An Efficient Partial-Order Representation of Feasible Schedules for Online Decisions

Saajid Haque, Donatello Materassi, Saverio Bolognani, Mardavij Roozbehani, Munther A. Dahleh

***Abstract*— In many scheduling problems involving sporadic tasks with multiple deadlines, there is typically a large degree of flexibility in determining which tasks to serve at each time step. Given a cost function it is often possible to cast a scheduling problem as an optimization problem to obtain the most suitable schedule. However, in several applications, especially when the schedule has to be computed in-line or periodically adjusted, the cost function may not be completely known a priori but only partially. For example, in some applications only the cost of the current allocation of resources to the tasks could be available. Under this scenario, a sensible approach is to optimally allocate resources without compromising the schedulability of the tasks. The article tackles this problem by introducing a notion of partial ordering on the set of feasible schedules. This partial ordering is particularly useful to characterize which allocations of resources at the current time preserve the feasibility of the problem in the future. This enables the realization of fast algorithms for real-time scheduling. The model and algorithm presented in this article can be utilized in different applications such as electric vehicle charging, cloud computing and advertising on websites.**

I. INTRODUCTION

Scheduling is a decision-making process that allocates resources to tasks over a period of time and in most cases, attempts to optimize their completion according to a given objective function [1]. The resources and tasks can be of different nature based on the application, but the mathematical models used to describe a scheduling problem share many characteristics. The general problem consists of a set of tasks with potentially heterogeneous deadlines, where all of the tasks cannot be attended to at the same time or instantaneously. Consequently, a schedule is essential to ensure that all of the tasks are completed before their deadlines. Linear programming is a prevalent tool to compute schedules and can also be used to optimize with respect to a linear cost function [1]. Other optimization techniques are, of course, also possible [2], [3]. In some applications, while the constraints are predetermined (such as the demand and deadline), the cost function may not be completely known a priori, either because it depends on exogenous factors, or because of privacy concerns in multi-agent scenarios. Indeed the overall cost function might be given by the sum of the costs of the specific actions taken at each time t, while the cost of future actions is not known. Since the space of current actions is smaller than the space of all schedules, a characterization of all admissible actions can lead to a computationally efficient algorithm. The algorithm presented in this article optimizes with respect to the current action instead of the global schedule while guaranteeing that the deadlines will still be met. Potential applications span several domains.

S. Haque and D. Materassi are with the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996, USA. shaque6@vols.utk.edu and dmateras@utk.edu

S. Bolognani is with the Automatic Control Laboratory, ETH Zürich, 8092 Züirch, Switzerland. bsaverio@ethz.ch

M. Roozbehani and M. Dahleh are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. mardavij@mit.edu and dahleh@mit.edu

A. Electric Vehicle Charging

In the power grid, a fundamental concept is that supply must equal the demand constantly. Traditionally, the output of the generators is assumed to be controllable while user demand is treated to be inelastic [4]. However, the addition of wind and solar to the grid challenges the convention that generation must satisfy all loads instantaneously. The variability of alternative energy sources poses an obstacle to transitioning to a grid more reliant on renewable energy. A strategy being explored changes the idea that the loads are rigid and treats them as deferrable. For example, electric vehicle batteries among other loads can be considered to be flexible.

Various studies have been completed recently on supplying flexible loads with different demand side schemes. Heuristic scheduling algorithms such as least slack first (LSF) and earliest deadline first (EDF) are presented in [5] to demonstrate the possible scheduling of deferrable loads to utilize renewable energy sources. An energy delivery management method is developed in [6] that shows consumers may receive a significant discounted price for EV charging by delaying consumption. Other approaches [7] assume partial information given by the consumer and create an efficient pricing method that incentives customers to reveal their true deadlines regardless of actions by others. A non-causal algorithm is presented in [8] that fulfills all the energy requirements for a group of deferrable loads without surplus or deficit. A decentralized optimization approach is proposed in [9] for the case in which each agent's utility depends on an individual cost function and on a common exogenous price. The model presented in this article can be generally applied to a group of flexible loads that are connected to the same bus so that there is an aggregate limit on how many tasks can be served at the same time.

