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

Emmanouil S. Rigas*, Sarvapali D. Ramchurn†, Nick Bassiliades*

*Department of Informatics, Aristotle University of Thessaloniki, 54124, Thessaloniki, Greece
{erigas, nbassili}@csd.auth.gr
†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. Each point in the MoD scheme is equipped with a battery swap facility that helps cope with the EVs' limited range, while the goal of the system is to maximise the number of customers that are serviced. Thus, we first model and solve this problem optimally using Mixed-Integer Programming (MIP) techniques and show that the solution scales up to medium sized problems. Given this, we develop a greedy approach that is shown to output solutions that are close to the optimal and can scale to thousands of consumer requests and EVs. Both algorithms are evaluated in a setting using data of actual locations of shared vehicle pick-up and drop-off stations in Washington DC, USA and the greedy algorithm is shown to be on average 90% of the optimal in terms of average task completion.**

I. Introduction

In a world where over 60% of the total population will be leaving in, or around cities, the current personal transportation model is not viable as it is based almost entirely on privately owned internal combustion engined vehicles. These vehicles cause high pollution (e.g., air and sound), and suffer from low utilization rates [1]. One of the key elements of the vision of future Smart Cities is the development of Mobility-on-Demand (MoD) systems, especially ones using fleets of compact-sized Electric Vehicles (EVs) [2]. Such vehicles emit no tailpipe pollutants and, once powered by electricity (partially) produced from renewable sources, they can play an important role towards the transition to a new and sustainable transportation era.

A number of MoD schemes, such as ZipCar[1], or CarShare[2] have recently 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 will be 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 this context, Pavone et al. have developed mathematical programming-based rebalancing mechanisms for deciding on the relocation of vehicles to restore imbalances across a MoD network, either using robotic autonomous driving vehicles [3], or human drivers [4], while Smith et al. [5] use mathematical programming to optimally route such rebalancing drivers. Moreover, Carpenter et al. [6] propose solutions for the optimal sizing of shared vehicle pools. However, in all these works internal combustion engined vehicles are assumed and hence do not account for the limited range of EVs and how to balance the load (i.e., number of pending requests at specific nodes) across the network (i.e., by choosing which trips to execute) while serving the maximum number of users.

Against this background, we model the MoD scheme for EVs and develop novel algorithms to solve the problem of scheduling trips for MoD consumers in order to maximise the number of requests serviced while coping with the limited range of EVs. Crucially, we show how the schedule of trips in the MoD network is a highly combinatorial problem, for which an optimal solution scales only up to medium sized problems. Thus, to cope with large problems, we also develop a greedy solution. Specifically, this paper advances the state of the art as follows: 1) We propose an optimal MIP formulation of the problem of scheduling EVs in an MoD scheme that maximizes the number of completed tasks (i.e., trips). 2) We show that the optimal solution scales up to a few hundred tasks and we propose a greedy algorithm which achieves near optimal performance, has low execution time, and scales to thousands of tasks. 3)We propose a battery swap optimization algorithm, which, given an EV's travelling schedule, minimizes the number of necessary battery swaps. 4) We empirically show, using real data of shared vehicle stations in Washington, DC that the greedy algorithm is on average 90% of the optimal in terms of number of tasks completed, while it scales to thousands of EVs and tasks.

The rest of the paper is structured as follows. Section II presents the model of a typical MoD scheme, while Section III presents the MIP formulation of the problem and Section IV the greedy algorithm. Section V describes our battery swap optimization algorithm, Section VI describes our empirical evaluation and finally, Section VII concludes.

[1]http://www.zipcar.com/.
[2]http://www.enterprisecarshare.com/.

II. 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 intentions have been collected by the MoD company (we assume a single MoD company to exist), it applies an optimization algorithm and schedules the available EVs to drive across a set of locations. In choosing the tasks it will execute, the MoD company aims to maximize the number of customers that will be serviced. We assume that EVs have a limited driving range which requires them to have their battery swapped [7] at the stops that form part of the MoD scheme.[3] Battery swap is an efficient alternative to battery recharging in cases of shared EVs, such as the one we study here, as the vehicle fleet has a certain size and uses a specific battery type (we assume all EVs are from the same model), while batteries' ownership is not an issue as a single stakeholder exists.

