all $(t \in T)$ **do**
  for all $(k \in L)$ **do** {If available EVs at start, $\geq$ total task number.}
    if $(evs_{t,k_i^{start}} \geq R_{t_i^{start},k_i^{end}})$ **then**
      for all $(\delta_i \in CT_{k,t})$ **do**
        If $evs_{t+\tau_i,k_i^{end}} < c_{k_i^{end}}$, search for an j to execute i: ($prk_{j,t,k} = 1$ AND $b_{j,t} \geq \tau_i$)

        If an EV is found, execute task.
      end for
    {If available EVs at start $<$ total task number.}
    else
    {Calculate $score_k$ using Equation 22 and assign to sc_k.}
      for all $(k \in L)$ **do**
        $sc_k = \{k, score_k\}$.
      end for
      Sort sc_k in descending order based on $score_k$.
      for all $(k \in sc_k)$ **do**
        for all $(i \in CT_{k,t})$ **do**
        {If the end location of the task is equal to the location at the sorted score array, then this task should be executed.}
          if $(k_i^{end} == l_k)$ **then**
            for all $(j \in A)$ **do**
              If $evs_{t+\tau_i,k_i^{end}} < c_{k_i^{end}}$, Search for an j:
              ($prk_{j,t,k} = 1$ AND $b_{j,t} \geq \tau_i$) to execute the first task in $CT_{k,t}$ (remove tasks from list).
              If an EV is found, call Algorithm 3.
            end for
          end if
        end for
      end for
    end if
    {For all EVs the battery state is updated, based on whether the EV is driving or not, and based on whether energy is needed}
    for all $(j \in A)$ **do**
      if $\sum_{k \in L} prk_{j,t,k} = 1$ AND $b_{j,t} + 1 * \times ch_j \leq 100$ **then**
        $b_{j,t} = b_{j,t-1} + 1 * \times ch_j$
      else
        $b_{j,t} = b_{j,t-1} - 1 * \times con_j$
      end if
    end for
  end for
end for
Return ϵ , prk , $taskSum$

PROOF. Following a similar thinking as in the previous case, the computational cost is:

$$\text{Cost} = |T|\times|L|\times(CT_{k,t}+|L|\times CT_{k,t}+|L|\times log|L|+|L|+|L|\times CT_{k,t}\times|A|+|A|) \tag{28}$$

which is equal to:

$$\text{Cost} = |T|\times|L|\times CT_{k,t}(1+|L|+|L|\times|A|)+|T|\times|L|\times(|L|\times log|L|+|L|+|A|) \tag{29}$$

In the average case, where tasks are equally distributed across locations and time points, $CT_{k,t} = \frac{|\Delta|}{|T|\times|L|}$ and Equation 28 becomes:

$$Cost = |T|\times|L|\times\frac{|\Delta|}{|T|\times|L|}\times(1+|L|+|L|\times|A|)+|T|\times|L|\times(|L|\times log|L|+|L|+|A|) \tag{30}$$

and as $|A| >> 1$ and $|L| >> 1$ it is equal to:

$$\text{Cost} = |\Delta|\times|L|\times|A|+|T|\times|L|\times(|L|\times log|L|+|L|+|A|) \tag{31}$$

Therefore, the average case complexity of Algorithm 5 is $O(|\Delta|\times|L|\times|A|+|T|\times|L|\times(|L|\times log|L|+|A|))$. Note, that the average case complexity of Algorithm 5 is higher compared to the one of Algorithm 4. However as we show in Section 7.1, the practical difference is very small.

In the next section, a local search technique to further improve the solution quality of the non-optimal algorithms is presented.

5.6. Local search algorithm

Local search can be used on problems that can be formulated as finding an optimal solution among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a stopping condition is met. Local search is a well established technique to improve the solution provided by a non-optimal algorithm. In our case, we need a local search algorithm in order to improve an initial solution that was calculated very fast by the greedy algorithm. Thus, we developed a tabu search-based algorithm. Tabu search is a meta-heuristic

technique which includes as a subroutine a local search procedure which is appropriate for the problem being solved. This subroutine runs iteratively and finds new solutions in the neighborhood of a current solution. A key characteristic of tabu search is that it does not require each new trial solution to be better than the current one. In this way the algorithm can escape from local minima [24]. Tabu-search was preferred over alternative local search methods (e.g., simulated annealing) because 1) its representational characteristics better match our problem formulation and 2) it is known to outperform other local search algorithms in terms of solution quality / robustness and computation time [31]. It has also been proved to perform well in problems with similarities to ours, such as the TSP and the QAP problems [23], [32]. In what follows, we present the formulation of our tabu search-based algorithm for the MoD problem:

1. *Local search procedure*: At each iteration, select the neighboring solution that leads to the maximum number of serviced tasks (or maximum EV utilization).
2. *Neighborhood structure*: An immediate neighbor of the current trial solution is one that is reached by changing the task assignment of one EV.
3. *Form of tabu moves*: All EVs and their assigned tasks.
4. *Addition of a tabu move*: We iterate over all EVs and we calculate the task-to-EV assignment using any of the Off-Opt-Swap or Off-Opt-Charge to solve the problem. The EV and its assignment that leads to higher number of total serviced tasks is added to the tabu list.
5. *Maximum size of the tabu list*: The number of EVs (i.e., for each EV, at most one tabu move exists - for some EVs it may not contain any move).
6. *Stopping rule*: A number of iterations without any improvement in the current solution.

Algorithm 6 presents the local search procedure for the MoD scheme. This algorithm takes as input the task-to-EV assignment as this is calculated by the Greedy algorithm. Then all tasks that have been assigned to an EV are added to set $EVsTasksMap_j$. All tasks that were not assigned to any EV are added to set $TasksUnexecuted$ (lines 2-12). Then, Off-Opt-Swap or Off-Opt-Charge is called for each EV and for all tasks that were initially

assigned to that EV plus the ones that were not assigned to any EV.[14] The completed tasks for each EV, as calculated by the optimal algorithm, are added to *currentSolution* variable. After all EVs have been examined, the one that leads to higher increase in the number of executed tasks and is not in the tabu list, is selected and the current solution is updated based on the assignment of that EV. Also, this assignment is added to the tabu list. This procedure continues until at least n iterations over all EVs have finished without any improvement in the current solution. In that case we assume that the solution cannot further improve and the algorithm terminates (lines 13-25). In the next section, a battery swap optimization algorithm which applies to all scheduling algorithms using battery swap, is presented.

5.7. Battery Swap Optimization

The algorithms that make use of battery swap, calculate the schedule of each EV assuming that after every stop a fully charged battery is swapped into it. However, battery swap at each station may not actually be necessary as the vehicle might have enough energy to execute more tasks. Moreover, frequent battery swapping demands higher volumes of available batteries at the stations, thus increasing the cost for the MoD company [56]. For these reasons, we present an optimal scheduling algorithm which takes as input the EV's travelling schedule $taskExec_{j,t} \in \{0, 1\}$ and minimizes the number of battery swaps a posteriori.

We formulate the problem of the battery swap minimization using MIP techniques and we solve it optimally. We denote two decision variables: 1) $swap_{j,t} \in \{0, 1\}$ which is a binary decision variable on whether EV j swaps its battery at time point t. 2) $bt_{j,t} \in \{0, range_j\}$ which is an integer decision variable on the battery level of each EV at each time point. The value of $bt_{j,t}$ must always be between 0 and the maximum range of the EV and CPLEX will limit the values within this range. The objective function (Equation 32) which consists of the sum of all battery swaps for all EVs, is minimized subject to three constraints: No EV can swap its battery when executing a task (Equation 33), while the initial battery level of each EV must be equal to its full range (i.e., task execution starts with a fully charged battery) (Equation 34). Moreover, at each time point, the battery level of

[14] We believe that the unassigned tasks along with the already assigned tasks could be a source of improving the solution for one EV alone.

Algorithm 6 Local search algorithm.

Require: $\epsilon_{j,i,t}$, A, L, T

$bestSolution = \epsilon_{j,i,t}$
for all $(i \in \Delta)$ **do**
    $Executed = False$
    for all $(j \in A)$ **do** {Collect the tasks executed by each EV }
        if $\epsilon_{j,i,t_i^{start}} = 1$ **then**
            $EVsTasksMap_j \longleftarrow EVsTasksMap_j + \delta_i$
            $Executed = True$
        else{Collect the unexecuted tasks }
            $tasksUnexecuted \longleftarrow tasksUnexecuted + \delta_i$
        end if
    end for
end for
while !(No improvement for n rounds) **do**
    for all $(j \in A)$ **do**
        Call MIP for $EVsTasksMap_j \cup TasksUnexecuted$
        Assign number of completed tasks in $currentSolution_j$
    end for

    Find the assignment that maximizes the number of completed tasks. If this max assignment not in the tabu list, let it be $currentSolution_k$
    Update $EVsTasksMap_k$ and $TasksUnexecuted$
    $tabuList_k = EVsTasksMap_k$
    if $(currentSolution_k > bestSolution)$ **then**
        $bestSolution = currentSolution_k$
    end if
end while

return *bestSolution*

each EV must be equal to the level the previous time point, plus full range if a battery was swapped into the EV and minus the energy consumed while driving (Equation 35) (i.e., CPLEX will limit the values of $bt_{j,t}$ between 0 and $range_j$).

Objective functions:

$$min \sum_{j}(\sum_{t}(swap_{j,t})), \forall j, \forall t \tag{32}$$

Subject to:

$$swap_{j,t} + taskExec_{j,t} \leq 1, \forall j, \forall t \tag{33}$$

$$bt_{j,t=0} = range_j, \forall j \tag{34}$$

$$bt_{j,t} = bt_{j,t-1} + swap_{j,t-1} \times range_j - taskExec_{j,t} \times cons_j, \forall j, \forall t \tag{35}$$

Having discussed a number of offline approaches to solving the EV allocation in MoD schemes, we next turn to the online problem. In particular, we address the problem of coping with unknown future trip requests.

6. Online Scheduling Algorithm

So far, we assumed complete knowledge of future customer demand to exist. However, as this assumption does not always hold in reality, here we present an algorithm which has the ability to effectively cope with the uncertainty in future demand.

This algorithm uses concepts related to the Model Predictive Control (MPC) [11] approach. In more detail, assuming that the MoD company has full knowledge of the demand for only a small number of tasks $\Delta_{fix} \subseteq \Delta$, the remainder of (usually longer term) future demand is stochastic in nature and, to some degree, can be captured using a probability distribution learnt from historical data - expected set of tasks, Δ_{exp}. Note that the start time of all expected tasks is after the execution of the latest task in the fixed set. Given these two sets of tasks, the aim is to calculate a schedule for the execution of the fixed tasks, such that it leads to higher probability of executing more tasks in the future. To achieve this, two steps are followed (see also Algorithm 7):

- Step 1: a set of expected tasks is calculated and a plan for the execution of both fixed and expected tasks is calculated (line 3).
- Step 2: the execution plan related to the fixed tasks remains unchanged and a number $\eta \in H$ of sets of expected tasks (i.e., Δ_{exp}) are iteratively sampled to calculate the average expected number of executed tasks (lines 4-7).

