A Model-Based Online Mechanism with Pre-Commitment and its Application to Electric Vehicle Charging

Sebastian Stein, Enrico Gerding, Valentin Robu and Nicholas R. Jennings

Electronics and Computer Science, University of Southampton
Southampton, SO17 1AE, United Kingdom
{ss2,eg,vr2,nrj}@ecs.soton.ac.uk

ABSTRACT

We introduce a novel online mechanism that schedules the allocation of an expiring and continuously-produced resource to self-interested agents with private preferences. A key application of our mechanism is the charging of pure electric vehicles, where owners arrive dynamically over time, and each owner requires a minimum amount of charge by its departure to complete its next trip. To truthfully elicit the agents' preferences in this setting, we introduce the new concept of pre-commitment: Whenever an agent is selected, our mechanism pre-commits to charging the vehicle by its reported departure time, but maintains flexibility about *when* the charging takes place and at *what rate*. Furthermore, to make effective allocation decisions we use a model-based approach by modifying Consensus, a well-known online optimisation algorithm. We show that our pre-commitment mechanism with modified Consensus incentivises truthful reporting. Furthermore, through simulations based on real-world data, we show empirically that the average utility achieved by our mechanism is 93% or more of the offline optimal.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*

General Terms

Algorithms, Design, Economics

Keywords

electric vehicles, mechanism design, pricing

1. INTRODUCTION

Recent years have seen a proliferation of interest in electric vehicles (EVs), generally perceived as a key technology for achieving sustainable mass transportation with low carbon emissions [7]. While EVs are a promising technology, their widespread use is also expected to place considerable strains on existing electricity distribution networks. EVs typically require high charging rates, up to 3 times the maximum demand of a typical home. This means that if all vehicle owners plug in at peak times (typically in the early evening), the transformers involved in distributing electricity to local neighbourhoods may be overloaded by the additional demand [7]. To this end, the charging of vehicles needs to be scheduled in order to balance the load. However, different consumers may have different time constraints and willingness to pay, which the schedule needs to take into consideration. To address this challenge, in this paper, we apply and extend techniques from mechanism design and the stochastic optimisation literature to ensure incentive compatibility (i.e., to incentivise *strategic* agents to reveal their preferences truthfully), and produce effective schedules for EV charging.

Dealing with the limitation of local distribution networks has been discussed in a range of recent works. For example, [2] and [12] provide a thorough descriptive analysis of this problem, although they only discuss the problem at a high level and do not propose specific scheduling heuristics. Moreover, these and most other papers assume that information about the EVs is known and they do not consider the elicitation problem, where strategic agents may misreport their preferences if this is in their best interest.

Framed in more general terms, the problem involves the real-time scheduling of jobs released over time (in our case, electric vehicles that require a certain amount of charge by their departure) sharing a scarce resource (in our case, electricity that is limited by the maximum transformer capacity), and given uncertainty about future arrivals. Such problems are addressed in the important and growing field of stochastic optimisation [6]. However, it has been shown that solving these problems optimally is NP-hard (see [10] for an overview) and many heuristics have been developed. The approach in our paper is based on one of the most widely used such heuristics, the Consensus approach introduced in [1]. In their approach, a number of future scenarios are sampled, and then the scheduling is solved for each of these scenarios (which can be solved using an offline algorithm). Then, the decision whether to schedule a particular job (in our case, an EV) is made based on a Consensus vote between these scenarios. However, unlike our work, [1] only applies to settings with a single machine (corresponding to a setting where only a single EV can be charged at any time) and agents are assumed to be non-strategic.

A number of papers have considered scheduling with strategic agents. Specifically, [11] examines the scheduling of jobs on a single machine and proposes an incentive compatible mechanism for this setting. However, their work assumes a computational setting where the results of a job are released to the agents only on completion or by the agent's reported deadline. This approach cannot be used for electricity distribution, since electricity must be allocated instantly when available. More recently, [5] proposes a mechanism that deals with multi-dimensional, marginally decreasing valuations, showing how this setting naturally applies to the charging of *hybrid* EVs (where any shortfall in electricity can be supplemented by fuel). However, that approach does not readily extend to the case of pure EVs, which is better characterised by complemen-

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

tary preferences (since agents only receive value when the battery is sufficiently charged), and it assumes both discrete time and supply. Furthermore, all of these approaches are *model-free*, i.e., they do not consider possible future arrivals in the allocations. This can be especially inefficient in case of complementary preferences.

There are a few online mechanism design papers which also account for future arrivals [8, 3]. Like our paper, these are based on a version of the Consensus algorithm, but their approach results in allocations which are not necessarily *monotonic* (a necessary condition for incentive compatibility). As a result, they require an additional *ironing* procedure which cancels allocations that violate the monotonicity property. This approach has two drawbacks. First, ironing is computationally prohibitive and thus impractical for the EV domain, since it needs to consider all possible misreports in order to establish whether cancelling an allocation is required. Second, it causes inefficiencies in the allocation, since ironed resources are lost and cannot be allocated to any other agents. Furthermore, unlike our setting, both [8] and [3] assume discrete time and indivisible resources.

In this work, we address these limitations by proposing the first incentive-compatible, model-based mechanism for a real-time setting with a continuously divisible resource. This achieves incentive compatibility by ensuring monotonicity for all allocations, thus avoiding the need for expensive ironing. In more detail, we make the following contributions to the state of the art:

1) We introduce, for the first time, the notion of *pre-commitment* in online mechanism design. When the mechanism decides to pre-commit to an agent, it ensures that sufficient resources are reserved for the agent, but, importantly, retains flexibility over when and how the resource is allocated. This leaves considerable flexibility for accommodating other, potentially less patient agents in the future.

2) To ensure incentive compatibility, we furthermore modify the existing Consensus algorithm. Specifically, we introduce a *serialisation* of agents, ensuring that an unallocated agent cannot influence future pre-commitment decisions. Second, we ensure agents do not have an incentive to delay their arrival in the system by identifying *re-evaluation points*, at which scheduling and pre-commitment decisions may change, and by using *partly-fixed* schedules, whereby the schedule for committed agents is fixed until the next re-evaluation point.

3) We evaluate our mechanism using real-world data from the largest trial of EVs in the UK and show that the average performance obtained by our mechanism is 93% or more of an offline optimal, and that it significantly outperforms a mechanism that allocates electricity without knowledge of the agents' preferences.