In more detail, we denote a set of EVs $a_j \in A$ and a set of locations $l_k \in L$ which are pick-up and drop-off stations, where each $l_k \in L$ 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 $\delta_i \in \Delta$ where each task is denoted by a tuple $< k_i^{start}, k_i^{end}, t_i^{start}, \tau_i, b_i >$. k_i^{start} and k_i^{end} are the start and end locations of the task, t_i is the starting time point of the task, while 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. Note here, that one-way rental is assumed, and therefore, start and end locations of a task are always different. Henceforth, index j stands for EVs, k for locations, t for time points and i for tasks.

Each EV a_j has a current location at time point t, denoted as $k_{j,t}$, and this location changes only each time a_j executes one task (i.e., no EV can change its location without executing a task). Here, we assume that at time point $t = 0$ all EVs are at their initial locations $k_{j,t=0}^{initial} \in L$, and that their operation starts at time point $t \geq 1$. Moreover, each a_j has a current battery level $b_{t,j} \in N$, a consumption rate con_j and therefore, a current driving range in terms of time $\tau_{j,t} = [b_{t,j}/con_j] \in N$. Now, for a task δ_i to be accomplished, at least one EV a_j must be at location k_i^{start} at time point t_i. We also define binary variable $prk_{t,j,k} \in \{0,1\}$ to capture the location where each EV is parked at each time point, and binary variable $\epsilon_{j,i,t} \in \{0,1\}$ denoting whether EV a_j is executing task δ_i at time point t. Note, that at any t, each EV should either be parked at exactly one location, or travelling between exactly one pair of locations.

Once the model is defined, in the next section the MIP formulation of the problem is presented.

[3]In a battery swap station the battery is not recharged but instead it is replaced by an already charged one. Such stations can reduce battery reloading time, but they come with a high cost [8].

III. MIP Formulation

Once the problem is defined and formulated, here we present an optimal solution based on Mixed Integer Programming (MIP) (solved using IBM ILOG CPLEX 12.5). Specifically, the aim is to maximize the number of tasks that are completed (Equation 1). To this end, we define three binary decision variables: 1) $\lambda_{\delta_i} \in \{0,1\}$ denoting whether a task δ_i is accomplished or not, 2) $\epsilon_{j,i,t} \in \{0,1\}$ denoting whether EV a_j is executing task δ_i at time t or not, and 3) $prk_{j,t,k} \in \{0,1\}$ denoting whether a_j is parked at time point t at location l_k or not. Moreover, a set of constraints is used:

Objective function:

$$max \sum_{\delta_i \in \Delta} (\lambda_{\delta_i}) \tag{1}$$

Subject to:

- *Completion constraints:*

$$\sum_{a_j \in A} \sum_{t_i^{start} \leq t < t_i^{end}} \epsilon_{j,i,t} = \tau_i \times \lambda_{\delta_i}, \forall \delta_i \tag{2}$$

$$\sum_{a_j \in A} \sum_{t_i^{start} > t \geq t_i^{end}} \epsilon_{j,i,t} = 0, \forall \delta_i \tag{3}$$

$$\epsilon_{j,i,t+1} = \epsilon_{j,i,t} \forall a, \forall \delta_i, \forall t : t_i^{start} \leq t < t_i^{end} - 1 \tag{4}$$

$$\sum_{t_i^{start} \leq t < t_i^{end}} \epsilon_{j,i,t} \leq \tau_{j,t_i^{start}}, \forall \delta_i, \forall a_j \tag{5}$$