B. Cloud Computing

Another problem that may fall under the model of this article is real-time scheduling of tasks in the cloud [10]. The problem is to execute a large number of time-sensitive tasks

that need to be completed with limited computational power, and a schedule can lead to efficient use of the resources by determining the use of the processors and the priority of the tasks [11]. A general method to derive schedulability conditions for these types of systems is presented in [12] where real-time tasks are considered with deadlines on a platform of identical processors. A task splitting approach is explored in [13] for scheduling sporadic task systems on multiprocessor platforms. The authors of [10] utilize an algorithm combining priority based and EDF techniques to provide better performance by reducing the total execution time of the tasks in cloud computing. Our method is different in that all of the possible choices at the current time t can be determined, and then a suitable action can be chosen to move forward with.

C. Advertising on Websites

Advertising revenue is a significant source of income for many websites. The goal is to maximize profits by having an optimal advertising schedule and policy. Websites attempt to target specific groups with particular product advertisements based on identification characteristics such as user keyword searches. At the same time, most advertisements need to be shown a certain number of times within an appropriate time horizon. The model proposed in this paper could be applied to a scheme that provides guarantees about the frequency under which a product is promoted. In addition to ensuring admissibility, the cost function could be related to matching the commercials with targeted groups of users who are currently browsing the website.

II. PROBLEM FORMULATION

We consider a problem setup analogous to the one described in [14]. First we introduce a definition for a task.

Definition 1 (Task): A task B is a 3-tuple $(D, T, \overline{u}) \in \mathbb{R}^3$ where $D > 0$ defines the demand, T is the deadline, i.e. the time left to satisfy such demand, and $\overline{u} > 0$ is the maximum serving rate.

In line with many practical applications, and with minimal loss of generality, we introduce a discretization of time and of the parameters that define the tasks. We assume a unitary sampling time and introduce the following definition.

Definition 2 (Integer task): A task $(D, T, \overline{u})$ is *integer* if $T \in \mathbb{N}$ and if $\frac{D}{\overline{u}} \in \mathbb{N}$.

We can therefore introduce the following discrete time model to describe the dynamic process of task completion

$$\begin{aligned} x_{k+1} &= x_k + u_k \\ x_0 &= -D \end{aligned}$$

where $x_k \leq 0$ is the state of completion of the task, and $u_k \geq 0$ is the rate at which the task is served (in a piecewise constant way) during the k-th interval, namely between time k and $k+1$.

The constraint on the maximum rate and the deadline for task completion correspond respectively to the inequalities

$$0 \leq u_k \leq \overline{u}, \quad \text{for } k = 0, \dots, T-1,$$

and the equality

$$x_T = 0.$$

We now consider the case of multiple task, the behaviors of which are coupled by the fact that, at any time, the aggregate serving rate (the sum of the rates at which different tasks are served) cannot exceed a given limit $\overline{U}$.

We therefore consider N tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$, where $i = 1, ..., N$, and we consider the following problem.

Problem 3 (Task scheduling problem): We define the *task scheduling problem* as the problem of deciding the variables $u_k^{(i)}$ for $k = 0, ..., T := \max_{1 \leq i \leq N} T^{(i)}$ such that for any task $i = 1, ..., N$

$$\begin{aligned} x_{k+1}^{(i)} &= x_k^{(i)} + u_k^{(i)} \\ x_0^{(i)} &= -D^{(i)} \\ x_{T^{(i)}}^{(i)} &= 0 \end{aligned} \tag{1}$$

where $x_k^{(i)} \leq 0$ is the state of task i at time k, and such that, at any time k,

$$\begin{aligned} &0 \leq u_k^{(i)} \leq \overline{u}^{(i)} \quad \text{for } i = 1, ..., N \\ &\sum_{i=1}^{N} u_k^{(i)} \leq \overline{U}. \end{aligned} \tag{2}$$

We refer to $[0, T]$ as the horizon of the problem.

We introduce the following definitions.

Definition 4 (Action): We define the *action* at time k as the vector in $u_k = [u_k^{(1)} \dots u_k^{(N)}]^\intercal$ obtained by stacking the rates of the different tasks.

Definition 5 (Feasible schedule): Given the tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$ for $i = 1, ..., N$, and an aggregate limit $\overline{U}$, a schedule (i.e. a sequence of actions) $\{u_k\}_{k=0}^{T-1}$ is *feasible* if it solves the task scheduling problem, i.e. it satisfies the constraints (1) and (2).