The remainder of the paper is organised as follows. In Section 2, we describe our model. In Section 3, we solve the online scheduling problem for a setting with cooperative agents, while in Section 4 we consider how to ensure truthful reporting by strategic agents. Finally, an experimental evaluation is presented in Section 5.

2. MODEL

We consider a setting where a limited and expiring resource (e.g., electricity) is continuously being produced and must be allocated to consumers immediately or is otherwise lost. Formally, the *production rate* (e.g., in kW) of the resource at time $t \in \mathbb{R}$ is given by $s(t)$. We assume future supply is known. Furthermore, we model supply using a step function defined by a set of rates $S = \{s_0, s_1, \dots, s_m\}$ and transition points $T = \{t_1, t_2, \dots, t_m\}$, such that the supply rate transitions from s_{i-1} to s_i at time t_i.

We let $I = \{1, 2, \dots, n\}$ denote the set of potential consumers, henceforth called *agents*, who arrive over time. Each agent $i \in I$ is interested in a *required amount*, q_i, of the resource and has a *value*, v_i, for this amount. The agent has no additional value for receiving more, and has a value of 0 for getting less than q_i. Additionally, agents can only receive the resource during their *active interval*, $[a_i, d_i]$, where a_i and d_i are the agent's arrival and departure times. Finally, each agent has a *maximum consumption rate* r_i, such that, in a given time interval Δt, agent i can consume at most $r_i \cdot \Delta t$ of the resource. Note that agents are indifferent about when they receive the resource as long as it is within the active interval. Together these form agent i's *type*, $\theta_i = \langle q_i, v_i, r_i, a_i, d_i \rangle$, and we denote the type profile for agents in I by $\theta_I = \{\theta_1, \theta_2, \dots, \theta_n\}$. This generic model maps nicely into our EV charging scenario, where r_i is the maximum charging speed (in kW) and $[a_i, d_i]$ represents the time interval during which the car is at home and available for charging. Furthermore, this model can be readily applied to a wide range of other domains where expiring resources become available continuously over time, such as the allocation of computational resources in cloud settings, the allocation of bandwidth in networks, or even allocating human resources to projects.

Given this model, we are interested in designing a mechanism for finding a real-time schedule, ρ, where $\rho(i, t) \in \mathbb{R}$ determines the *consumption rate* at which agent i receives the resource at time t. Clearly, such a schedule must satisfy the various constraints outlined above (i.e., total consumption at any time t may not exceed the production rate $s(t)$, agents cannot receive resources outside their active intervals, and they must not exceed their individual maximum consumption rates). Throughout this paper, we are interested in finding a schedule that maximises the overall *social welfare*, i.e., the sum of all agents' values, as this maximises the overall benefit to all agents.

3. SCHEDULING COOPERATIVE AGENTS

In this section, we begin by assuming that agents are completely cooperative and reveal their respective types truthfully to the mechanism on arrival. This allows us to develop a number of scheduling mechanisms which we will later (in Section 4) adapt for settings with strategic agents.

We start by discussing how to solve the *offline* version of the scheduling problem, where the scheduling mechanism has full information about all types in θ_I, including future arrivals. This will form the basis for the more realistic *online* mechanisms we propose later.

3.1 Offline Scheduling

To solve the offline problem, we are simply interested in finding a schedule ρ that maximises the overall social welfare. Formally, we define the social welfare as $W(\rho, \theta_I) = \sum_{i \in I} \delta_i(\rho, \theta_i) \cdot v_i$, where $\delta_i(\rho, \theta_i)$ is an indicator function with $\delta_i(\rho, \theta_i) = 1$ if agent i receives at least its required amount q_i under schedule ρ and $\delta_i(\rho, \theta_i) = 0$ otherwise.

Unfortunately, finding such an optimal schedule ρ is NP-hard.[1] However, we will describe how to find the optimal solution and then discuss a more tractable heuristic procedure. In both cases, we simplify the scheduling problem by partitioning time into non-overlapping intervals within which neither the set of active agents nor the production rate changes. This can be done by using all arrival times, a_i, departure times, d_i, and supply transition times, t_i, to define the partition. Since agents are indifferent about when they

[1] Briefly, this is because it is a generalisation of the NP-hard $1 \mid\mid \sum w_j U_j$ scheduling problem, i.e., minimising the weighted number of tardy jobs [10].

receive the resource within each interval, this allows us to restrict our attention to finding the appropriate consumption rate for each agent within each interval.

3.1.1 Optimal Offline Scheduling

We can solve this problem optimally by formulating it as a non-linear programming problem. Here, the objective function is the overall social welfare, the decision variables are the (constant) consumption rates for each agent within each interval, and the constraints are given by the problem domain, as described in Section 2. This problem can be solved using standard optimisation tools — we use ILOG CPLEX in our implementation and denote this mechanism by OPTIMAL. However, due to the inherent complexity of the problem, finding a solution may be time-intensive, and so we propose a faster heuristic in the next section.

3.1.2 Greedy Offline Scheduling

To solve the offline scheduling problem more quickly, we first note that it is possible to find a *feasible* schedule for a given set of agents in polynomial time by formulating it as a maximum flow problem (similar to the technique used in [4]) and then using a standard algorithm to solve it. In our implementation, we employ the Edmonds-Karp algorithm and denote this by FINDFEASIBLE (which returns an empty schedule if no feasible schedule exists).

With this, we can apply a greedy approach and iteratively add agents to an overall solution. In more detail, our heuristic greedy algorithm first orders all agents in I in decreasing order of their *value densities*, i.e., v_i/q_i. This order is important, because it is a heuristic indicator of the agent's potential value to the final solution (other indicators such as v_i could be chosen, but we found v_i/q_i to perform particularly well). Then, the algorithm iteratively considers each agent, adding it to a set of selected agents if a feasible schedule can still be found after including that agent (using FINDFEASIBLE). This continues until all agents have been considered and the final feasible schedule including all selected agents is adopted. We refer to this heuristic algorithm as GREEDY.