- *Temporal, spatial, and routing constraints:*

$$\sum_{l_k \in L} prk_{j,t,k} = 1 - \sum_{\delta_i \in \Delta} \epsilon_{j,i,t}, \forall a_j, \forall t \tag{6}$$

$$2 \times \sum_{\delta_i \in \Delta} \epsilon_{j,i,t_i^{start}} = \sum_{l_k \in L} \sum_{t \in T} |prk_{j,t+1,k} - prk_{j,t,k}|, \forall a_j \tag{7}$$

$$\sum_{a_j \in A} prk_{j,t,k} = prk_{j,t-1,k} + \sum_{\Delta^{st}(t,k)} \lambda_{\delta_i} - \sum_{\Delta^{end}(t,k)} \lambda_{\delta_i} \forall t, \forall l_k \tag{8}$$

$$prk_{j,t_i^{start},k_i^{start}} - 1 \geq \epsilon_{j,i,t_i^{start}}, \forall \delta_i, \forall a_j \tag{9}$$

$$prk_{j,t_i^{end},k_i^{end}} \geq \epsilon_{j,i,t_i^{end}}, \forall \delta_i, \forall a_j \tag{10}$$

$$\sum_{a_j \in A} (prk_{j,t,k}) \leq c_l, \forall l_k, \forall t \tag{11}$$

$$prk_{j,t=0,k} = k_a^{initial}, \forall a_j, l_k \tag{12}$$

The *completion* constraints ensure the proper execution of tasks. In more detail, for each executed task, the time travelled must be equal to the duration of the trip concerned (Equation 2), while, at the same time no travelling must take place when a task is not executed (Equation 3). Moreover, each task is executed by only one EV at a time (Equation 4 together with Equation 7). Now, for an EV to execute a task, its full range, calculated based on the battery level at the starting time of the task, must not be violated (Equation 5). Note, that 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. Also note, that the number of battery swaps is minimized a posteriori (see Section V).

The *temporal, spatial and routing* constraints ensure the proper management of the EVs. In more detail, Equation 6 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 4) that at each time point, each EV executes at most one task. Moreover, Equation 7 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. On top of this, for every location, the total number of EVs changes only when EVs depart or arrive to execute, or after executing tasks (Equation 8). Note, that although this constraint is covered by Equation 7, it is added to the formulation as it significantly speeds up the execution time. For example, for a setting with 8 locations, 15 EVs and 70 tasks, constraint 7 reduces the average execution time drops from 450 seconds to 220 seconds. Note, that $\Delta^{st}(t,k) = \{d_i \in \Delta\colon t_i^{start} = t, k_i^{start} = k\}$ and $\Delta^{end}(t,k) = \{d_i \in \Delta\colon t_i^{end} = t, k_i^{end} = k\}$. 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 9), and similarly, whenever a task has been executed, the EV that has executed this task must be at the task's ending location the time point the task ends (Equation 10). Moreover, at every time point, the maximum capacity of each location must not be violated (Equation 11). Finally, at time point $t = 0$, all EVs must be at their initial locations (Equation 12), which also means that no tasks are executed at $t = 0$.

The solution presented here calculates the optimal schedule for the EVs. However it comes with a high computational and time cost (see Section VI-A) and therefore, it is usable for small and medium sized problems. For this reason, an algorithm that can calculate solutions close to the optimal, but with a low time and computational complexity, is essential. Given that greedy algorithms based on heuristic search have been proved to be effective in similar highly combinatorial problems [9], in the next section, such a greedy algorithm is presented.

IV. Greedy Scheduling Formulation

The intuition of the greedy scheduling algorithm presented in this section is the following. 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 the case of the MIP formulation, CPLEX finds the optimal schedule for EVs' travelling to maximize task execution. However, in the case of the greedy algorithm exhaustive search is not an option as this would dramatically increase time and computational complexity. For this reason we apply a one-step look ahead heuristic search mechanism which, as we show later, achieves close to the optimal performance.

In more detail, for each t and for each l_k, the number of the available EVs, as well as the number of the remaining tasks to be executed are taken into consideration. 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, optimal selection of the tasks to be executed must be achieved. As mentioned before, a one-step look ahead technique is used. In more detail, at each time point, tasks remaining to be executed from each l_k are sorted (within a set O_k) based on the number of the remaining tasks to be executed at their destination divided by the number of EVs at the destination plus one (i.e., the one that will go there after executing the task). Given that the aim of this algorithm is to maximize the number of completed tasks, selecting to execute tasks that lead EVs to destinations where the probability of executing more tasks is higher sounds like the best choice. Therefore, a task is executed only if it belongs to the top $cutoff\%$ of O_k ($cutoff\%$ is provided to the scheduling algorithm by the user).