Definition 6 (Schedulable tasks): Given an aggregate limit $\overline{U}$, the tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$ for $i = 1, ..., N$, are *schedulable* if a feasible schedule for the corresponding task scheduling problem exists.

The goal of the analysis presented in this paper is to give necessary and sufficient conditions so that the actions of the users meet the constraints of the task scheduling problem. While implementing a schedule in real time, it is of paramount importance to choose an appropriate action that does not compromise feasibility of the schedule in the future.

Our analysis is performed at time $k = 0$ and provides guarantees that, if the tasks are schedulable to begin with, then a certain action u_0 leaves the tasks in a schedulable configuration over the horizon $[1, T]$. Then the same approach can be iteratively applied.

An effective characterization of the set of all (and only) the admissible actions allows one to determine an optimal action according to some cost criteria.

We therefore introduce the following definitions.

Definition 7 (Admissible action): Given the tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$ for $i = 1, ..., N$, and an aggregate

limit $\overline{U}$, an action

$$u_0 = \begin{bmatrix} u_0^{(1)} \\ \vdots \\ u_0^{(N)} \end{bmatrix}$$

is *admissible* if:

- it satisfies

$$0 \leq u_0^{(i)} \leq \overline{u}^{(i)},\ i = 1, \ldots, N, \quad \text{and} \quad \sum_{i=1}^{N} u_0^{(i)} \leq \overline{U},$$

- the tasks $(D^{(i)} - u_0^{(i)}, T^{(i)} - 1, \overline{u}^{(i)})$ are schedulable with the global constraint $\overline{U}$.

We now make a technical assumption, with minimal loss of generality.

Assumption 8: The tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$, $i = 1, ..., N$, are all integer tasks. Moreover, there exist a common $\overline{u}$, that we denote as the *unit rate*, such that

$$\frac{D^{(i)}}{\overline{u}} \in \mathbb{N}, \quad \frac{\overline{u}^{(i)}}{\overline{u}} \in \mathbb{N}, \quad \frac{\overline{U}}{\overline{u}} \in \mathbb{N},$$

for all $i = 1, ..., N$.

Based on this, we introduce the following definition.

Definition 9 (Integer schedule): Given a task scheduling problem, a schedule is integer if all its elements are integer multiples of a unit rate $\overline{u}$.

In the following, we will focus exclusively on *integer schedules*. This choice is reasonable in the common practice where most of the time tasks can be served at some quantized rate of service (i.e. according to the number of processors assigned to the thread in a multiprocessor system [1], or according to some of the EV battery charging standards where different levels of charging speed are available).

It is worth remarking that, even if the problem of finding a solution to the scheduling problem (1)-(2) becomes easier if the integer constraints are relaxed (and reduces to the application of linear programming to load scheduling [15], [16]), a practical characterization of all the admissible actions remains a difficult problem. On the other hand, the following result shows that the restriction to integer schedules has no effect on the schedulability analysis of the task scheduling problem.

Theorem 10: Consider a set of N tasks $(D^{(i)}, T^{(i)}, \overline{u}^{(i)})$, with an aggregate bound $\overline{U}$ and let Assumption 8 hold. The tasks are schedulable if and only if an integer schedule exists. The proof for this theorem is shown in [14].

Moreover, there is no loss of generality by assuming that all tasks have the same rate constraint $\overline{u}^{(i)} = 1$, since the general case can always be reformulated in this way, by "splitting" each task into smaller tasks with homogeneous rate constraints [14].

Definition 11 (Aggregate serving rate): Given an action u_0, we indicate its *aggregate serving rate* as

$$\|u_0\|_1 = \sum_{i=1}^{N} u_0^{(i)}. \tag{3}$$

We also borrow the notion of slack from the area of multiprocessor scheduling [1].

Definition 12 (Slack): We define the *slack* of a task as

$$slack(B^{(i)}) = s^{(i)} := T^{(i)} - \frac{D^{(i)}}{\overline{u}^{(i)}}. \tag{4}$$

If the serving rate is assumed to be unitary $\overline{u}^{(i)} = 1$, Eq. 4 reduces to

$$slack(B^{(i)}) = s^{(i)} := T^{(i)} - D^{(i)}. \tag{5}$$

The slack represents the maximum idle time that the task can wait, before having to be necessarily served, in order to meet its deadline.

Based on this definition, we introduce a function $\mathbf{s}$ of the action u_k that returns the slacks of the tasks that are served according to that action.