3.2 Online Scheduling

In an online setting, the scheduling mechanism may only have probabilistic information about future arrivals (e.g., historical data or domain knowledge). The agents' actual types are only revealed to the mechanism at their respective arrival times. Formally, the type profile at time t is given by $\theta_I^{\langle t \rangle} = \{\theta_i \mid i \in I \wedge a_i \leq t\}$. To address this setting, we compare two approaches: model-free scheduling, where no model of future arrivals is available, and model-based scheduling, where the mechanism has access to some statistical information about future arrivals. While this paper focuses on a model-based approach, we use the model-free solutions as a benchmark and to see if having a model of future arrivals provides any additional benefits.

3.2.1 Model-Free Online Scheduling

The most straight-forward model-free approach is to simply apply our offline algorithms to $\theta_I^{\langle t \rangle}$. This is done by following the schedule provided by the OPTIMAL or GREEDY offline algorithms, and repeating the algorithm every time a new agent arrives (while updating the required amount of resource for those agents that have been partially satisfied). This allows the algorithm to take advantage of higher-value agents as they arrive, possibly abandoning agents that were previously included in the schedule. Due to the complexity of solving OPTIMAL, we will focus on the heuristic version of this algorithm and denote this by ONLINEGREEDY.

Note that the schedules produced by the offline algorithm, and the FINDFEASIBLE algorithm in particular, may not necessarily result in a high utilisation of the resource in the short-term. This is because *any* feasible schedule is found, which may allocate the resource at any time in the future or at a lower rate than the supply would allow. However, in the online setting, it is desirable to fully utilise the available resources early, so that more agents can be satisfied when they arrive in the future. For this reason, we modify FINDFEASIBLE to use simple heuristic rules that shift all allocations to the earliest possible times, within the problem constraints.

So far, we have concentrated on model-free settings, where the scheduling mechanism does not anticipate future arrivals of agents. However, in many realistic settings, it is reasonable for the mechanism to have some model of the future, for example derived from historical data or the algorithm designer's own knowledge of the system. We explore this in the following section.

3.2.2 Model-Based Scheduling with Consensus

Using a model of the future is useful, because it allows the mechanism to adapt its scheduling decisions for likely future events. In particular, it can anticipate the arrival of future high-value agents and accordingly allocate less of the resource to the currently active set of agents when this is likely to be beneficial. Thus, we now assume that our mechanism has some probabilistic knowledge about future arrivals.

As discussed in Section 1, an optimal online scheduling mechanism in these settings is intractable, and so we design a model-based online mechanism based on the Consensus algorithm. This algorithm samples a number of possible future scenarios, solves these using an offline scheduling algorithm, and then uses the majority vote to select which immediate action to take. In addition to being computationally tractable, this approach has the advantage of not requiring a precise model of the future, and so it can be used even in settings where only imprecise statistical information about future arrivals is available. However, adopting Consensus in our setting raises a number of challenges. First, given the continuous-time nature of our setting, it is not clear when to make each online decision. An obvious choice here, and one that we verified experimentally to work well, is to do this every time a new agent arrives in the system.

Second, and more critically, since the resource is continuously divisible, there are an infinite number of possible actions to take. This problem is made even worse if the next action consists of a schedule until the next decision point. To address this problem, instead of voting on schedules, we modify Consensus to vote only on the agents that should be included in the online schedule (and receive their required amount of the resource) and then introduce a second phase in which we produce a feasible schedule with the selected agents. Selecting the set of agents is done iteratively by considering each potential agent in turn, using the greedy heuristic ordering discussed earlier — if the majority of offline solutions include this agent in their schedules, we accept the agent, otherwise we reject it (for the time being). As soon as an agent is accepted, all other candidate agents are tested again, this time enforcing that the accepted agent is part of the solution. This repeats until no more agents are selected, and a final feasible schedule with all accepted agents is calculated and adopted.

This procedure separates the decision of which agents to schedule (as a binary choice for each agent) from the low-level task of finding an overall schedule for all agents. Note that a valid alternative here might be to present all agents to the Consensus scenarios at once, as is done in [8], but this raises several issues. First, the offline algorithms now vote on subsets of agents to include rather than individual agents, which leads to a large decision space that

is voted upon. Moreover, a high-value agent may appear in many distinct subsets, but it may not get voted into the solution, due to its votes being divided across these subsets. This issue presents a problem not only for allocation efficiency, but also for incentive compatibility (discussed in Section 4). Serial consideration of the agents avoids these issues.

The full details are given in Algorithm 1. This algorithm keeps in its state k scenarios that are sampled from a suitable model (line 1), where each $\tilde{\theta}_I^j$ is a set of sampled *virtual* agents. For efficiency, we sample these once and re-use them every time Consensus is called. The algorithm also keeps schedule ρ (line 2), which is constructed gradually over time as more agents arrive. Specifically, we assume the function SIGNALARRIVAL($\theta_I^{\langle t \rangle}, t$) is used to notify the algorithm whenever the set of currently known types, $\theta_I^{\langle t \rangle}$, changes. When called, it executes the RUNCONSENSUS function to re-schedule the known agents, but keeps the previous schedule fixed to satisfy the online property ($\rho_{(-\infty,t)}$ is the original schedule ρ, truncated to the time interval $(-\infty, t)$).

RUNCONSENSUS first updates the set of known agent types with the amount of the resource they have received so far following schedule ρ, using an appropriate APPLYSCHEDULE function. All arrival times are also set to t here, to reflect the fact that no past allocations can be altered. Next, the algorithm retains only those virtual agents in the scenarios that arrive in the future. Now, the main loop (lines 11–23) iteratively builds a set of selected agents, C, which is initially empty. To do this, it considers the agents in the same order as our greedy heuristic (to reflect their likely value to the solution), and then solves each of the k scenarios using the offline GREEDY algorithm. If an agent is included in at least half the scenarios, we add it to C. Finally, a feasible schedule for only the agents in C is returned.

Note here that in addition to the set of candidate agents, we add two further parameters to the GREEDY offline algorithm called in line 17. Specifically, C is a subset of the types that must be included in the solution (to reflect the choice of selected agents so far), while the parameter ρ_c will become important in the setting with strategic agents and denotes part of a schedule that has been fixed up to the next decision point. However, here, it is always empty, and, similarly, the set C, holding the set of selected agents, is re-initialised to the empty set at every invocation (line 5). This gives the algorithm full flexibility to change its schedule without having to make any binding pre-commitment choices (a feature that becomes necessary in Section 4).

So far, we have assumed that agents are cooperative and reveal their private types truthfully to the scheduling mechanism. In the next section, we investigate what happens when this is not the case.