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

A. Pre-processing

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

B. Task Execution

After each time an EV is assigned to a task the following steps are executed: a) EV a_j is set to be working on task δ_i, by changing the value of variable $\epsilon_{j,i,t}$ from 0 to 1 for the duration of the trip. b) The parked location of EV a_j is updated based on the end location of the task and the arrival time. c) The total number of tasks completed is increased by one, and d) the total number of EVs parked at the start and end location of the trip are updated for the correct time points. Note, that following the same modelling of the problem as in the MIP formulation

(Equation 10), once an EV arrives at a destination, it stays there for at least one time point. This time point is used for the necessary battery swap. In the next section, the main scheduling algorithm is presented.

C. The EV-MoD Scheduling Algorithm

Coming to the greedy scheduling algorithm (Algorithm 1), for each time point t and for every pick-up and drop-off location l_k, the following steps are executed:

1) For each l_k, pairs of task id and number of tasks remaining to be executed at the destination of each task, divided by the number of EVs at the destination plus one are assigned to O_k. Later, O_k is sorted in descending order of the number of tasks remaining to be executed at the destination. We then keep the top $cutoff\%$ values of O_k (lines 1 -6).
2) For all $\delta_i \in CT_{k,t}$ awaiting to be executed at the current time point, the following steps are executed repetitively for each task:
3) If the number of the EVs parked at the current location is greater than or equal to the tasks remaining to be executed, then all tasks are set to be executed sequentially (lines 7-11): a) If the execution of the task, does not lead to a violation of the maximum capacity of the destination location (waiting queues are assumed not to exist), then: b) once an EV parked at the current location having enough range to complete the task is found, it is assigned to the task and the task is executed. If no such EV is found, then the task is not completed.
4) If the number of the EVs parked at the current location is less than the tasks remaining to be executed, then (lines 12-19): a) If the execution of the task, does not lead to a violation of the maximum capacity of the destination location, then: b) The current task is executed if it belongs to the O_k set. c) Once an EV that is parked at the current location and has enough range to complete the task has been found, it is assigned to the task is executed. If no such EV is found, then the task is not executed.

Once the execution of algorithm 1 is finished, the travelling schedule of all EVs, as well as the total number of completed tasks is returned as an output. In the next section, the analysis of the average case time complexity of the greedy algorithm is presented.

D. Complexity Analysis

While the algorithm is greedy, some time consuming calculations such as the sorting of O_k still exist and therefore, the analysis of the complexity of it is important.

Theorem 1. (Complexity class of Algorithm 1)

The complexity of Algorithm 1 is $O(|\Delta| \times log\frac{|\Delta|}{|T|\times|L|})$.

Proof:

$$Cost = |T| \times |L| \times (|\Delta + |\Delta| \times log|\Delta| + |\Delta| + |\Delta|) \quad (13)$$

Given that the sorting $(|\Delta| \times log|\Delta|)$ is the most time consuming calculation, Equation 13 becomes:

$$Cost = O(|T| \times |L| \times (|\Delta| \times log|\Delta|)) \quad (14)$$

In the average case where tasks are equally distributed across locations and time points, equation 14 becomes:

$$Cost = O(|T| \times |L| \times \frac{|\Delta|}{|T| \times |L|} \times log(\frac{|\Delta|}{|T| \times |L|})) \quad (15)$$

which is equal to:

$$Cost = O(|\Delta| \times log(\frac{|\Delta|}{|T| \times |L|})) \quad (16)$$

Therefore, the average case complexity of Algorithm 1 is $O(|\Delta| \times log(\frac{|\Delta|}{|T|\times|L|})$. This result is also verified experimentally (Section VI-A).

■

In the next section, the battery swap optimization algorithm is presented.