Definition 13 (Slack Vector): $\mathbf{s}(\cdot)$ maps $u \in \{0,1\}^N$ into a vector of $\overline{U}$ entries that can either be integers or ∞

$$\mathbf{s} : \{0,1\}^N \to (\mathbb{N} \cup \{\infty\})^{\overline{U}}. \tag{6}$$

Let j be the number of tasks that are served in u_k. The first j entries of $\mathbf{s}(u_k)$ are the ordered slacks of the j tasks being served and the remaining $(|\overline{U}| - j)$ entries are ∞.

As an example, in the case of an action serving tasks 2 and 3 among 5 tasks and $\overline{U} = 3$, we have that

$$\mathbf{s}\left(\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} s^{(2)} \\ s^{(3)} \\ \infty \end{bmatrix}$$

under the assumption that $s^{(2)} \leq s^{(3)}$.

It is possible to define a *partial ordering* relation between vectors of dimension $\overline{U}$ via the following definition.

Definition 14 (Slack partial ordering): Let $\mathbf{s}'$ and $\mathbf{s}''$ be two slack vectors with dimension d. We say that

$$\mathbf{s}' \preccurlyeq \mathbf{s}''$$

if the i^{th} entry of $\mathbf{s}'$ is less than or equal to the i^{th} entry of $\mathbf{s}''$ for $i = 1, ..., d$.

Consequently, two vectors are not slack-comparable if neither of the following hold

$$\mathbf{s}' \preccurlyeq \mathbf{s}''$$

$$\mathbf{s}' \succcurlyeq \mathbf{s}''.$$

As an example, we have that $\left[\begin{smallmatrix} 1 \\ 3 \end{smallmatrix}\right] \preccurlyeq \left[\begin{smallmatrix} 2 \\ 4 \end{smallmatrix}\right]$, while $\left[\begin{smallmatrix} 1 \\ 4 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} 2 \\ 3 \end{smallmatrix}\right]$ are not slack-comparable.

Given an admissible action, the following results shows that it is possible to use the slack partial ordering to find other admissible actions.

Theorem 15: Consider the tasks $B^{(i)} = (D^{(i)}, T^{(i)}, 1)$ for $i = 1, ..., N$, together with the aggregate constraint $\overline{U}$, and let Assumption 8 hold. Let u_0 be an admissible action. Then any action v_0 is also admissible if

$$\mathbf{s}(v_0) \preccurlyeq \mathbf{s}(u_0).$$

Proof: Refer to [14]. ∎

In other words, Theorem 15 states that if an action is admissible, then any other action smaller or equal in slack, in the sense provided by the relation "$\preccurlyeq$", is also admissible.

A natural question arises whether a converse necessary condition exists. In other words, whether all the admissible actions need to be smaller in slack than a given unique maximal element, or if instead there exist admissible actions that are not slack-comparable and are not smaller in slack than a common admissible action. The answer to this question is provided in the next section.

III. UNIQUE MAXIMAL ELEMENT

Using the partial ordering relation described in the problem formulation, a maximal slack vector can be found to help determine all the admissible actions.

Theorem 16: Given a feasible scheduling problem $\mathcal{P}$, there exists a unique slack vector $\hat{\mathbf{s}}$ where an action u_0 is admissible if and only if

$$\mathbf{s}(u_0) \preccurlyeq \hat{\mathbf{s}}.$$

Proof: The proof is omitted for the sake of space. ■

From Theorem 16, it becomes possible to test if any given action u_0 is admissible. Indeed, $\mathbf{s}(u_0) \preccurlyeq \hat{\mathbf{s}}$ if and only if u_0 is admissible. Thus, the knowledge of $\hat{\mathbf{s}}$ completely characterizes the set of all possible admissible actions. The problem then becomes to find $\hat{\mathbf{s}}$ in an efficient way. The Maximal Element Algorithm implements a search for $\hat{\mathbf{s}}$ by operating sequentially on the entries of a $\overline{U}$-dimensional vector computing $\hat{\mathbf{s}}$ in provably $\mathcal{O}(\overline{U}N^2T)$ steps in the worst case scenario. It makes use of the Minimum Effort Algorithm proposed in [14] to test the schedulability of a given set of tasks.