Algorithm 7 Tasks Scheduling Algorithm with Uncertainty.

1: **for** $\forall r \in Rounds$ **do**
2: $Sum_r = 0$ and $Average_r = 0$
3: Call Optimal with $\Delta_{fix} + \Delta_{exp}$
4: **for** $\forall \eta \in H$ **do**
5: Keep schedule for Δ_{fix} unchanged and call Optimal with $\Delta_{exp}(\eta)$.
6: $Sum_r = Sum_r + \sum(\lambda_{\Delta_{exp}})$ {Number of completed tasks for each η is added to Sum_r.}
7: **end for**
8: $Average_r = Sum_r/|H|$.
9: **end for**
10: **Return** Schedule for Δ_{fix} where $Average_r = argmax(Average)$

This procedure (steps 1 and 2) is executed for a number of rounds $r \in R$. Then, the schedule of the fixed tasks that leads to a higher expected utility (number of tasks expected to be executed) is selected. The fixed tasks that have been scheduled to be executed can definitely be executed, under the assumption that no overbooking from the side of the stations, nor booking cancellations from the side of the customers exists. If a booking is canceled, then the schedule may not be feasible anymore. However, we do not study this scenario in this paper.[15] Moreover, a longer time period for the fixed tasks usually leads to higher number of executed tasks, but also leads to higher waiting time for future EVs to announce their request. This algorithm works both with battery swap and battery charge. We will refer to the battery swap variation as *On-Swap* and to the one with battery charging

[15]If a booking cannot be covered (for example in a real-world implementation where traffic conditions don't permit it) we could either pay a compensation or ask users to provide a larger time window for the departure time.

as *On-Charge*. In the next section, we present a detailed evaluation of all algorithms.

7. Evaluation

Here we evaluate our algorithms on a number of settings in order to determine their ability to handle potentially large numbers of tasks, locations and EVs. To this end, we use real locations of pick-up and drop-off points owned by ZipCar[16] in Washington DC, USA which are available as open data,[17] while the distance and duration of all trips[18] were calculated using Google maps (see Figure 2). Car station locations too close to each other were ignored and 8 stations were selected[19]. Note, that Washington DC is one of the cities with the highest traffic congestion in the USA[20] [43]. MoD can reduce congestion as it reduces the demand for parking spots (such parked cars create some congestion). Therefore, a MoD scheme would fit perfectly in such a setting as it has the potential to reduce the congestion caused by privately owned vehicles. The evaluation of our algorithms is executed in six main parts:

- EXP1: The execution time and the scalability of the max flow, the optimal, the incremental MIP, the greedy and the local search algorithms are evaluated.
- EXP2: The performance of the algorithms in terms of the average number of completed tasks and the EV utilization is evaluated.
- EXP3: The sensitivity of the max flow algorithm is analyzed.
- EXP4: The efficiency of the battery swap optimization algorithm is evaluated.
- EXP5: The performance of the online algorithm against the optimal offline one is evaluated.
- EXP6: The correctness of the optimal and greedy algorithms is verified.

[16] http://www.zipcar.com/find-cars/dc.

[17] http://opendata.dc.gov/datasets/.

[18] combinations of some locations acting as start or end points for a trip.

[19] All data regarding the locations and the trips used in the following experiments can be found here http://intelligence.csd.auth.gr/files/datasets/ev/AIJ-data.zip.

[20] http://mobility.tamu.edu.

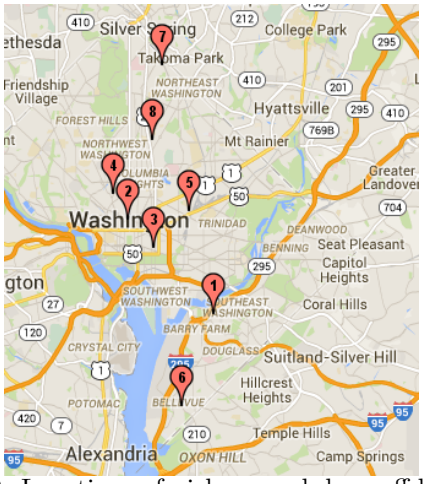


Figure 2: Locations of pick-up and drop-off locations.

All experiments were executed in the following setting: 1) one time point was selected to be equal to 15 mins, and in total 58 time points exist (i.e., equivalent to the execution of the MoD service from 7:00 to 18:00). 2) For the optimal algorithms, the objective function that maximizes task execution (Equation 6) is used (i.e., maximization of completed tasks), unless otherwise stated, and 3) tasks can be formulated based on one of 56 possible trips (i.e., the trips are the combinations of the 8 locations that form the MoD scheme. However, locations too close to each other were ignored). Trips were categorized into three groups: 1) from suburbs to the city centre, 2) around the city centre, and 3) from the city centre to the suburbs. These groups were selected according to our intuition of the expected flow of traffic within the city over the day. Also, we have divided the day into three sessions: 1) morning, 2) noon, and 3) afternoon. In the morning session the majority of the trips come from group 1 (i.e., 60% of the tasks are from group 1 and 40% from the rest), in the noon session they come from group 2 i.e., 60% of the tasks are from group 2 and 40% from the rest), and in the afternoon session they come from group 3 (i.e., 60% of the tasks are from group 3 and 40% from the rest). Within each session, tasks are selected using a uniform distribution (i.e., tasks selected with equal probability). Start times of tasks within each group were drawn from a uniform distribution (i.e., for morning session start times are between $1 - 12$ ($mean = 6.5$ and $\sigma = 3.17$), for noon between $12 - 36$ ($mean = 24$ and $\sigma = 6.9$) and for the afternoon between $36 - 50$ ($mean = 43$ and $variance = 4.04$). Also, $con_j = 10$ and $ch_j = 25$,

Algorithm	**Short name**
MIP with battery swap	Off-Opt-Swap
MIP with battery charge	Off-Opt-Charge
Greedy with battery swap	Off-Greedy-Swap
Greedy with battery charge	Off-Greedy-Charge
Greedy with battery swap and local search	Off-Greedy-Swap-LS
Greedy with battery charge and local search	Off-Greedy-Charge-LS
Incremental-MIP with battery swap	Off-Incr-Swap
Incremental-MIP with battery charge	Off-Incr-Charge

Table 2: Names of the main algorithms used in the evaluation

i.e., for each time point that an EV is working the battery level is reduced by 10 units of energy, and for each time point an EV is charging the battery level is increased by up to 25 units of energy (fast battery charging). The average range of an EV is currently at around 150km. We assume an average speed of 40km/hour which means that an EV can drive for 3.75 hours. In our evaluation setting, one time point is equal to 15 minutes, and 3.75 hours equal to 15 time points. Thus, $con_j = 10\%$ of battery for each time point. A fast charger can fully charge an EV at around one hour. Thus, $ch_j = 25\%$ of the battery for each time point. Both con_j and ch_j are configuration parameters and can be selected by the user. Note, that all experiments were executed on a Windows PC with an Intel i7-4790K CPU and 16 GB of RAM running at 2400MHz.

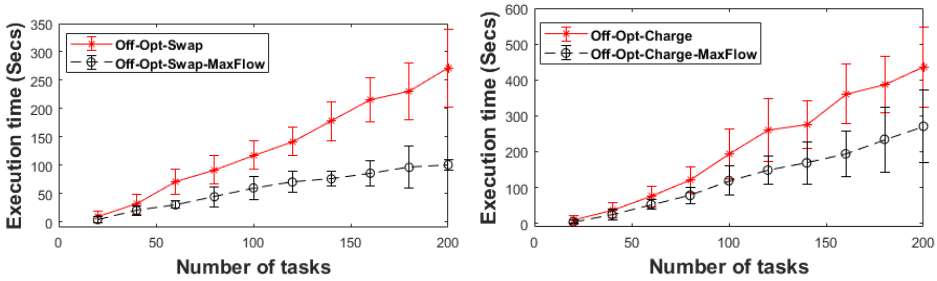
7.1. EXP1: Execution Time and Scalability

Execution time and scalability are typical metrics for scheduling algorithms. Hence, in this set of experiments, we vary the number of tasks and measure the execution time of the Off-Opt-Swap, the Off-Opt-Charge, the Off-Incr-Swap, the Off-Incr-Charge, the Off-Greedy-Swap, the Off-Greedy-Charge, the Off-Greedy-Swap-LS and the Off-Greedy-Charge-LS (Table 2).

7.1.1. Improvement in execution time with pre-processing

Here, we evaluate the improvement in execution time for the Off-Opt-Swap and Off-Opt-Charge when the MaxFlow algorithm is used as a pre-processing step. As can be seen in Figure 3, when the pre-processing step is used, the execution time decreases significantly. For example, for 200 tasks,

the execution time for the Off-Opt-Swap is reduced by 62.9%, and for the Off-Opt-Charge by 60.8%. This is due to the fact that the MaxFlow algorithm prunes the set of tasks. Thus, for the rest of the experiments, we always use the optimal algorithm in combination with the pre-processing step. Note that improvement in execution time is achieved also for the greedy algorithm, but in this case it is very small due to the fact that the execution time was already low.



(a) Battery swap (b) Battery charge

Figure 3: Execution Time with and without pre-processing.

Regarding the pre-processing step itself, the execution time is found to be very low. For example, in a setting with 100 EVs and up to 3000 tasks we see that the execution times of both variations of the MaxFlow algorithm increase linearly (i.e., $R^2 = 0.9971$ for the MaxFlow and $R^2 = 0.992$ for the MaxFlowInit).[21] As can be seen in Figure 4 both have very low execution time as for 3000 tasks both execute in well under 0.35 seconds with the variation where the initial location is optimized always being the slowest, because one more constraint and one more decision variable exists. The optimization of the initial location of EVs, leads to an increase in average task execution. However, doing so can create other problems. For example, relocation of vehicles may be needed prior to the beginning of the operation of the MoD company which incurs some extra cost. Thus, in the following experiments we focus in the case where the initial location is not optimized.

[21]We use MATLAB's Curve Fitting Toolbox. Note that the findings and the reported values of R^2 are guaranteed to be true only for the settings and experiments presented in this paper.

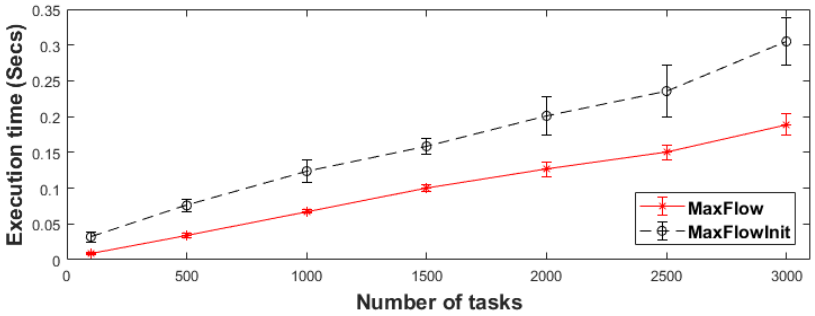


Figure 4: Execution Time Max Flow (100 EVs).