Algorithm 1 CONSENSUS Algorithm.

```
1:  θ̃_I^1, θ̃_I^2, ..., θ̃_I^k                                  ▷ Consensus scenarios
2:  ρ ← ∅                                                      ▷ Keep track of schedule
3:  s                                                          ▷ Supply
4:  procedure SIGNALARRIVAL(θ_I^⟨t⟩, t)
5:      ρ' ← RUNCONSENSUS(θ_I^⟨t⟩, ∅, ∅, t)                     ▷ Future schedule
6:      ρ ← ρ_(−∞,t) ∪ ρ'_[t,∞)
7:  procedure RUNCONSENSUS(θ_I^⟨t⟩, C, ρ_c, t)
8:      θ'_I ← APPLYSCHEDULE(θ_I^⟨t⟩, ρ, t)                     ▷ Update agents
9:      for all j ∈ {1, ..., k} do
10:         θ̃_I^j ← {θ_i | θ_i ∈ θ̃_I^j ∧ a_j > t}              ▷ Update scenarios
11:     repeat
12:         added ← false
13:         for all θ_i ← GREEDYORDER(θ'_I \ C) do
14:             if added = false then
15:                 votes ← 0                                  ▷ Initialise votes
16:                 for all j ∈ {1, ..., k} do                 ▷ Scenarios
17:                     ρ' ← GREEDY(θ̃_I^j ∪ {θ_i}, C, ρ_c)     ▷ Solve
18:                     if δ_i(ρ', θ_i) then                   ▷ In?
19:                         votes ← votes + 1                  ▷ Vote yes
20:                 if votes ≥ k/2 then
21:                     added ← true                           ▷ Include it
22:                     C ← C ∪ {θ_i}
23:     until added = false
24:     return FINDFEASIBLE(C, ρ_c, s)                         ▷ Return final schedule
```

4. SCHEDULING STRATEGIC AGENTS

In many realistic settings, agents are self-interested and cannot be assumed to provide truthful information when this is not in their best interest. To examine such settings, we will first introduce some additional terminology from the area of mechanism design (Section 4.1), then we will show why our current Consensus implementation is vulnerable to manipulation (Section 4.2) and finally show how it can be modified to incentivise truthfulness (Section 4.3).

4.1 Model

In this section, we assume that agents can *misreport* their private types to the scheduling mechanism. Specifically, in the EV setting, vehicle owners may plug in their vehicles later than their true arrival times, unplug them earlier, or report needing more charge than is actually the case. We denote by $\hat{\theta}_i = \langle \hat{c}_i, \hat{v}_i, \hat{r}_i, \hat{a}_i, \hat{d}_i \rangle$ the type agent i reveals at its *reported arrival time* $\hat{a}_i$, where $\hat{\theta}_i$ may not be equal to θ_i. However, we make a number of reasonable assumptions about the misreports an agent may make, which typically hold in practice. First, it may not report earlier arrivals and it may not report later departures, i.e., it must hold that $\hat{a}_i \geq a_i$ and $\hat{d}_i \leq d_i$. This is reasonable, because the agent must typically be physically present during its active interval to receive the resource, and so its presence can be verified (in the EV setting, a car cannot be plugged in before it physically arrives and, similarly, unplugging earlier than the reported departure could be detected). However, delaying the agent's arrival in the system or departing earlier than necessary are possible manipulations. Second, we assume that agents cannot overstate their maximum consumption rate, i.e., it must hold that $\hat{r}_i \leq r_i$.[2]

Since a scheduling mechanism now uses the reports of all agents as input, denoted by $\hat{\theta}_I$, we write the schedule produced by it as $\pi(\hat{\theta}_I, s)$. Furthermore, to examine and engineer the agents' individual incentives, we also define a *payment function* $x_i(\hat{\theta}_I)$, which determines how much agent i should pay the mechanism on its departure, given the reports of all agents. As is common in mechanism design, we use this payment function to ensure truthfulness. Given this notation, the *utility* of an agent i is $u_i(\pi, x, \theta_i, \hat{\theta}_I, s) = \delta_i(\pi(\hat{\theta}_I, s), \theta_i) \cdot v_i - x_i(\hat{\theta}_I)$, where we note that δ_i here depends on the real type θ_i.

Now, to address potential manipulation by the agents, we are interested in a property called *dominant strategy incentive compatibility* (DSIC). Briefly, if a mechanism π and payment function x are DSIC, this means that it is best for every agent to report their true type, i.e., $\hat{\theta}_i = \theta_i$, regardless of what everyone else reports. Formally, DSIC holds for π and x if and only if it is

[2]This is justified in settings where receiving the resource at a higher rate than r_i either carries a high intrinsic penalty (as is the case in the EV charging setting, where it may damage the expensive battery) or can be detected by the mechanism, for example if not all of the allocated resource is consumed. In these settings, schedules can be constructed in such a way that every agent receives the resource at its maximum rate for some time. However, to simplify the exposition, we do not include such a scheduling mechanism here.

true that $u_i(\pi, x, \theta_i, \hat{\theta}_{-i} \cup \{\theta_i\}, s) \geq u_i(\pi, x, \theta_i, \hat{\theta}_{-i} \cup \{\hat{\theta}_i\}, s)$, $\forall \theta_i, \hat{\theta}_i, \hat{\theta}_{-i}, s$, where we use $\hat{\theta}_{-i}$ to denote the reports of all agents apart from i. Additionally, to ensure participation, we also want π and x to be *individually rational* (IR), which means agents never make a loss when participating. Formally, it must hold that $u_i(\pi, x, \theta_i, \hat{\theta}_{-i} \cup \{\theta_i\}, s) \geq 0, \forall \theta_i, \hat{\theta}_{-i}, s$.

To achieve DSIC and IR, there are a number of results we can draw upon [9]. Specifically, in single-valued domains like ours, the mechanism π must be *monotonic* for DSIC and IR to hold, provided unallocated agents are not paid. To define monotonicity in the context of our work, let $\preceq$ be a partial order over the types with $\theta_i \preceq \theta_j \equiv (a_i \geq a_j) \wedge (d_i \leq d_j) \wedge (r_i \leq r_j) \wedge (q_i \geq q_j) \wedge (v_i \leq v_j)$. Then, we say π is monotonic if and only if $\delta_i(\pi(\{\theta_i\} \cup \theta_{-i}, s), \theta_i) = 1 \Rightarrow \delta_i(\pi(\{\theta_i'\} \cup \theta_{-i}, s), \theta_i') = 1$, $\forall \theta_i \preceq \theta_i', \theta_{-i}, s$. This means if an agent θ_i is allocated by an allocation mechanism π, it would remain allocated by π if its type was θ_i', where $\theta_i \preceq \theta_i'$.