Algorithm 1 EVs Scheduling Algorithm.

```
Require: $\Delta$ and $A$ and $L$ and $T$ and $\forall d_i \in \Delta$, $\tau_i$, and $\forall a_j \in A$,
    $k_j^{initial}$, $\tau_j^{max}$, and $\forall l_k \in L$, $c_k$.
 1: for all ($t \in T$ and $l_k \in L$) do
 2:     for all ($\delta_i \in CT_{k,t}$) do
 3: {Add all pairs of task id and number of tasks remaining to
    be executed at the destination divided by the number of
    EVs +1 to $O_k$.}
 4:         $O_k \leftarrow O_k \cup \{\delta_i, R_{t_i^{end},k_i^{end}}/(evs_{k_i^{end},t} + 1)\}$
 5:     end for
 6:     Sort $O_k$ in descending order of the number of tasks at
    the destination, and keep $cutoff\%$ values from $O_k$.
        {If available EVs at start, $\geq$ total task number.}
 7:     if ($evs_{t,k_i^{start}} > R_{t_i^{start},k_i^{end}}$ ) then
 8:         for all ($\delta_i \in CT_{k,t}$) do
 9:             If $evs_{t+\tau_i,k_i^{end}} < c_{k_i^{end}}$, search for an $a_j$ to
    execute $\delta_i$: $prk_{j,t,k} = 1$ AND $\tau_{j,t_i^{start}} \geq \tau_i$
10:             Once an EV is found, execute task.
11:         end for
12: {If available EVs at start < total task number.}
13:     else
14:         for all ($\delta_i \in O_k$) do
15:             If $evs_{t+\tau_i,k_i^{end}} < c_{k_i^{end}}$, Search for an $a_j$ to
    execute $\delta_i$: $prk_{j,t,k} = 1$ AND $\tau_{j,t_i^{start}} \geq \tau_i$
16:             Once an EV is found, execute task.
17:         end for
18:     end if
19: end for
20: Return $\epsilon$ , $prk$ , $taskSum$
```

V. Battery Swap Optimization

Both the MIP and the greedy algorithms calculate the schedule of each EV assuming that after every stop, a fully charged battery is swapped into it. In other words the longest single trip that an EV can travel is constrained by its maximum battery capacity. However, such a battery swap is sometimes redundant. For this reason, here we present a simple heuristic algorithm (Algorithm 2) which takes as input the EV's travelling schedule and minimizes the number of battery swaps.

In more detail, for every EV a_j, all tasks δ_i that are executed by a_j are assigned to set Ω_j, and are later sorted in ascending order of their start execution time (lines 1,2). Once all tasks have been assigned to sets, for every $\delta_i \in \Omega_j$ the execution time of i is added to variable *sum* until this variable becomes greater than or equal to the maximum range of a_j multiplied with 50% (i.e., the battery is discharged only up to the 50% of its full capacity to prolong its life time[4]). When this happens, the final task is not executed before a battery swap, and for this reason it is assigned to set Ω'_j and the *sum* is initialized again to zero. In the end of this procedure, Ω'_j contains all tasks before the execution of which a battery swap must take place (lines 3-9). In the next section, a detailed evaluation of both scheduling algorithms, as well as of the battery swap optimization algorithm is presented.

Algorithm 2 EVs Battery Swap Optimization.

```
Require: $\epsilon$.
 1: $\forall\ \delta_i \in \Delta$ and $a_j \in A$, if $\epsilon_{j,i,t} = 1$ then $\Omega_j \leftarrow \Omega_j + \delta_i$
 2: $\forall a_j \in A$: Sort $\Omega_j$ according to $t_i^{start}$ in ascending order.
 3: for all ($a_j \in A$) do
 4:     $sum = 0$
 5:     for all ($\delta_i \in \Omega_j$) do
 6:         $sum = sum + \tau_i$
 7:         If $sum \geq \tau_j^{max} \times 0.5$: $\Omega'_j \leftarrow \Omega'_j + \delta_i$ and $sum = 0$
 8:     end for
 9: end for
10: Return $\Omega'$.
```