7.1.2. Scalability across varying (small to medium) task sizes

In a setting with 15 EVs and up to 300 tasks, in the case of battery swap and as can be seen in Figure 5, the greedy algorithm is 10 times faster than the incremental-MIP and 1000 times faster than the optimal. The rate of growth of the greedy algorithm is linear, while for the other two algorithms the rate is quadratic (see Table 3). The linear time complexity of the greedy is related to the pre-processing step which prunes all infeasible tasks (see bottom of Section 5.4). The number of tasks shown on the x-axis of the following figures is based on the number of tasks after the pre-processing step. Thus, each point of the plot is the centroid of the execution times of multiple experiments for various actual numbers of tasks. In the case of the incremental-MIP, the pre-processing step leads to a lower number of executed tasks and as will be discussed in the next subsection it is not used in the rest of the experiments. However, here we plot the execution time in combination with the pre-processing step to have a fair comparison with the other algorithms. Regarding the incremental-MIP, we have investigated the use of cut constraints to improve the scalability of this algorithm. In fact, we added two cut constraints: 1) For the tasks that overlap in time, only one of them can be executed. 2) The tasks that are isolated (i.e., they are not in the EV's initial location and there are no tasks leading to that location), are not executed (i.e., decision variable set to zero). The addition of these extra constraints has reduced the MIP pre-processing time, but not the total execution time. For example, in a setting with 2000 tasks and 45 EVs, the MIP pre-processing time has been reduced from 0.352 secs to 0.345 secs, but the total execution time has been increased from 22296.4 secs to 23390 secs. Thus, we decided not to use these extra constraints.

In the case of battery charge and as can be seen in Figure 6, all algorithms

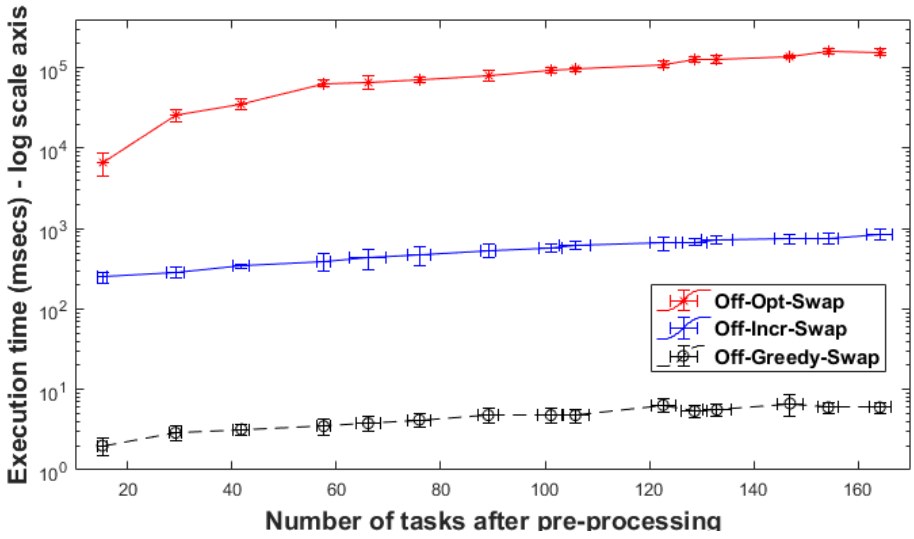


Figure 5: Execution time with battery swap- (15 EVs) - centroids of average values.

Algorithm	Trend	R^2
Off-Opt-Swap	Quadratic	98.42%
Off-Greedy-Swap	Linear	92.13%
Off-Incr-Swap	Quadratic	99.35%

Table 3: All algorithms- battery swap trend (15 EVs).

behave similarly to the battery swap, but the average execution time of all of them is higher (see also Table 4). In the case of battery charge, we also plot the execution time of the greedy algorithm in combination with the local search technique. This combination leads to higher execution times compared to the incremental-MIP, but lower compared to the optimal. In fact, the execution time of the Off-Greedy-Charge-LS initially increases but as the number of tasks increases beyond 120, the execution time decreases. This can be explained by the fact that given the number of EVs remains stable and the number of tasks increases, the greedy algorithm already calculates a schedule which covers most of the EVs' availability and for this reason the local search cannot improve the solution a lot. Thus, it terminates sooner compared to other settings where the space for improvement is larger. As will be described in Section 7.2, when the Off-Greedy-Swap is combined with the MaxFlow and under some assumptions, it provides the optimal solution. Thus, there was no point to combine the local search with battery swap.

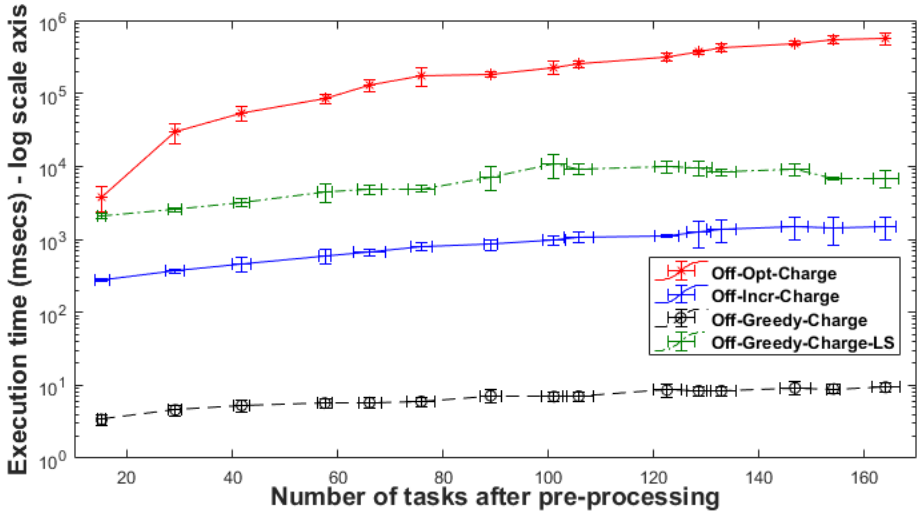


Figure 6: Execution time with battery charge- (15 EVs) - centroids of average values.

Algorithm	Trent	R^2
Off-Opt-Charge	Quadratic	99.24%
Off-Greedy-Charge	Linear	96.75%
Off-Greedy-Charge-LS	Quadratic	79.62%
Off-Incr-Charge	Quadratic	98.64%

Table 4: All algorithms- battery charge trend (15 EVs).

7.1.3. Scalability across varying large task sizes

Here we assume the full range of some EVs (i.e., 20% of the available EVs) not to be long enough to execute some of the longer trips. Thus, the MaxFlow provides the upper limit of all feasible tasks (see more details on this in Section 7.2) (i.e., the Off-Greedy-Swap does not provide the optimal solution any more). We choose a setting with 100 EVs and up to 3000 tasks. For such settings the optimal algorithm does not scale, and the low execution times of the incremental and the greedy algorithms make them the preferable options. When battery swap is used and as can be seen in Figure 7, the Off-Greedy-Swap is 1000 times faster than the Off-Incr-Swap and 10000 times faster than the Off-Greedy-Swap-LS. It is interesting to notice that when local search is used, the execution time initially increases but then, for more than 1300 tasks starts decreasing (see also Table 5). The explanation is similar to the one given in the previous subsection.

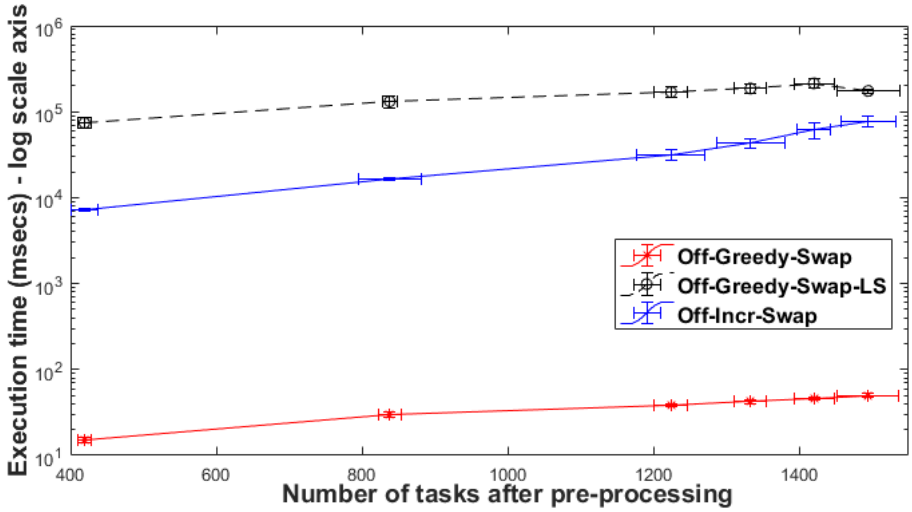


Figure 7: Execution time with battery swap- (100 EVs) - centroids of average values.

Algorithm	Trend	R^2
Off-Greedy-Swap	Quadratic	98.47
Off-Greedy-Swap-LS	Quadratic	91.48
Off-Incr-Swap	Quadratic	95.48

Table 5: All algorithms- battery swap trend (100 EVs).

When battery charge is used and as can be seen in Figure 8, the execution times of all algorithms follow a similar trend (see also Table 6). However, in contrast to battery swap, for large number of tasks the Off-Incr-Charge has higher execution time compared to Off-Greedy-Charge-LS. This can be explained by the fact that Off-Opt-Charge, which is used as part of both algorithms, is more time consuming compared to Off-Opt-Swap and similarly to the previous case, for large numbers of tasks, the local search algorithm terminates earlier. Note, that in the case of Off-Greedy-Swap-LS the time is second degree polynomial but the x^2 term is negative. However, in the case of Off-Greedy-Charge-LS it is positive. This can be explained by the fact that in the case of battery charging (as we will see in Section 7.2) the space for improvement is bigger. Thus, the local search technique runs for longer time. Note that for much larger numbers of tasks we expect the execution time to start dropping as well. We next evaluate the execution time by varying the rest of the dimensions (i.e., EVs, time points and locations).