Furthermore, [9] shows that, for a mechanism to satisfy DSIC and IR, the payment for each allocated agent must be equal to its *critical value*. Essentially, this is the lowest value an agent could report to the mechanism and still remain allocated. More formally, the payment of agent i must be $x_i(\{\hat{\theta}_i\} \cup \hat{\theta}_{-i}) = \min v_i'$, such that $\delta_i(\pi(\{\theta_i'\} \cup \theta_{-i}, s), \theta_i') = 1$, with $\theta_i' = \langle q_i, v_i', r_i, a_i, d_i \rangle$ ($x_i(\{\hat{\theta}_i\} \cup \hat{\theta}_{-i}) = \infty$ if there is no such v_i'). Unfortunately, we now show that Consensus, as presented in the previous section, is not monotonic.

4.2 Failure of Monotonicity

The key problem with Consensus (and the other algorithms presented in Section 3) is that it does not preserve monotonicity for agents with regard to their departure time. Often, more patient agents are delayed in favour of more constrained agents, but new arrivals may mean they are no longer allocated in the future. As a concrete example, assume there are two agents with types $\theta_1 = \langle 1, 1, 1, 0, 1 \rangle$ and $\theta_2 = \langle 1, 100, 1, 0, 2 \rangle$, and the supply is $s(t) = 1$. For sake of argument, assume that none of the scenarios sampled by Consensus expect more arrivals before time 2, so that, when reporting truthfully, both agents 1 and 2 are included in the set of selected agents, C. However, due to its tighter time constraints, the less valuable but also less patient agent 1 is scheduled first in the interval $[0, 1]$.

Now assume that a new agent arrives at time 1 with $\theta_3 = \langle 1, 200, 1, 1, 2 \rangle$ (which may possibly represent an extremely rare event that was not anticipated by Consensus). Due to its higher value, this is now included in C and scheduled in the interval $[1, 2]$, preventing agent 2 from being allocated. Given this outcome, had agent 2 lied about its patience by misreporting its own type as $\hat{\theta}_2 = \langle 1, 100, 1, 0, 1 \rangle$, it would have been allocated. This breaks monotonicity, as $\hat{\theta}_2 \prec \theta_2$, and it is also clear that it does not satisfy DSIC. In more detail, using critical value payments, the utility for agent 2 when misreporting $\hat{\theta}_2$ is $u_2(\pi, x, \theta_2, \{\theta_1, \hat{\theta}_2, \theta_3\}, s) = 100 - 1 = 99$, while a truthful report would result in being unallocated, with $u_2(\pi, x, \theta_2, \{\theta_1, \theta_2, \theta_3\}, s) = 0$.

To address this issue, we now proceed to introduce a number of modifications to Consensus that achieve monotonicity.

4.3 Truthful Allocation

As discussed in Section 1, one approach to making the allocation policy π monotonic is to apply ironing. This involves identifying cases where an allocation is made, but where the allocated agent's type might have been a misreport that violates monotonicity. In this case, the agent remains unallocated and the resource is discarded. For example, if agent 2 from the example above had reported $\hat{d}_2 = 1$, it would be left unallocated and no resource would be allocated in the interval $[0, 1]$. Clearly, this can be highly inefficient, especially since it means that had agent 2's type really been $\theta_2 = \langle 1, 100, 1, 0, 1 \rangle$, it would still remain unallocated. A second disadvantage of ironing is that testing for monotonicity violations is computationally expensive, as a large set of possible misreports has to be considered.

For this reason, we adopt a different approach. Specifically, we ensure that our Consensus algorithm, π, is monotonic in the first place, avoiding the need for ironing. We achieve this through the following modifications.

4.3.1 Pre-Commitments

First, to address the intrinsic disadvantage for patient agents in the current algorithm, we propose the notion of *pre-commitments*. To this end, we modify Consensus to make a firm commitment to schedule agents if they, at any point, receive a majority vote in the offline scenarios. Once this happens, the agent is pre-committed, which means that it is now guaranteed to receive its required resource (specifically, this is added as an additional constraint to all future scheduling decisions). We achieve this by moving the set of selected agents, C, into the state of the Consensus algorithm, instead of re-initialising it at every invocation of Consensus.

In a sense, while ironing punishes impatient agents, using pre-commitments rewards patient agents instead, because their larger flexibility means they are more likely to be included in the sampled scenarios. However, it is important to note that the commitment decisions still depend on the sampled scenarios, and so even a very patient agent may not receive a pre-commitment until it is highly likely that no better agents will arrive in the future. Furthermore, while a commitment to satisfy the agent is made, the time of when to schedule the agent is initially left open. This allows Consensus to flexibly interrupt or re-schedule the agent as necessary, e.g., when an impatient agent with a high value arrives, as long as the pre-committed agent is still satisfied eventually.

4.3.2 Re-evaluation Points

While pre-commitments ensure monotonicity with regard to departure times, agents may still strategically misreport their arrival times to be allocated (thus breaking monotonicity with regard to arrival times). This is because the decision of Consensus is sensitive to the time it is invoked. Whenever a virtual agent is removed from an offline scenario (line 10 of Algorithm 1), this may change the votes of that particular scenario. As a result, an agent might benefit from misreporting its arrival time to exactly the time where the majority of offline solutions would vote in its favour. To address this, we need to re-run the Consensus decision at every possible time where the solution might change for any unallocated agents. As just discussed, this can happen any time a virtual agent is removed from the scenarios, and, for this reason, we introduce additional *re-evaluation points* whenever this happens. More specifically, we re-run Consensus for every arrival time of a virtual agent across all scenarios.[3]

4.3.3 Partly-Fixed Schedules

However, simply re-evaluating Consensus regularly is not sufficient for monotonicity. Since the schedule ρ' produced by FINDFEASIBLE does not necessarily correspond to the schedules produced

[3]In fact, the set of pre-committed agents cannot change at all re-evaluation points, since more than one vote change is required in most cases. However, for space reasons, we do not discuss this in more detail.