Minimum Effort Algorithm

INPUT

- tasks: $(D^{(i)}, T^{(i)}, 1)$, $i = 1, ..., N$
- aggregate limit: $\overline{U}$

EXECUTE

1) INITIALIZE
   - $T := \max_{1 \le i \le N} T^{(i)}$
   - $u_k^{(i)} = 0$ for $0 \le k < T$, $1 \le i \le N$
2) FOR $k = T - 1 : -1 : 0$
   - Create **ActiveTasks**:= list of all tasks with deadline $T^{(i)} > k$ and demand $D^{(i)} > 0$
   - Compute $N_A :=$ # of elements in **ActiveTasks**
   - Order **ActiveTasks** according to their *reverse* slack $r^{(i)} := k + 1 - D^{(i)}$
   - For the first $\min(N_A, \overline{U})$ tasks with least reverse slack $r^{(i)}$ in **ActiveTasks**, decrease $D^{(i)}$ by 1 and assign $u_k^{(i)} := 1$

   OUTPUT:
   - schedulability: if $D^{(i)} = 0$ for all $i = 1, ..., N$ return **TRUE** otherwise return **FALSE**
   - schedule: $u_k^{(i)} = 0$ for $0 \le k < T$, $1 \le i \le N$
   - effort: $\underline{U} := \sum_{i=1}^{N} u_0^{(i)}$

Maximal Element Algorithm

INPUT

- The ordered set of all tasks augmented with $\overline{U}$ auxiliary tasks labeled as ∞ $\{B^{(1)}, B^{(2)}, ..., B^{(n)}, \infty^{(n+1)}, ..., \infty^{(\overline{U})}\}$.
  ∞ is a special task such that its slack = ∞.

EXECUTE

- INITIALIZE
  - Create the action with the least slack vector $u = \{B^{(1)}, B^{(2)}, ..., B^{(\overline{U})}\}$.
- FOR $d = \overline{U} : -1 : 1$
  - WHILE (u is admissible AND $\|u\| \ge d$)
    - Let $\{B^{i_1}, B^{i_2}, ..., B^{i_{\overline{U}}}\}$ be the tasks served in u in ascending order of slack.
    - Take action u and replace B^{i_d} with $B^{i_{d+1}}$.
    - IF ($B^{i_{d+1}} == \infty$) then remove $B^{i_{d+1}}$ in action u.
  - Check the admissibility of action u with the Minimum Effort Algorithm.
  - If (u is not admissible), then take action u and replace $B^{i_{d+1}}$ with B^{i_d}.

  OUTPUT:
- $\hat{\mathbf{s}}$ = the slack vector of u.

Tasks	$B^{(1)}$	$B^{(2)}$	$B^{(3)}$	$B^{(4)}$	$B^{(5)}$	$B^{(6)}$
Demand $D^{(i)}$	4	2	2	3	2	1
Deadline $T^{(i)}$	5	3	3	4	5	5
Slack $s^{(i)}$	1	1	1	1	3	4

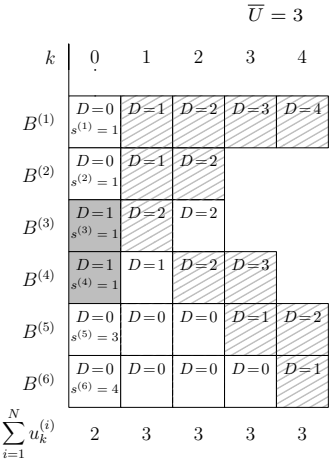


Fig. 1. A schedule is shown with the initial action serving $B^{(3)}$ and $B^{(4)}$ corresponding to a slack vector of $[1, 1, \infty]^\intercal$. A gray box indicates that at time 0 the task i is served ($u_k^{(i)} = 1$) while a white box indicates that the task is idle ($u_k^{(i)} = 0$). The shaded boxes show a possible projected schedule for the future providing that the chosen action at time 0 is admissible. Each box also contains the numerical values of $D^{(i)}$. As all the demands are served for all tasks after the iteration at time $k = 4$, the problem is schedulable.