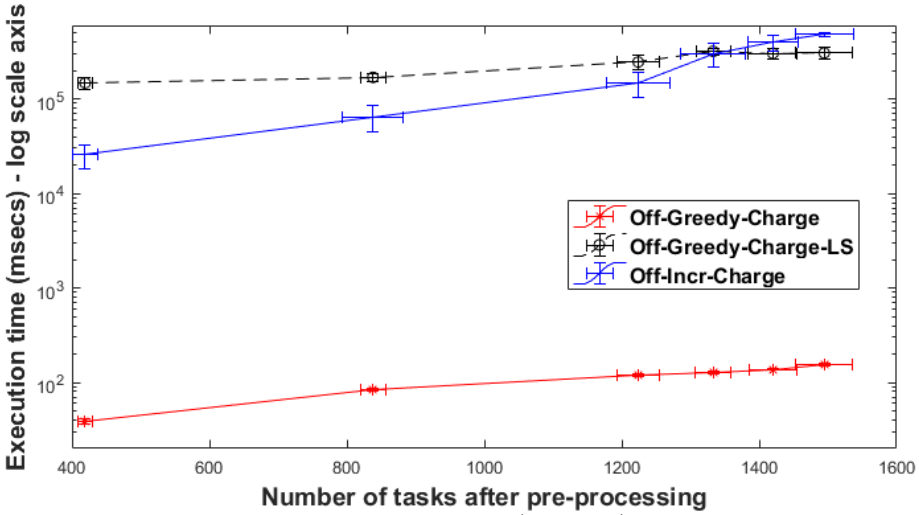


Figure 8: Execution time with battery charge- (100 EVs) - centroids of average values.

Algorithm	Trend	R^2
Off-Greedy-Charge	Quadratic	99.54
Off-Greedy-Charge-LS	Quadratic	92.58
Off-Incr-Charge	Quadratic	96.1

Table 6: All algorithms- battery swap trend (100 EVs).

7.1.4. Scalability across other dimensions

In this section, we evaluate our algorithms in terms of execution time while varying the other dimensions of the problem (i.e., EVs, time points and locations). As can be seen in Figures 9, 10, 11 and in Table 7 in all cases the execution times for the Off-Opt-Swap and the Off-Incr-Swap increase quadratically. However, when the locations vary, the term of x^2 is negative (i.e., execution time initially increases and then decreases). This can be explained due to the fact that when the number of locations increases, but the number of EVs remains unchanged, the tasks spread around too much and fewer of them can be executed making the problem easier to solve. In terms of the Off-Greedy-Swap, when varying the number of EVs and time points, the execution time increases linearly, while when varying the number of locations it increases quadratically. Thus, the theoretical complexity analysis presented in Section 5.4 is verified. The results for battery charge are similar.

	EVs		Time points		Locations	
Algorithm	Trend	R^2	Trend	R^2	Trend	R^2
Off-Opt-Swap	Quadratic	98.39%	Quadratic	99.1%	Quadratic	97.9%
Off-Greedy-Swap	Linear	98.29%	Linear	91%	Quadratic	98.49%
Off-Incr-Swap	Quadratic	98.32%	Quadratic	99.9%	Quadratic	92.17%

Table 7: Variable dimensions - all algorithms- battery swap

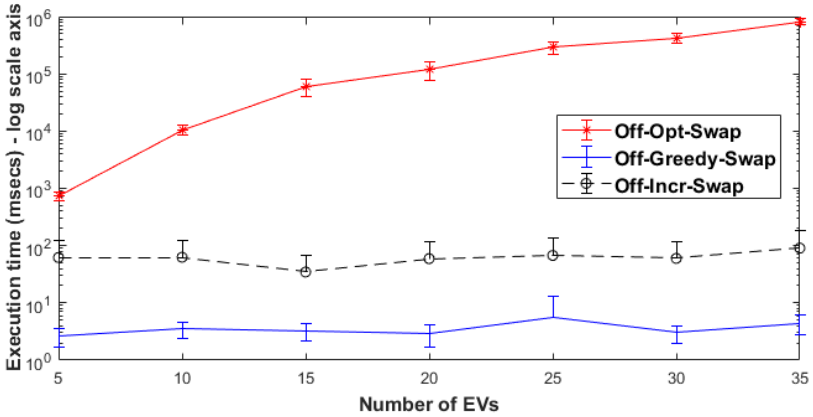


Figure 9: Execution Time MIP (100 Tasks) - Variable number of EVs.

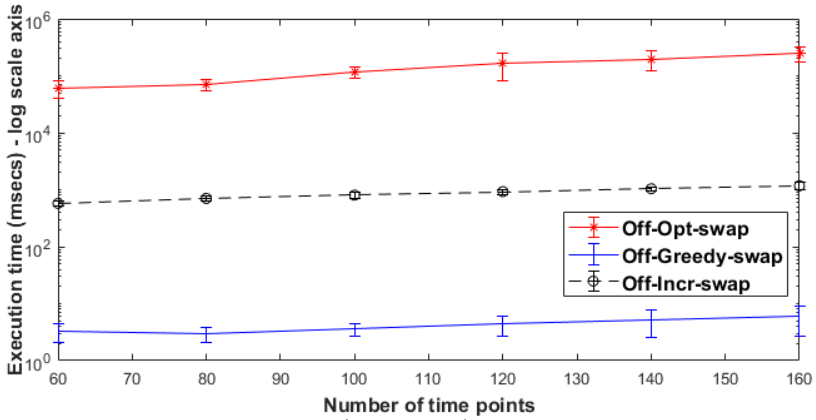


Figure 10: Execution Time MIP (100 Tasks) - Variable number of time points.

7.2. EXP2: Completion of tasks and EV utilization

Here we evaluate all algorithms in terms of average task completion and average EV utilization. In so doing, we have two main scenarios: In the first one we assume that all EVs carry the same battery which, when fully charged, is large enough to execute even the longest task. In the second one

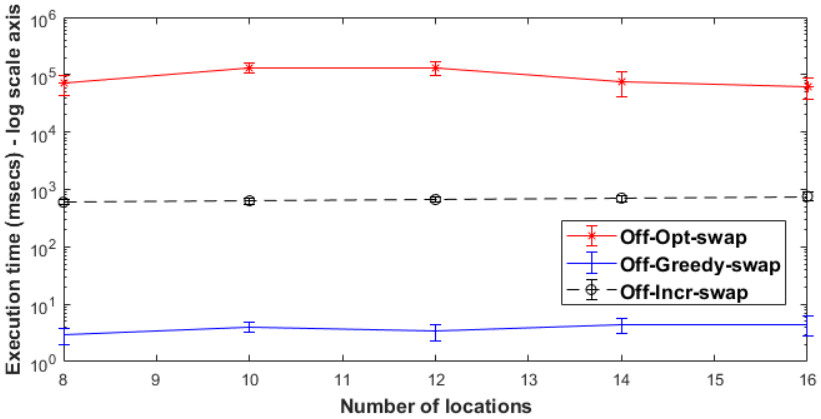


Figure 11: Execution Time MIP (100 Tasks) - Variable number of locations.

we assume that some EVs carry a smaller battery (or a not-fully charged one) and are not able to execute some of the longest tasks.

In the case where all EVs carry the same battery and as can be seen in Figures 12 and 13, when the Off-Greedy-Swap is used after the pre-processing step (i.e., MaxFlow algorithm), it always provides the optimal solution (see explanation in Section 5.4). In contrast, when the Off-Greedy-Charge is used after the pre-processing step, it provides a solution close to the optimal (i.e., 97% of the optimal for 300 tasks). This can be explained by the fact that battery charging is more time consuming compared to the battery swap and not all EVs have full range once they are about to start executing a task. However, when the local search algorithm is used, having as input the solution of the greedy algorithm, the performance of the Off-Greedy-Charge improves and reaches 98.5% of the Off-Opt-Charge. It is interesting to note that when the greedy algorithm is combined with the MaxFlow, the improvement in the number of completed tasks, compared to the case where the MaxFlow is not used and for a setting with 200 tasks, is 26.2% for the Off-Greedy-Swap and 24% for the Off-Greedy-Charge. Regarding the incremental-MIP algorithm, in all cases performs worse compared to both the optimal and the greedy algorithms. For example, for 300 tasks it is at 96.34% of the Greedy-Swap and the Off-Opt-Swap and at 98.74% of the Greedy-Charge without local search and at 97.27% when its solution is improved with local search. When it is used in comparison with the MaxFlow, the results are even worse. This could be explained by the fact that the MaxFlow provides the optimal set of tasks given the full set of EVs. Thus, when this is broken down into sub-problems the solution deteriorates. Thus, in this experimental setting, the

incremental-MIP is not used.

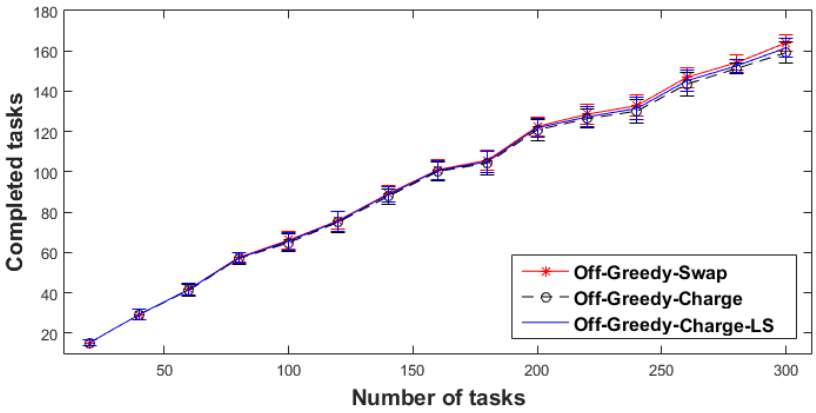


Figure 12: Completed tasks (15 EVs) - Offline algorithms.

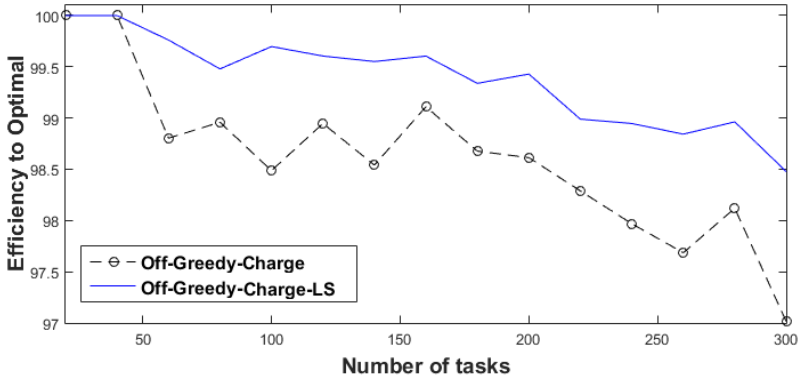


Figure 13: % Efficiency compared to the optimal (15 EVs).