Algorithm 2 CONSENSUS-PC with Pre-Commitments.

```
1:  C ← ∅                                                    ▷ Pre-committed agents
2:  t_next ← min{a_i | θ_i ∈ θ_I^j, ∀j}                       ▷ Re-evaluation time
3:  procedure SIGNALARRIVAL(θ_I^⟨t⟩, t)
4:      ρ ← MODIFIEDCONSENSUS(θ_I^⟨t⟩, C, ρ_[t,t_next), t)
5:  procedure REEVALUATE(θ_I^⟨t⟩, t)                          ▷ Called at time t = t_next
6:      t_next ← min{a_i | θ_i ∈ θ_I^j, ∀j}
7:      ρ ← MODIFIEDCONSENSUS(θ_I^⟨t⟩, C, ∅, t)
8:  procedure RUNMODIFIEDCONSENSUS(θ_I^⟨t⟩, ρ_c, t)
9:      repeat
10:         ρ'_c ← ρ_c                                       ▷ Old partly-fixed schedule
11:         ρ' ← RUNCONSENSUS(θ_I^⟨t⟩, C, ρ_c, t)
12:         ρ_c ← ρ'_[t,t_next)
13:     until ρ_c = ρ'_c                                    ▷ No more changes
14:     ρ ← ρ_(−∞,t) ∪ ρ_c
15:     return ρ
```

by the offline scenarios, following it may cause some of the offline solutions to change even before the re-evaluation point. To address this, we force Consensus to now *partly fix* the schedule over the next interval, $\rho_{[t,t_{\text{next}})}$, where t_{next} is the next re-evaluation point, and then immediately re-run Consensus, this time forcing any scheduling solution to allocate at least as much of the resource to the agents as dictated by $\rho_{[t,t_{\text{next}})}$ (this is the additional input ρ_c to the GREEDY function). This continues until no more changes are made to the set of pre-committed agents. In effect, this gives each offline scenario foresight over what will happen over the next time interval and thereby forces them to make any changes in their votes immediately rather than during the interval. For this reason, $\rho'_{[t,t_{\text{next}})}$ needs to remain fixed over this interval, although any spare resource may still be allocated, e.g., to new agents that arrive during $[t, t_{\text{next}})$. While this restricts the flexibility of the scheduling algorithm slightly, it is possible to add arbitrary new re-evaluation points, which lead to higher flexibility, but also require more evaluations of the Consensus algorithm.

A sketch of our modified Consensus algorithm is given in Algorithm 2, which only shows the main differences to Algorithm 1. Here, an additional function REEVALUATE is called for every re-evaluation point t_{next}. Note, also, when the scenarios contain no agents, this algorithm constitutes a truthful version of the model-free algorithm from Section 3.2.1. We will now show that our new Consensus mechanism (along with critical value payments) is DSIC and IR.

THEOREM 1. *Consensus with pre-commitments, re-evaluation points and partly-fixed schedules is DSIC and IR with critical value payments.*

PROOF. We prove this by using the results from [9] and showing that the modified Consensus is monotonic. To this end, we need to show that if an agent with type θ_i is allocated, it would also be allocated if it had a higher type $\theta'_i \succ \theta_i$. We do this by assuming θ'_i differs from θ_i in exactly one dimension:

Required amount ($q'_i < q_i$): Consider the time t that type θ_i is pre-committed. If θ'_i is not yet pre-committed before t, we will show that it will also be pre-committed at t. In more detail, this is because it is always considered at the same time or earlier than θ_i in the order given by GREEDYORDER (both within Consensus and the Greedy algorithm). Since an active uncommitted agent's presence has no effect on other scheduling decisions up to the point it is pre-committed (due to the serial voting procedure we use, which considers active agents independently of each other), each time feasibility is tested in the offline greedy algorithm, θ'_i will be tested with the same set or a subset of the agents that θ_i was tested with. As θ'_i requires a lower amount of the resource, it will therefore appear in at least as many feasible schedules as θ_i, thus receiving at least the same amount of votes in Consensus.

Value ($v'_i > v_i$): The same argument as above applies.

Rate ($r'_i > r_i$): This follows a similar argument as above. Here, feasibility for θ'_i is tested against the same set of constraints whenever θ_i is tested (as their position given by the greedy order is the same). Thus, when θ_i is pre-committed by being selected by the majority of offline scenarios, so will θ'_i (at the latest), because any schedule that is feasible for θ_i will also be feasible for θ'_i.

Arrival time ($a'_i < a_i$): Consider the time t at which θ_i arrives. If it is not pre-committed immediately, it will be at some future re-evaluation point t'. Since future re-evaluation points are independent of a_i or a'_i, θ'_i would also be pre-committed at the same time t'. On the other hand, if θ_i is pre-committed immediately, we can show that θ'_i would have been pre-committed earlier. To show this, we note that θ_i arrives between two re-evaluation points, α and β, for which the schedule will have been partly fixed. Now, assume θ'_i was present at re-evaluation point α and is still not pre-committed. At the latest, it will now be pre-committed once the schedule over $\rho_{[\alpha,\beta)}$ has been partly-fixed. This is because it competes against exactly the same set of virtual agents as θ_i does at time t (by definition of the re-evaluation points), with the same constraints, and so any time an offline solution contains θ_i at time t, it will also contain θ'_i at time α. A similar argument can be made when θ'_i arrives between α and t, where the same conditions hold.

Departure time ($d'_i > d_i$): This follows a similar argument as for the maximum rate. θ'_i is considered at the same position as θ_i within the greedy solution of each offline scenario. As any feasible schedule for θ_i is feasible for θ'_i, it will be pre-committed at the same time as θ_i at the latest.