VI. 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 data on the locations of pick-up and drop-off points owned by ZipCar in Washington DC, USA which are available as open linked data,[5] while the distance and duration of all trips (combinations of all locations acting as start or end points for a trip) were calculated using Google maps. Note, that Washington DC is one of the cities with the highest traffic congestion in the USA[6] [10] and 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 three main parts: EXP1: The execution time and the scalability of both the optimal and the greedy algorithms, EXP2: the performance of the greedy algorithm against the optimal in terms of the average number of completed tasks, EXP3: the sensitivity analysis of the greedy algorithm and, EXP4: the efficiency of the battery swap optimization algorithm.

For EXP1, EXP2 and EXP4 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) $cutoff = 85\%$, 3) 15 EVs exist, and 4) tasks can be formulated based on one of 56 possible trips. All tasks were categorized into three groups (i.e., from suburbs to the city center, around the city center, and from the city center to suburbs), where these groups were selected based on realistic data of the expected flow of traffic within the city over the day. Start times of tasks from each group were drown from a uniform distribution.

A. EXP1: Execution Time and Scalability

Execution time and scalability is a major factor in the usability of a given scheduling algorithm. For this reason, here, keeping all parameters but the number of tasks fixed, we measure the execution time of both the optimal and the greedy algorithms (using an Intel i7 CPU and 6 GB of RAM). Given these execution times (see Figures 1, 2), we could argue that for the optimal algorithm the execution time increases slightly over-linear, but with a rather high rate of growth, while for the greedy one, the execution time increases near-linearly with a rather low rate of growth. For example, the greedy algorithm can solve problems with 6000 tasks in under 5 seconds, while the optimal needs over 700 seconds for a setting with 200 tasks. Moreover, we have observed that for a setting with 500 tasks and 25 EVs, the average execution time of the optimal algorithm is over 4000 seconds. Thus, we argue the optimal algorithm to be usable for problems with up to a few hundred of tasks and a few tenths of EVs, while the greedy to be usable for problems with thousands of tasks and EVs. We next discuss how efficient the solutions it computes are.

B. EXP2: Completion of Tasks

Given the high execution times of the optimal algorithm, the question now comes to how close the greedy algorithm performs in terms of average task completion against the optimal. To answer this question, the greedy algorithm was evaluated against the optimal and given identical values as input. In Figures 3 and 4, a comparison between the two algorithms in terms of the number of completed tasks for various numbers of tasks and fixed number of EVs and locations is presented. Up to the number of 110 tasks, the greedy algorithm is on average at 90% of the optimal. In other words, the large difference in execution times leads to relatively small deficit in terms of performance. We next discuss the sensitivity analysis of the greedy algorithm.

C. EXP3: Sensitivity Analysis

Here we further evaluate our greedy algorithm where we vary both the number of locations and the number of EVs. In this experiment, we fix the number of tasks to be completed to 200, and vary the number of EVs as well as the number of the locations of the pick-up and drop-off points (here we use synthetic data for the locations). 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

[4]http://goo.gl/A0feT.
[5]http://opendata.dc.gov/datasets/.
[6]http://mobility.tamu.edu.

this is indeed the case (see Figure 5). However, in the case where the number of locations increases we note an opposite trend. It is interesting to see that when 30 EVs and 9 locations exist, the number of completed tasks is actually higher compared to the case where 80 EVs and 49 locations exist. We can potentially explain this as follows: 1) as the number of locations increases, the number of EVs located at each one 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. We next discuss the efficiency of the battery swap optimization algorithm.

D. 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 results as presented in VI-B. In doing so, we assume all EVs to carry the same battery type, which provides them with an average range of 80 kms, and an average trip length to be approximately 10 kms. Given these data, the battery swap minimization algorithm is shown to achieve a reduction of up to 80% in the number of necessary battery swaps.

Based on these observations we can conclude that despite the fact that the optimal algorithm is practical for settings with small and medium size, while the greedy algorithm has good scalability, and performs well. This performance 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. Also, an optimization of the battery swaps is needed to reduce redundant swaps.