For a larger setting with 100 EVs and up to 4000 tasks, the performance of the algorithms remains similar. The Off-Greedy-Swap in combination with the MaxFlow provides the optimal solution, while the Off-Greedy-Charge is very close to the optimal. In fact, for 4000 tasks the Off-Greedy-Charge is at 99.6% of the optimal, and when the local search is applied, the efficiency increases to 99.75% of the optimal. The better performance of the Off-Greedy-Charge, compared to the smaller setting, can be explained by the fact that for very large numbers of tasks, the options for the greedy algorithm are many and it is not very sensitive on the optimal selection of each task. The fact that the Off-Greedy-Charge does not reach the absolute optimal was expected due to its myopic nature (i.e., local heuristic).

Given the performance of all algorithms, we argue that when battery swap is used, the Off-Greedy-Swap in combination with the MaxFlow is the appropriate solution for all problem sizes as it provides the optimal solution with very low execution time. Whereas, when battery charge is used, the combination of the Off-Greedy-Charge with the MaxFlow and the use of local search afterwords leads to the best solution especially for large problems. However, for small problems (i.e., tens of EVs and few hundreds of tasks) the Off-Opt-Charge is the best choice as it provides the optimal solution in reasonable time (even if it slower compared to the others).

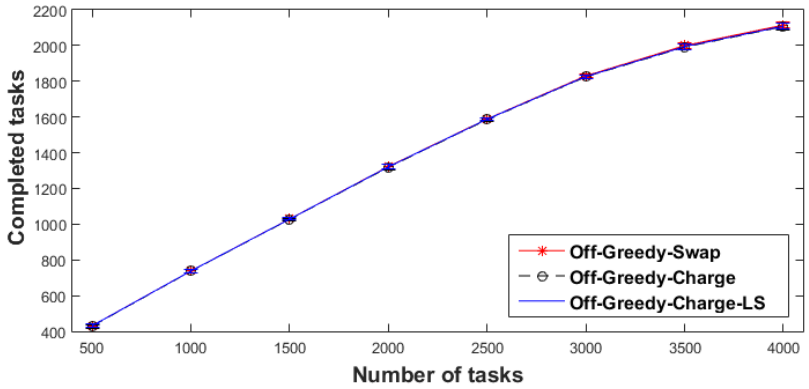


Figure 14: Completed tasks (100 EVs) - Offline algorithms.

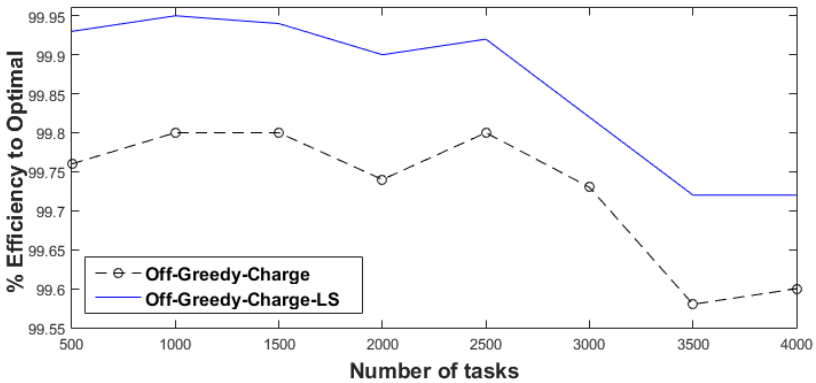


Figure 15: % Efficiency compared to the optimal (100 EVs).

Now, when trying to maximize the EV utilization (i.e., using as objective function Equation 7) and as can be seen in Figure 16, we observe an average 3.6% increase in EV utilization, while the average number of completed tasks is decreased by 1.5%. In other words, the difference between the two

objective functions in terms of EV utilization and task completion are small. However, in settings where tasks with longer duration also exist, we expect this difference to increase, as the solver would have more options to fill the travelling schedule of the EVs with long tasks. Note that in all cases the differences between the battery swap and the battery charge variations in terms of completed tasks are very small. For example, for 15 EVs and 200 tasks both the Off-Opt-Swap and the Off-Opt-Charge executed on average 119,6 tasks (see also Figure 17). On some rare occasions, we have noticed some very slight differences (1 or 2 tasks). This can be explained by the fact that fast charging is used, thus making the delay compared to the battery swap small (i.e., at most 3 time points, as opposed to 1 time point).

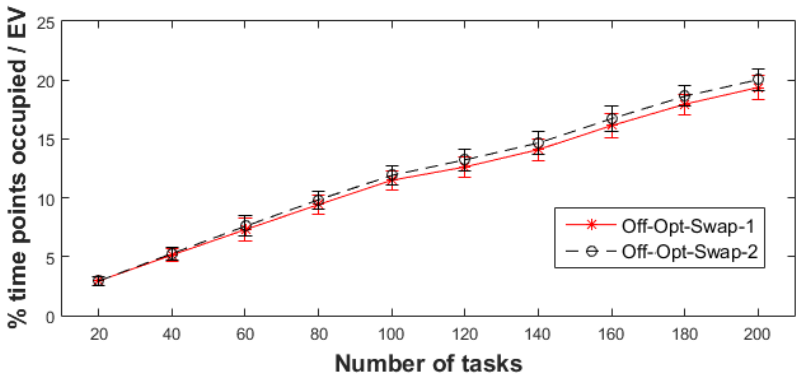


Figure 16: EV utilization (15 EVs) - % time points occupied.

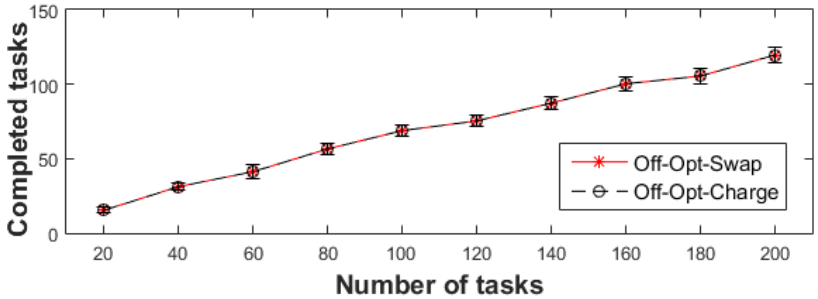


Figure 17: Completed tasks Off-Opt-Swap VS Off-Opt-Charge (15 EVs).

The MaxFlow algorithm treats all EVs equally and assumes that all of them can execute any task. However, when some EVs (i.e., 20% of the available EVs) carry smaller batteries and cannot execute some of the longest tasks, then the set of tasks returned by the MaxFlow is the upper limit of

the tasks that can be executed. An interesting question is how the scheduling algorithms perform in this case. In a setting with up to 200 tasks and 15 EVs and as can be seen in Figure 18, the Off-Opt-Swap always achieves the best performance (i.e., 11% better performance compared to the greedy for 200 tasks). Note that when the MaxFlow returns the upper limit of tasks to be executed, the Off-Greedy-Swap and as was expected, does not provide the optimal solution anymore. The Off-Greedy-Swap-LS and the Off-Incr-Swap have similar performance with the greedy being slightly ahead for smaller number of tasks, while the incremental-MIP matches the performance of the greedy for larger number of tasks. Now, in a setting with up to 3000 tasks and 100 EVs and as can be seen in Figure 19, the performance of the Off-Greedy-Swap is at 99.5% of the Off-Greedy-Swap-LS when 3000 tasks exist. Moreover, the Off-Incr-Swap performs initially worse compared to the greedy algorithm, but for more than 1000 tasks, it has an advantage. However, for more than 2500 tasks it starts leveling off. Note that the standard deviation of the Off-Incr-Swap is higher compared to the Off-Greedy-Swap. Also note that the incremental-MIP algorithm is not used in combination with the local search because the execution time increases 10-times on average, and the solution quality improves by less than 0.3% on average. Given the execution time of the incremental and the greedy algorithms combined with local search, we argue that for a setting where the EVs carry different battery types, and when the number of tasks is larger that 1000, the incremental-MIP is the appropriate solution. As can be seen in Figure 20, the performance for battery charge is similar. We next discuss the sensitivity of the MaxFlow algorithm.

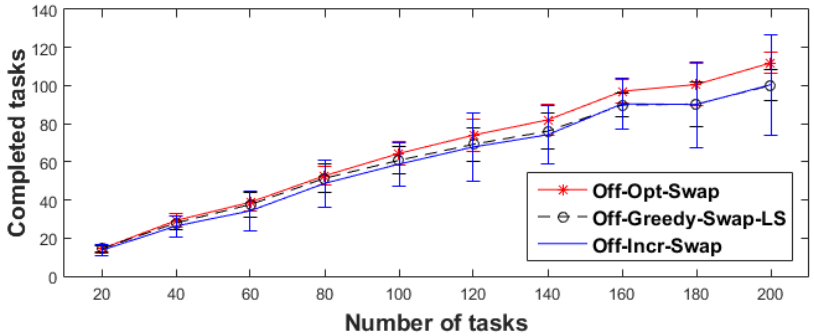


Figure 18: Completed tasks (15 EVs) - Offline algorithms with battery swap.

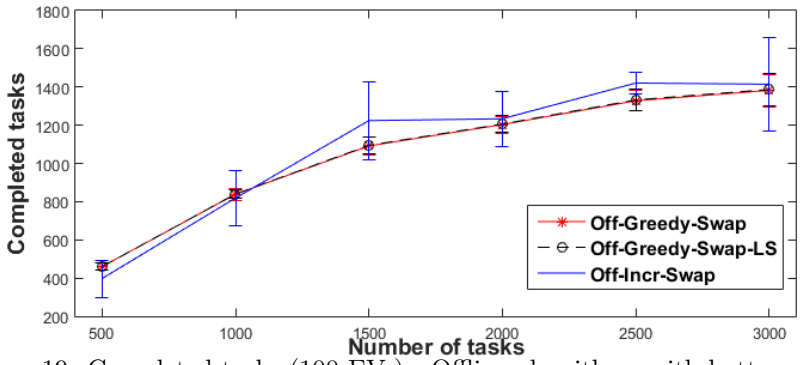


Figure 19: Completed tasks (100 EVs) - Offline algorithms with battery swap.

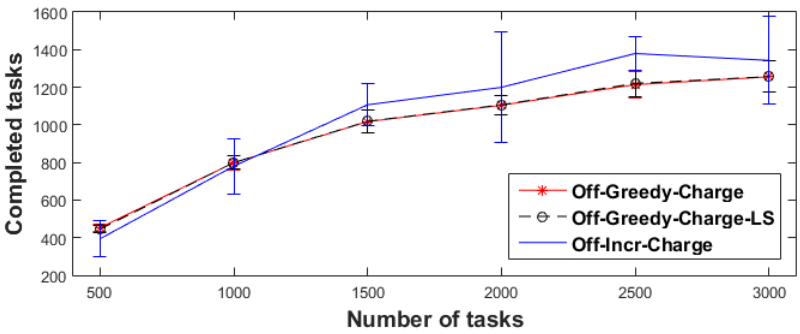


Figure 20: Completed tasks (100 EVs) - Offline algorithms with battery charge.