We can now show that monotonicity holds also across several dimensions. For this, consider any $\theta_i \preceq \theta'_i$ that vary in x dimensions. Then, we can find a set of intermediate types $\theta_i \preceq \theta_i^1 \preceq \ldots \preceq \theta_i^{x-1} \preceq \theta'_i$, where each type varies in at most one dimension with its predecessor. Using the reasoning above, if θ_i is allocated, so must θ'_i. □

4.4 Practical Considerations

For space reasons, we do not give a detailed algorithm for calculating the critical payments here. However, this is relatively straightforward to implement and can be done using similar techniques as in [5, 8]. To give the reader an intuition, x_i can be calculated by simulating the system without agent i's presence in the interval $[t, \hat{d}_i]$, where t is the time agent i is first pre-committed. At each iteration of Consensus in the simulated system, we then first find the minimum value v'_i agent i would need to report to be included by the majority of offline scenarios (this can be done using a binary search on the greedily ordered virtual agents). Then, we calculate the minimum value v''_i the agent would need to report to be considered before the agent j that is pre-committed during that iteration ($v''_i = 0$ if none is pre-committed). The critical value for this particular iteration of Consensus is then $\max(v'_i, v''_i)$. This is repeated for every iteration and every re-evaluation point and the final payment x_i is then simply the minimum of these critical values.

An attractive feature of calculating the payments in this way is that they can be used also when agents depart earlier than anticipated. Rather than charge them arbitrary penalties, x_i can be calculated for the actual interval the agent was present and for the actual resource received. This allows agents to leave the system when necessary (e.g., in case of an unforeseen emergency), without imposing heavy fines.

5. EXPERIMENTAL EVALUATION

In this section, we apply our mechanism to scheduling the charging of EVs, using data from the first large-scale real-world trial of EVs in the UK. The purpose of this is two-fold. First, we are interested in quantifying the benefit of using a model of future arrivals in these settings. Second, we wish to determine how an incentive-compatible mechanism compares to one that assumes cooperative agents.

5.1 Experimental Setup

To evaluate our mechanisms in a realistic setting, we utilise data gathered during the CABLED (Coventry And Birmingham Low Emissions Demonstration) project.[4] As part of this project, EVs fitted with data loggers were given to the public, in order to study their daily travel and charging patterns. In our experiments, we sample actual recorded journeys from this data set to yield realistic agent arrival and departure times, and we use the actual battery charges consumed during journeys to determine an agent's required amount of electricity. We also use these distributions to (independently) sample scenarios for the Consensus-based mechanisms. Finally, we set all maximum charging rates to 3kW to reflect the capabilities of the EVs used in the trial.

Regarding an agent's value, we distinguish between two types of agents. The first, *low-value* agents, are willing to pay between £0.05 and £0.15 per kWh in their required amount (determined uniformly at random). These would rather remain uncharged by their departure time than pay much more than the usual price of electricity. The second, *high-value* agents, are willing to pay up to £1.50 – £2.50 per mile that they are planning to travel (as given by the sampled CABLED data). These constitute agents that have to travel at their departure time, for example to reach their place of work, and so their valuation represents the approximate cost of having to take a taxi instead. We will vary the relative proportions between the groups to represent varying levels of heterogeneity in the agent population.

Finally, to simulate the production rate $s(t)$, we use the average electricity consumption profile of a small neighbourhood consisting of 50 households, and allow any spare electricity to be used for EV charging whenever consumption drops to below 80% of the peak consumption. This simulates the constraints of the local transformer (with an additional safety margin to account for unexpected fluctuations in consumption), and means that no electricity is available during the peak hours of the early evening, but considerable spare capacity is available during the night.[5]

5.2 Benchmarks

To evaluate our mechanism, we use several benchmarks:

- FAIRCONTENTION: This mechanism divides the available electricity equally between all present uncharged agents. We assume agents unplug as soon as their required amount q_i is reached. As such, this represents a naïve scheduling mechanism that can be easily implemented without requiring agents to report their types.

- ONLINEGREEDY: This *model-free, non-truthful* mechanism uses the heuristic greedy algorithm to schedule the present agents (as described in Section 3.2.1).

- CONSENSUS: This *model-based, non-truthful* mechanism uses Consensus, as given by Algorithm 1. We generate 20 scenarios, as this obtains good results in practice.

- ONLINEGREEDY-PC: This *model-free, truthful* mechanism uses the ONLINEGREEDY algorithm to schedule the present agents, along with pre-commitments to ensure DSIC.

- CONSENSUS-PC: This *model-based, truthful* mechanism uses our modified Consensus algorithm (Algorithm 2).

5.3 Results

The full results of our experiments are shown in Figure 1. Here, we vary the number of EVs within the neighbourhood from 0 to 100 to represent different levels of demand,[6] and we show the results for different proportions of high-value agents, $\nu = 0.0$, $\nu = 0.25$ and $\nu = 0.75$. For statistical significance, we repeat all experiments 1000 times and plot 95% confidence intervals. We chose average social welfare (excluding payments) as the key performance metric, as this is the overall utility generated by each mechanism, and we normalise it to the performance of the offline OPTIMAL.[7] As this uses full information about future arrivals, it serves as a useful upper bound of any mechanism.

First, we consider the performance of FAIRCONTENTION (in green). Clearly, this achieves a very poor performance as soon as demand increases within the neighbourhood. When there are only 25 EVs, its performance drops to around 50% of OPTIMAL, eventually reaching an overall performance of only 5%–10%, depending on the setting. This is because the mechanism considers neither the deadlines nor the charging requirements of the users and so this highlights the need to schedule EVs in a more informed manner.

The simple non-truthful mechanisms, GREEDY and CONSENSUS (both shown in blue), achieve a significantly better performance, because they greedily select a promising set of agents to charge (guided by their value densities) and produce a feasible schedule that considers the deadlines and charging requirements of the agents. By doing this, they consistently achieve around 93% of OPTIMAL. Surprisingly, there is no significant difference between the two mechanisms, implying that the use of a model is not necessary in these settings. This is because the strategies have considerable flexibility in responding to new arrivals in the system by adapting their schedules and so there is little benefit in anticipating these beforehand.

Finally, we consider the two truthful mechanisms, GREEDY-PC and CONSENSUS-PC (both shown in red), which are the main focus on this paper. Here, we note that GREEDY-PC initially performs well, but its performance decreases quickly in settings with high demand. This is particularly pronounced when there are only a few high-value agents, i.e., $\nu = 0.25$, where it achieves less than 35% of the optimal. This is because this strategy does not use a model of the future when making pre-commitments. Thus, in the setting with $\nu = 0.25$, it will pre-commit even to low-value agents, which may then prevent it from accepting high-value agents in the future.

[4]See `http://cabled.org.uk`

[5]We use real consumption data of domestic households published by SCE for June 2010 (`http://www.sce.com/`).