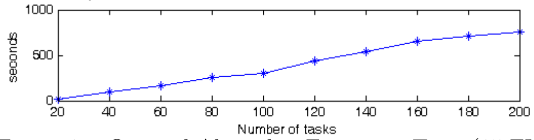

Figure 1. Optimal Algorithm Execution Time (15 EVs)

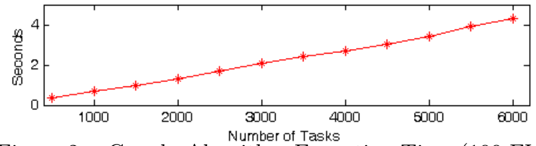

Figure 2. Greedy Algorithm Execution Time (100 EVs)

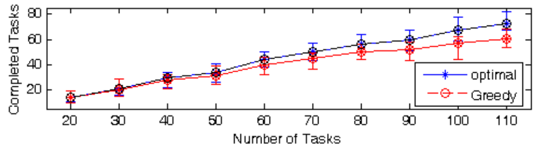

Figure 3. Optimal vs Greedy- Number of tasks completed

VII. Conclusions and Future Work

We have studied the problem of scheduling a set of shared EVs across a number of predefined pick-up and drop-off locations and we propose an MIP formulation for it, aiming to optimize (maximize) the number of completed tasks (trips undertaken across locations across a period of time). Moreover, given that this solution scales poorly, we also propose an offline close to optimal greedy algorithm, which can scale up to thousands of tasks and EVs while there is a tradeoff in terms of performance (completion of tasks). In both cases, in order to tackle with the limited range of EVs we use a simple battery swap scheme. Finally, we evaluate our algorithms in a setting partially using real data, and we observe that the greedy algorithm is on average 90% of the optimal. Future work will look into possible relocation mechanisms for the EVs, in order to further improve the task completion rates, as well as, the optimization of the initial location of the EVs.

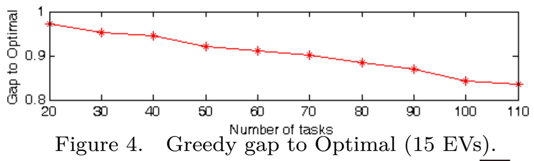

Figure 4. Greedy gap to Optimal (15 EVs).

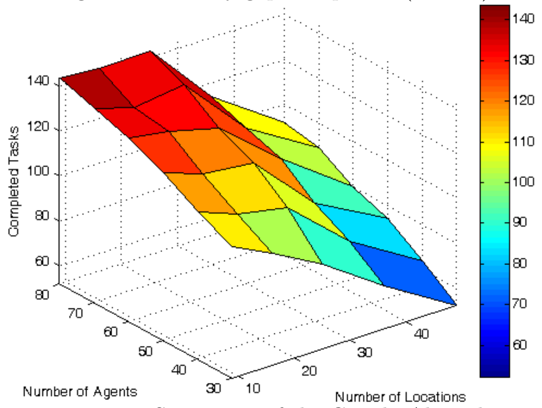

Figure 5. Sensitivity of the Greedy Algorithm

References

[1] J. Tomic and W. Kempton, "Using fleets of electric-drive vehicles for grid support," *Journal of Power Sources*, vol. 168, no. 2, pp. 459 – 468, 2007.

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

[3] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[4] M. Pavone, S. L. Smith, F. Emilio, and D. Rus, "Load balancing for mobility-on-demand systems," *Robotics: Science and Systems*, 2011.

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

[6] T. Carpenter, S. Keshav, and J. Wong, "Sizing finite-population vehicle pools," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, no. 99, pp. 1–11, 2014.

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

[8] I. Bayram, G. Michailidis, M. Devetsikiotis, F. Granelli, and S. Bhattacharya, "Smart vehicles in the smart grid: Challenges, trends, and application to the design of charging stations," ser. Power Electronics and Power Systems, 2012, pp. 133–145.

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

[10] D. Schrank, E. B., and T. Lomax, "Tti 2012 urban mobility report," Tech. Rep., 2012.