7.3. EXP3: Sensitivity of the MaxFlow

Here, we evaluate the MaxFlow algorithm in a setting where we fix the number of tasks to be completed to 500, and vary the number of EVs as well as the number of the locations of the pick-up and drop-off points in order to determine how the EV-to-task-allocation problem is affected by the change in the number of EVs and locations. Note that here synthetic data on locations are used. Given that the number of tasks to be completed remains fixed, one could expect that by increasing the number of EVs, the number of completed tasks would also increase. In the case where the number of locations remains fixed this is indeed the case (see Figure 21). However, in the case where the number of locations increases we observe an opposite trend. Interestingly, when 40 EVs and 10 locations exist, the number of completed tasks is actually higher compared to the case where 100 EVs and 60 locations exist. This is an interesting observation which can be explained as follows: 1) as the number of locations increases, the average number of EVs at one location at $t = 0$, decreases and therefore, the probability of an EV being able to execute a future task starting from a given location decreases, and 2) as the number of

locations increases, the number of all possible trips increases exponentially, and therefore, EVs tend to spread around too much. Thus, similarly to the previous point, the probability of an EV being able to execute a future task decreases. Thus, the optimization of the number and the location of the pick-up and drop-off points is an important problem to be solved by any MoD company [21] as it is related to the cost of a possible MoD deployment (i.e., number and locations of stations), as well as its performance in terms of serviced tasks and profit. Our algorithms and experimental results can help towards this. We next discuss the efficiency of the battery swap optimization algorithm.

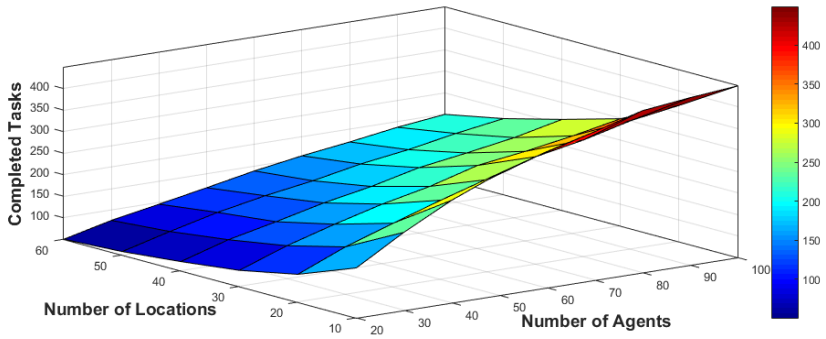


Figure 21: Sensitivity of the MaxFlow.

7.4. EXP4: Battery Swap Optimization

Here, we evaluate our proposed battery swap optimization algorithm against a setting where no such optimization takes place, based on the tasks assigned by the scheduling algorithms to each EV. In doing so, we assume all EVs to carry the same battery type, which has a discharge rate $con_j = 10$ (i.e., at each time point, 10% of the battery is discharged) and the average trip duration to be 2 time points. The battery swap minimization algorithm is shown to achieve a reduction of up to 87.5% (see Figure 22) in the number of necessary battery swaps. Its execution time is well under half second even for large settings. Overall, the minimization of the battery swaps is an important task in order to reduce the cost of battery swapping in a real world deployment. Note that this algorithm can be applied to all scheduling algorithms which use battery swap. In the next section, the optimal offline algorithm is evaluated against the online one.

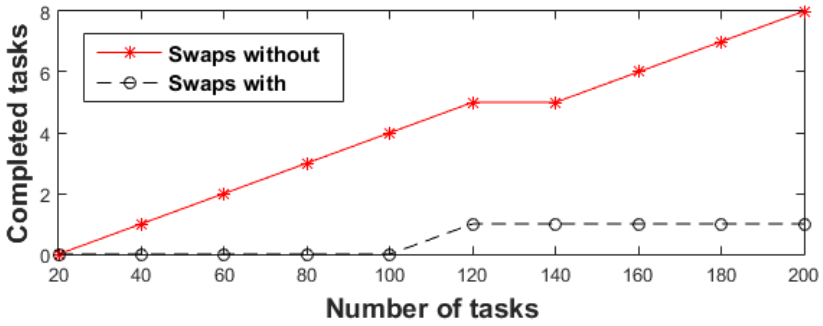


Figure 22: Number of battery swaps with and without optimization.

7.5. EXP5: Online VS Offline Scheduling- Task Execution

Given that full knowledge of future demand is not always possible, here the online algorithm (Algorithm 7) is evaluated against the optimal one. We choose $\eta = 15$ and $r = 5$ (reminder: η represents the number of sets of expected tasks, and r the number of rounds- see Section 6), as these numbers were observed to provide a good balance between performance and execution time. Also, the day is divided into three periods, (i.e., morning, noon, afternoon) and for each round the tasks for one period are fixed, while for the rest are drawn from a distribution (i.e., at first the morning tasks are fixed while noon and afternoon tasks come from the distribution. They are generated from a uniform and a Gaussian distribution: 50% of tasks come from a uniform distribution and 50% come from a Gaussian distribution, while mean and σ are selected in this way so as the tasks to come from future session (i.e., noon, or afternoon and based on the current time point).

As can be seen in Figures 23 and 24, the Online algorithm has an acceptable performance: for a setting with up to 100 tasks, the performance achieved by this algorithm in terms of average task completion, is no less than 90% compared to the optimal offline. Note, that future tasks come 50% from a Gaussian distribution, while the remaining 50% come from a uniform distribution. Thus, we also show that our methodology can efficiency handle tasks coming from different distributions. Also, the execution time for each η is similar to the times presented for the optimal offline algorithms for the same number of tasks. Thus, based on the number and the types of computers the MoD company possesses, η and r can be chosen accordingly so as better accuracy in the prediction to be achieved,[22] while the schedule to

[22]High numbers of η and r lead to better accuracy. However, they demand more com-

be calculated within the available time (i.e., the schedule of the tasks must be decided before the actual execution of them). In the next section, the correctness of the MIP and greedy algorithms is verified.

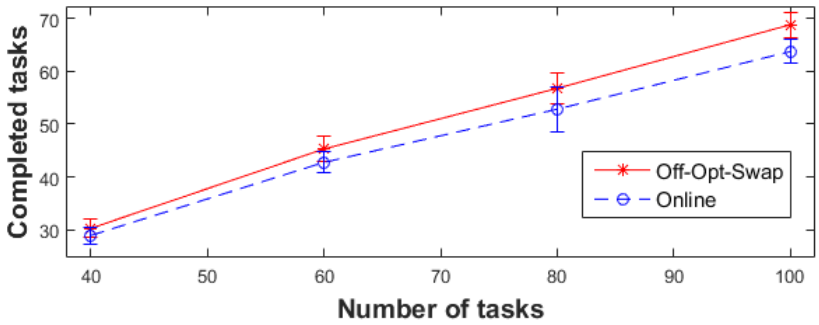


Figure 23: Offline Optimal VS Online algorithm.

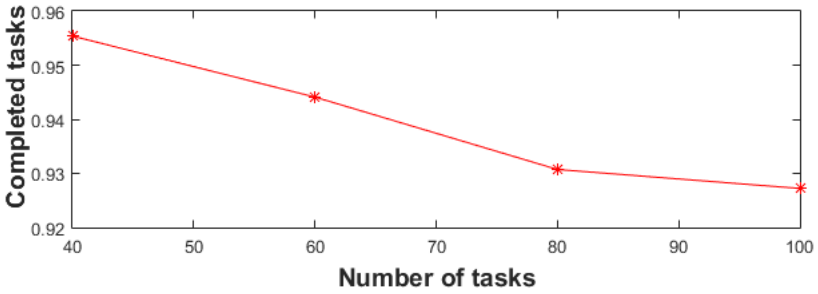


Figure 24: Online efficiency compared to optimal offline.

7.6. EXP6: Correctness of the MIP algorithm

In order to verify the correctness of both the MIP and greedy algorithms we conducted a number of experiments. We created a set of tasks, that could be executed by a single EV (i.e., the end location of one task was the start location of the next one). Then we run both algorithms for this set of tasks and both gave us the optimal solution. Then we expanded the set of tasks (with known outcomes) and we observed that the MIP returned the optimal solution. The greedy could provide the optimal solution up to a certain number of EVs and tasks, but for larger numbers (i.e., more than 20 tasks and more than 1 EV) it was giving sub-optimal solutions as was

putational power. Thus, parallel execution is preferable

expected. In the next section, issues related to the time complexity of the MIP formulations of the problem are discussed.

7.7. Discussion on the Computational Complexity of the Offline Optimal Solution

The MIP formulation for the offline optimal algorithm is solved using CPLEX. CPLEX uses the Branch & Cut optimization method [26] which is, in turn, based on a Branch & Bound search algorithm. Initially, at the root node the problem is solved as a linear program using, e.g., simplex, or the shifting optimizer algorithm (this approach was first introduced by IBM under the name SPRINT approach [2]). Then, if a solution exists but not all variables are integral, Branch and Bound begins by choosing a non-integral variable and fixing its value to the two closest (upper and lower) integers and resolving the problem with the remaining variables. According to [57], the average case complexity of the branch & bound algorithm is polynomial in the depth of the search tree, when the expected number of the children of a node that has the same cost as their parent, is at least 1. If a MIP problem has d variables, then the search space of Branch & Cut consists of the combination of all the possible splits of the problem variables, which accounts to a binary tree with depth d and 2^d nodes. In all our experiments we have observed in the CPLEX log that the value of the objective function at the root node of the Branch & Cut starts with the optimal value. Therefore all branches in the search tree lead to nodes that have the same cost with the root node. Due to this observation, we expect that the computational complexity of the algorithm should be polynomial to the number of variables of the problem. Thus, we next calculate the number of decision variables and examine their correlation to the execution time.