[6]Beyond 50, this means that some households own more than one EV. This allows us to evaluate the mechanism in settings with very high demand, and we assume that there is no collusion between the EVs within a household.

[7]As OPTIMAL is too computationally intensive in larger settings, we also applied the offline GREEDY approach. For consistency, the data shown in the graphs uses the GREEDY data throughout as an approximation of the optimal, but we verified experimentally that there is no statistically significant difference in those settings where we ran both algorithms.

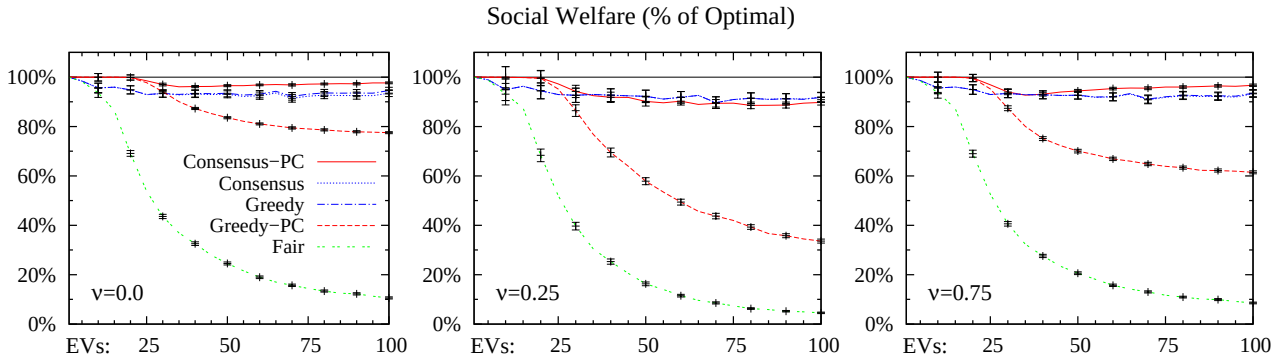

Figure 1: Average social welfare as number of EVs is increased and for various levels of heterogeneity.

In contrast, the model-based truthful mechanism we propose, CONSENSUS-PC, performs significantly better, consistently achieving 92–97% of OPTIMAL. This implies that using a model is critical, in order to make the use of pre-commitments viable in realistic settings. This is because pre-commitments are irrevocable decisions and so the mechanism must be confident that they are unlikely to have a detrimental impact on future scheduling decisions. While we do not consider the runtime performance of our mechanism in detail in this paper, we note that simulating a 24-hour day with 100 cars takes only a few seconds on a standard PC, indicating that it is feasible even in larger settings.

To conclude, we note also that CONSENSUS-PC sometimes outperforms the non-truthful benchmarks, implying that the use of pre-commitment can be beneficial even when DSIC is not a requirement. This is because it forces the algorithm to choose and concentrate on a set of agents, ensuring that these are fully charged. In contrast, the more flexible GREEDY and CONSENSUS re-evaluate their chosen set of agents each time. This means they can initially start charging some vehicles, which are later displaced by others with a higher value density (but not necessarily higher value). Thus, some amount of electricity is wasted, as no value is derived from partially charged vehicles.

6. CONCLUSIONS

In this paper, we considered an online setting where self-interested agents compete for a limited, expiring resource that is continuously being produced. To address shortcomings in existing work, we modified the well-known Consensus algorithm and introduced the novel concept of pre-commitments to ensure incentive compatibility.

Furthermore, we showed how our mechanism can be applied to the challenging problem of scheduling the charging of electric vehicles, a key bottleneck for the widespread adoption of EVs. In experiments using real data, we demonstrated that our mechanism considerably outperforms approaches that divide electricity equally between cars (without considering the owners' individual preferences), and we showed that the cost of ensuring incentive compatibility is small when coupled with a probabilistic model of the future.

However, our work has implications beyond the domain of electric vehicle charging. It can be applied in many application areas with expiring resources, including the scheduling of computational jobs in cloud settings, the allocation of bandwidth in networks or even allocating employees' time to projects within a business.

In future work, we plan to deploy our mechanism in real-world trials, which will focus particularly on the design of appropriate interfaces to allow car owners to express their preferences and interact with the mechanism in a non-obtrusive manner. Other relevant extensions include dealing with multi-value domains, where the utility an agent derives depends on the amount of resource received, and we will explicitly explore other application areas for our mechanism.

7. ACKNOWLEDGEMENT

This work was funded by the ORCHID (`www.orchid.ac.uk`) and iDEaS projects (`www.ideasproject.info`).

8. REFERENCES

[1] R. Bent and P. Van Hentenryck. The value of consensus in online stochastic scheduling. In *ICAPS'04*, 2004.

[2] K. Clement-Nyns, E. Haesen, and J. Driesen. The impact of charging plug-in hybrid electric vehicles on a residential distribution grid. *IEEE Transactions on Power Systems*, 25(1):371–380, 2010.

[3] F. Constantin and D. Parkes. Self-correcting sampling-based dynamic multi-unit auctions. In *EC'09*, pages 89–98, 2009.

[4] A. Federgruen and H. Groenevelt. Preemptive scheduling of uniform machines by network flow techniques. *Management Science*, 32(3):341–349, 1986.

[5] E. Gerding, V. Robu, S. Stein, D. Parkes, A. Rogers, and N. Jennings. Online mechanism design for electric vehicle charging. In *AAMAS'11*, pages 811–818, 2011.

[6] P. V. Hentenryck and R. Bent. *Online Stochastic Combinatorial Optimization*. MIT Press, 2006.

[7] R. A. of Engineering. *Electric Vehicles: Charged with potential*. Royal Academy of Engineering, 2010.

[8] D. Parkes and Q. Duong. An ironing-based approach to adaptive online mechanism design in single-valued domains. In *AAAI'07*, volume 22, page 94, 2007.

[9] D. C. Parkes. *Algorithmic Game Theory*, chapter Online mechanisms, pages 411–439. Cambridge University Press, 2007.

[10] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 3rd edition, 2008.

[11] R. Porter. Mechanism design for online real-time scheduling. In *EC'04*, pages 61–70, 2004.

[12] S. Vandael, K. D. Craemer, N. Boucké, T. Holvoet, and G. Deconinck. Decentralized coordination of plug-in hybrid vehicles for imbalance reduction. In *AAMAS'11*, pages 803–810, 2011.