For Off-Opt-Swap, the number d_{swap} of variables is given by the following equation: $d_{swap} = |\Delta| + |\Delta| \times |A| \times |T| + |A| \times |L| \times |T|$ where $|\Delta|$ is the total number of tasks, $|A|$ is the total number of EVs, $|T|$ is the total number of time points that the day is divided in, and |L| is the total number of locations (pick-up and drop-off points) (i.e., the problem has four dimensions in total). Decision variable λ_{δ_i} has 1 dimension namely $|\Delta|$, decision variable $\epsilon_{j,i,t}$ has 3 dimensions namely $|A|$, $|\Delta|$ and $|T|$ and decision variable $prk_{j,t,k}$ has 3 dimensions namely $|A|$, $|T|$ and $|L|$. In order to verify our expectation that the problem grows in polynomial time, we plot the time to solve the MIP problem against the number of the variables d_{swap} of each problem calculated using the above equation. Each time we vary one dimension of

the problem and hold the rest fixed (see Figures 25, 26, 27 and 28). Note that the number of decision variables is after the pre-processing step. For Δ, A and T our results verify (in all cases we observe $R^2 > 95\%$) that the time complexity is $O(d_{swap}^2)$. Therefore, the complexity according to the problem size is $O((\Delta + \Delta \times A \times T + A \times L \times T)^2) = O(\Delta^2 + \Delta^2 \times A^2 \times T^2 + A^2 \times L^2 \times T^2 + 2 \times \Delta^2 \times A \times T + 2 \times \Delta \times A \times L \times T + 2 \times \Delta \times A^2 \times L \times T^2) \cong O(A^2 \times T^2 \times (\Delta^2 + L^2))$. Interestingly though, for the L dimension of the problem although the execution time remains second degree polynomial in the number of decision variables, the coefficient for x^2 gets negative values (see Figure 28). As has been explained earlier (see Section 7.2), when the number of locations increases, the EVs spread around too much and their ability to execute tasks is reduced. Therefore, many infeasible tasks exist and the execution time drops. Thus, we can conclude to the result that the time complexity of the MIP formulation is not related solely to the number of decision variables but also to the *hardness* of the problem to be solved. Next, we evaluate the complexity of the variation with battery charge.

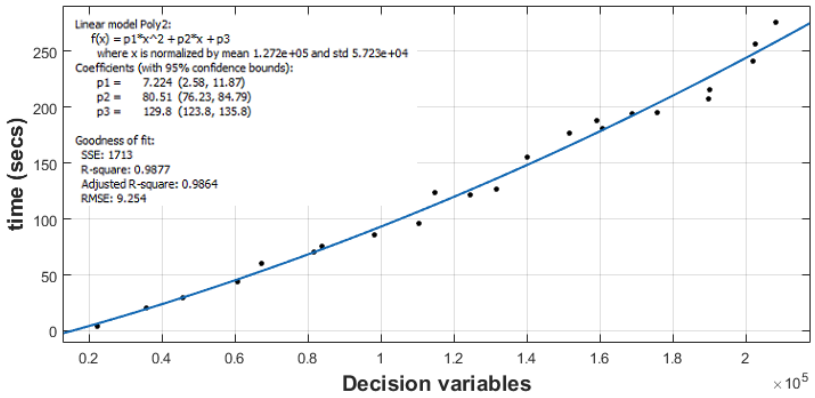


Figure 25: Variable number of tasks and fixed other dimensions - Off-Opt-Swap.

For the Off-Opt-Charge, the number d_{charge} of variables is given by the following equation: $d_{charge} = |\Delta| + |\Delta| \times |A| \times |T| + |A| \times |L| \times |T| + |A| \times |T|$ (i.e., decision variable λ_{δ_i} has 1 dimension namely $|\Delta|$, decision variable $\epsilon_{j,i,t}$ has 3 dimensions namely $|A|$, $|\Delta|$ and $|T|$, decision variable $prk_{j,t,k}$ has 3 dimensions namely $|A|$, $|T|$ and $|L|$ and decision variable $bch_{j,t}$ has 2 dimensions namely $|A|$ and $|T|$). We executed the same experiments as in the case of Off-Opt-Swap and the results also verify (in all cases we observe $R^2 > 96\%$) that for Δ, A and T the time complexity is $O(d_{charge}^2)$. Therefore, the complexity according to the problem size is $O((\Delta + \Delta \times A \times T + A \times L \times T + A \times T)^2) =$

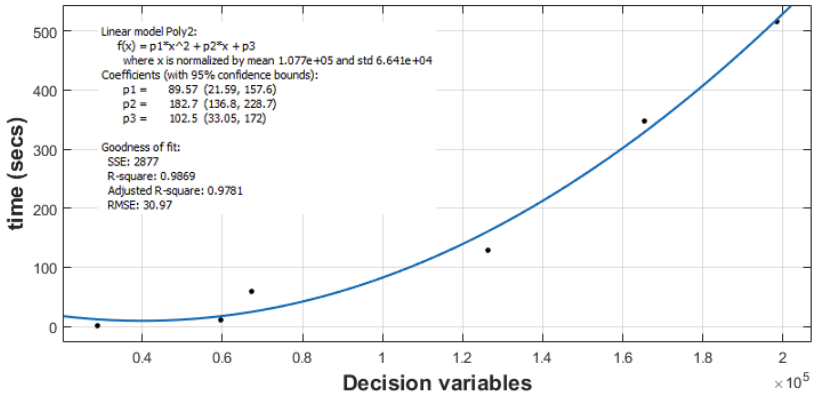


Figure 26: Variable number of EVs and fixed other dimensions - Off-Opt-Swap.

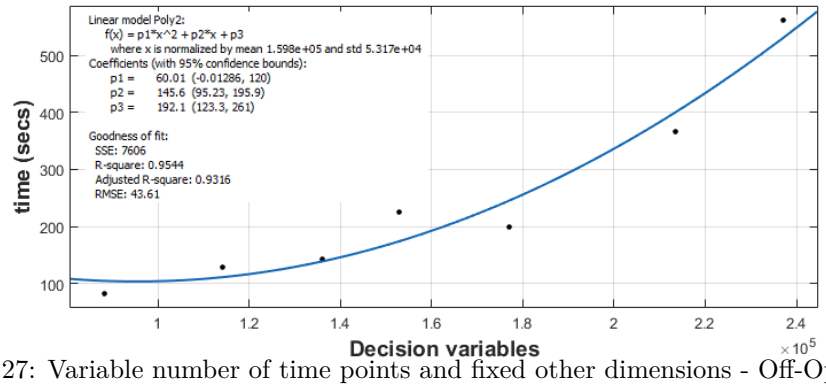


Figure 27: Variable number of time points and fixed other dimensions - Off-Opt-Swap.

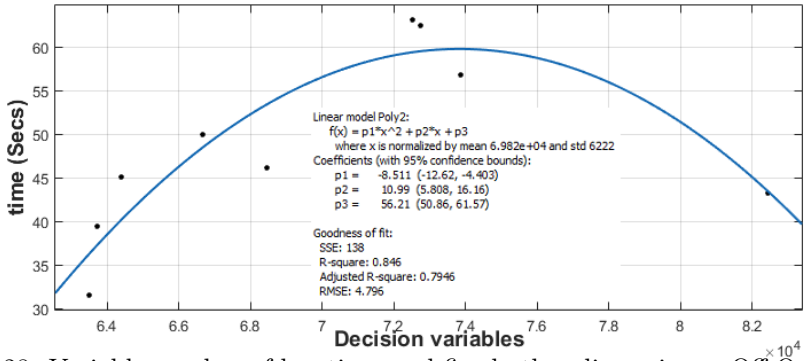


Figure 28: Variable number of locations and fixed other dimensions - Off-Opt-Swap.

$O(\Delta^2+2\times\Delta^2\times A\times T+2\times A\times L\times T\times\Delta+2\times A\times\Delta\times T+\Delta^2\times A^2\times T^2+2\times\Delta\times A^2\times L\times T^2+2\times\Delta\times A^2\times T^2+A^2\times L^2\times T^2+2\times A^2\times L\times T^2+A^2\times T^2)\cong O(A^2\times T^2\times(\Delta^2+L^2))$. For dimension $|L|$, the results are similar to the ones discussed in the previous paragraph.

In fact, as can be seen in Section 7.1, all the measurements of the exe-

cution times have revealed a quadratic time complexity, which is consistent with the above analysis. In the next section, the main results, observations and conclusions are summarized.

7.8. Summary of Key Results

To summarize, in our experimental evaluation we observe that when all EVs carry the same battery which is large enough for the longest trips, the greedy algorithm with battery swap in combination with the max flow provides the optimal solution. At the same time, the variation with battery charge is at 97% of the optimal without the local search, and at 98.5% when local search is used. Moreover, in case where some EVs do not have a large enough battery to execute some of the longest trips, we observe that the greedy algorithm does not provide the optimal solution anymore and that the incremental-MIP is the correct choice as it generates solution 0.5% better than the greedy and with lower execution time, while the optimal algorithm is still the best but scales up to medium sized problems only. Moreover, we show that when the objective is the maximization of EV utilization, the utilization increases by 3.6% and the number of completed tasks reduces by 1.5% on average. In addition, we proposed a battery swap minimization algorithm, which is applied a-posteriori to an already existing task execution schedule and minimizes the necessary battery swaps, thus also minimizing cost. Finally, we also proposed an online algorithm which takes into consideration the uncertainty in future demand and verified that it has good performance.

Generally speaking, the performance of these algorithms depends on the number of EVs, the number of pick-up and drop-off locations, and as a consequence of this, the number of all possible trips.

8. Conclusions and Future Work

In this paper, we have studied the problem of scheduling a set of shared EVs in an MoD scheme and initially characterize the problem as a max flow one to determine the set of feasible tasks, given the available EVs at each location. Later, we proposed an MIP formulation to solve it optimally. Given that this solution scales only up to medium sized problems, we also proposed two non-optimal algorithms. The first one is an incremental MIP solution, which calls the MIP incrementally for each EV. The second is a greedy scheduling algorithm which scales up to thousands of tasks and EVs.

Finally, we developed a tabu search-based local search technique to improve the solution quality of the non-optimal algorithms. In all cases, either battery swap or battery charge is used to cope with EVs' limited range. In our experiments, we observe that when all EVs carry the same battery which is large enough for the longest trips, the greedy algorithm with battery swap in combination with the max flow provides the optimal solution. At the same time, the variation with battery charge is close to the optimal and is further slightly improved when local search is used. When some EVs do not have a large enough battery to execute some of the longest trips, the incremental MIP generates solutions slightly better than the greedy, while the optimal algorithm is the best but scales up to medium sized problems only. Moreover, the online algorithm is shown to be on average at least 90% of the optimal. When taken together, our algorithms and results establish the first benchmarks for the study of EVs in MoD schemes and can be used as benchmarks for future research.

Future work will look into possible relocation mechanisms for the EVs, in order to further improve the task completion rates. Such mechanisms could be based on crowd-sourcing techniques where available drivers from the crowd could drive EVs across locations (as discussed in Section 2). In a similar vein, car pooling where multiple customers will ride the same EV could also be considered. In addition to these, we aim to investigate market-based techniques to incentivise customers to execute trips which improve the numbers of tasks executed. Moreover, given all the vagaries of an online implementation have not been factored in, we aim to further expand the online algorithm. Dealing with cancellations, predicting delays in travel, etc are issues that should be dealt with in a real-world deployment. Thus, we consider to improve and extend the online algorithm in a number of ways: 1) We aim to introduce probability distributions regarding future demand, or use machine learning techniques in order to have a better forecast of future demand and increase the number of executed tasks. 2) We aim to investigate the use of dynamic pricing techniques in order to balance the requests across the stations [39]. 3) We aim to introduce relocation drivers in order to cope with the case where bookings are being canceled. In this way the schedule of future tasks would remain feasible. 4) We aim to introduce the notion of "acceptable risk" regarding a schedule for future tasks by using chance constraints (tuning the constraints in a human-understandable way is then another research question).

9. Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments that boosted the efficiency of the proposed methods and greatly improved this paper.

10. References

[1] Agatz, N., Erera, A., Savelsbergh, M., Wang, X., 2012. Optimization for dynamic ride-sharing: A review. European Journal of Operational Research 223 (2), 295 – 303.

[2] Anbil, R., Tanga, R., Johnson, E. L., Jan. 1992. A global approach to crew-pairing optimization. IBM Syst. J. 31 (1), 71–78.

[3] Andersson, A., Tenhunen, M., Ygge, F., 2000. Integer programming for combinatorial auction winner determination. In: MultiAgent Systems, 2000. Proceedings. Fourth International Conference on. pp. 39–46.

[4] Barth, M., Shaheen, S., 2002. Shared-use vehicle systems: Framework for classifying carsharing, station cars, and combined approaches. Transportation Research Record: Journal of the Transportation Research Board (1791), 105–112.

[5] Bayen, A. M., Tomlin, C. J., Ye, Y., Zhang, J., Dec 2003. Milp formulation and polynomial time algorithm for an aircraft scheduling problem. In: 42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475). Vol. 5. pp. 5003–5010 Vol.5.

[6] Bent, R., Van Hentenryck, P., 2007. Waiting and relocation strategies in online stochastic vehicle routing. In: IJCAI. pp. 1816–1821.

[7] Bistaffa, F., Farinelli, A., Chalkiadakis, G., Ramchurn, S. D., 2015. Recommending fair payments for large-scale social ridesharing. In: Proceedings of the 9th ACM Conference on Recommender Systems. RecSys '15. ACM, New York, NY, USA, pp. 139–146.
URL http://doi.acm.org/10.1145/2792838.2800177

[8] Brafman, R. I., Domshlak, C., 2013. On the complexity of planning for agent teams and its implications for single agent planning. Artificial Intelligence 198, 52 – 71.

[9] Budai, G., Maróti, G., Dekker, R., Huisman, D., Kroon, L., 2010. Rescheduling in passenger railways: the rolling stock rebalancing problem. Journal of Scheduling 13 (3), 281–297.
URL http://dx.doi.org/10.1007/s10951-009-0133-9

[10] Burns, L. D., 2013. Sustainable mobility: a vision of our transport future. Nature 497 (7448), 181–182.

[11] Camacho, E. F., Alba, C. B., 2013. Model predictive control. Springer Science & Business Media.

[12] Carpenter, T., Keshav, S., Wong, J., June 2014. Sizing finite-population vehicle pools. IEEE Transactions on Intelligent Transportation Systems 15 (3), 1134–1144.

[13] Cepolina, E. M., Farina, A., 2012. A new shared vehicle system for urban areas. Transportation Research Part C: Emerging Technologies 21 (1), 230 – 243.

[14] Chandran, B., Raghavan, S., 2008. Modeling and Solving the Capacitated Vehicle Routing Problem on Trees. Springer US, Boston, MA, pp. 239–261.
URL http://dx.doi.org/10.1007/978-0-387-77778-8$_1$1

[15] Dantzig, G. B., Ramser, J. H., Oct. 1959. The truck dispatching problem. Manage. Sci. 6 (1), 80–91.
URL http://dx.doi.org/10.1287/mnsc.6.1.80

[16] De Weerdt, M. M., Gerding, E. H., Stein, S., Robu, V., Jennings, N. R., 2013. Intention-aware routing to minimise delays at electric vehicle charging stations. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. IJCAI'13. AAAI Press, pp. 83–89.

[17] Densing, M., Turton, H., Bäuml, G., 2012. Conditions for the successful deployment of electric vehicles–a global energy system perspective. Energy 47 (1), 137–149.

[18] Doppers, F.-A., Iwanowski, S., 2012. E-mobility fleet management using ant algorithms. Procedia - Social and Behavioral Sciences 54, 1058

– 1067, proceedings of {EWGT2012} - 15th Meeting of the {EURO} Working Group on Transportation, September 2012, Paris.

[19] Dorigo, M., Gambardella, L. M., Apr 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1 (1), 53–66.

[20] Floudas, C. A., Lin, X., 2005. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. Annals of Operations Research 139 (1), 131–162.

[21] Funke, S., Nusser, A., Storandt, S., 2015. Placement of loading stations for electric vehicles: No detours necessary! Journal of Artificial Intelligence Research 53, 633–658.

[22] George, D. K., Xia, C. H., 2011. Fleet-sizing and service availability for a vehicle rental system via closed queueing networks. European Journal of Operational Research 211 (1), 198 – 207.

[23] Glover, F., Taillard, E., Taillard, E., Mar 1993. A user's guide to tabu search. Annals of Operations Research 41 (1), 1–28.
URL https://doi.org/10.1007/BF02078647

[24] Hillier Frederick, S., Lieberman Gerald, J., 2010. Introduction to operations research.

[25] Hutter, F., Xu, L., Hoos, H. H., Leyton-Brown, K., 2014. Algorithm runtime prediction: Methods and evaluation. Artificial Intelligence 206, 79 – 111.

[26] IBM-Knowledge-Center, 2015. Branch and cut in cplex. Tech. rep., CPLEX Reference Manual.
URL http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.6.2/ilog.odms.cplex.help/refcppcplex/html/branch.html

[27] Lawler, E. L., 1963. The quadratic assignment problem. Management science 9 (4), 586–599.

[28] Lomnicki, Z. A., 1965. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. Journal of the Operational Research Society 16 (1), 89–100.
URL http://dx.doi.org/10.1057/jors.1965.7

[29] Ma, Z., Callaway, D., Hiskens, I., 2010. Decentralized charging control for large populations of plug-in electric vehicles. In: Decision and Control (CDC), 2010 49th IEEE Conference on. pp. 206–212.

[30] Mitchel, W. J., Borroni-Bird, C. E., Burns, L. D., 2010. Reinventing the automobile: Personal urban mobility for the 21st century. MIT Press.

[31] Osman, I. H., 1993. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of operations research 41 (4), 421–451.

[32] Pardalos, P. M., Pitsoulis, L., Mavridou, T., Resende, M. G., 1995. Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing, tabu search and grasp. In: International Workshop on Parallel Algorithms for Irregularly Structured Problems. Springer, pp. 317–331.

[33] Pavone, M., Smith, S. L., Frazzoli, E., Rus, D., 2012. Robotic load balancing for mobility-on-demand systems. The International Journal of Robotics Research 31 (7), 839–854.

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

[35] Ramchurn, S. D., Polukarov, M., Farinelli, A., Truong, C., Jennings, N. R., 2010. Coalition formation with spatial and temporal constraints. In: AAMAS. pp. 1181–1188.

[36] Raviv, T., Tzur, M., Forma, I. A., 2013. Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal on Transportation and Logistics 2 (3), 187–229.
URL http://dx.doi.org/10.1007/s13676-012-0017-6

[37] Rigas, E., Ramchurn, S., Bassiliades, N., Aug 2015. Managing electric vehicles in the smart grid using artificial intelligence: A survey. Intelligent Transportation Systems, IEEE Transactions on 16 (4), 1619–1635.

[38] Rigas, E. S., Ramchurn, S. D., Bassiliades, N., Sept 2015. Algorithms for electric vehicle scheduling in mobility-on-demand schemes. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems. pp. 1339–1344.

[39] Rigas, E. S., Ramchurn, S. D., Bassiliades, N., Koutitas, G., 2013. Congestion management for urban ev charging systems. In: Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on. pp. 121–126.

[40] Robu, V., Gerding, E. H., Stein, S., Parkes, D. C., Rogers, A., Jennings, N. R., 2013. An online mechanism for multi-unit demand and its application to plug-in hybrid electric vehicle charging. Journal of Artificial Intelligence Research 48, 175–230.

[41] Sachenbacher, M., Leucker, M., Artmeier, A., Haselmayr, J., 2011. Efficient energy-optimal routing for electric vehicles. In: AAAI.

[42] Sandholm, T., Suri, S., Gilpin, A., Levine, D., 2002. Winner determination in combinatorial auction generalizations. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1. AAMAS '02. ACM, New York, NY, USA, pp. 69–76.

[43] Schrank, D., B., E., Lomax, T., 2012. Tti 2012 urban mobility report. Tech. rep.

[44] Serfozo, R., 1999. Introduction to Stochastic Networks. Springer-Verlag New York.

[45] Smith, S., Pavone, M., Schwager, M., Frazzoli, E., Rus, D., June 2013. Rebalancing the rebalancers: optimally routing vehicles and drivers in mobility-on-demand systems. In: American Control Conference (ACC), 2013. pp. 2362–2367.

[46] Spieser, K., Treleaven, K., Zhang, R., Frazzoli, E., Morton, D., Pavone, M., 2014. Toward a systematic approach to the design and evaluation

of automated mobility-on-demand systems: A case study in singapore. In: Road Vehicle Automation. Springer, pp. 229–245.

[47] Storandt, S., 2012. Quick and energy-efficient routes: computing constrained shortest paths for electric vehicles. In: Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science. IWCTS '12. pp. 20–25.

[48] Storandt, S., Funke, S., 2012. Cruising with a battery-powered vehicle and not getting stranded. In: 26th Conf. on Artificial Intelligence (AAAI).

[49] Storandt, S., Funke, S., 2013. Enabling e-mobility: Facility location for battery loading stations. In: 27th Conf. on Artificial Intelligence (AAAI).

[50] Sundstrom, O., Binding, C., 2010. Planning electric-drive vehicle charging under constrained grid conditions. In: Power System Technology (POWERCON), 2010 International Conference on. pp. 1–6.

[51] Talbot, F. B., Patterson, J. H., 1978. An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. Management Science 24 (11), 1163–1174.

[52] Tomic, J., Kempton, W., 2007. Using fleets of electric-drive vehicles for grid support. Journal of Power Sources 168 (2), 459 – 468.

[53] Valogianni, K., Ketter, W., Collins, J., Zhdanov, D., 2014. Effective management of electric vehicle storage using smart charging. In: AAAI. pp. 472–478.

[54] Vandael, S., Boucké, N., Holvoet, T., De Craemer, K., Deconinck, G., 2011. Decentralized coordination of plug-in hybrid vehicles for imbalance reduction in a smart grid. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 803–810.

[55] Waserhole, A., Jost, V., 2014. Pricing in vehicle sharing systems: optimization in queuing networks with product forms. EURO Journal on Transportation and Logistics, 1–28.

[56] Worley, O., Klabjan, D., Sept 2011. Optimization of battery charging and purchasing at electric vehicle battery swap stations. In: 2011 IEEE Vehicle Power and Propulsion Conference. pp. 1–4.

[57] Zhang, W., 1996. Branch-and-bound search algorithms and their computational complexity. Tech. rep., DTIC Document.