Tasks	$B^{(1)}$	$B^{(2)}$	$B^{(3)}$	$B^{(4)}$	$B^{(5)}$	$B^{(6)}$	$B^{(7)}$
Demand $D^{(i)}$	4	1	1	1	2	2	1
Deadline $T^{(i)}$	4	2	2	2	4	4	4
Slack $s^{(i)}$	0	1	1	1	2	2	3

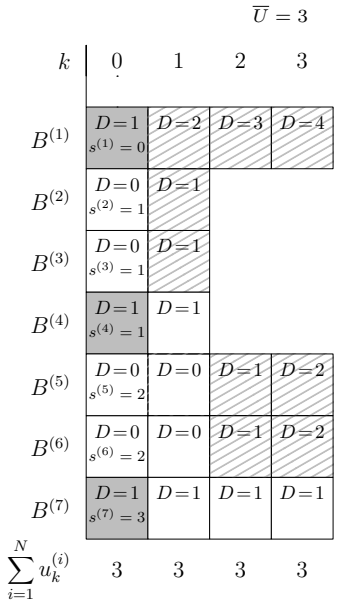


Fig. 2. A schedule is shown with the initial action serving $B^{(1)}$, $B^{(4)}$ and $B^{(7)}$ corresponding to a slack vector of $[0, 1, 3]^\intercal$. A gray box indicates that at time 0 the task i is served ($u_k^{(i)} = 1$) while a white box indicates that the task is idle ($u_k^{(i)} = 0$). The shaded boxes show a possible projected schedule for the future providing that the chosen action at time 0 is admissible. Each box also contains the numerical values of $D^{(i)}$. As all the demands are served for all tasks after the iteration at time $k = 3$, the problem is schedulable.

Tasks	$B^{(1)}$	$B^{(2)}$	$B^{(3)}$	$B^{(4)}$	$B^{(5)}$	$B^{(6)}$	$B^{(7)}$	$B^{(8)}$	$B^{(9)}$
Demand $D^{(i)}$	4	3	5	2	6	5	3	1	2
Deadline $T^{(i)}$	4	4	6	4	8	8	7	6	8
Slack $s^{(i)}$	0	1	1	2	2	3	4	5	6
Bid $p^{(0)}$	\$1.5	\$2.1	\$5.8	\$1.9	\$2.5	\$3.5	\$3.7	\$2.3	\$7.1

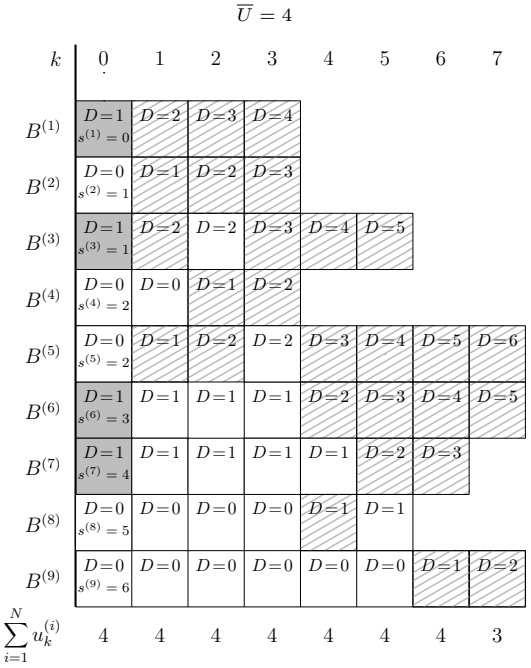


Fig. 3. A schedule is shown with the initial action serving $B^{(1)}$, $B^{(3)}$, $B^{(6)}$, and $B^{(7)}$ corresponding to a slack vector of $[0, 1, 3, 4]^\intercal$. A gray box indicates that at time 0 the task i is served ($u_k^{(i)} = 1$) while a white box indicates that the task is idle ($u_k^{(i)} = 0$). The shaded boxes show a possible projected schedule for the future providing that the chosen action at time 0 is admissible. Each box also contains the numerical values of $D^{(i)}$. As all the demands are served for all tasks after the iteration at time $k = 7$, the problem is schedulable.

A. Example One

The problem in Figure 1 shows that an example where the maximal slack vector for the initial action does not reach the aggregate bound. The aggregate bound is $\overline{U} = 3$, but only two tasks need to be served initially to keep schedulability for this example. The maximal slack vector is determined to be $\hat{\mathbf{s}} = [1, 1, \infty]^\intercal$ and the ∞ shows that a third task is not served in this action. Using this $\hat{\mathbf{s}}$, the initial action was chosen to be tasks $B^{(3)}$ and $B^{(4)}$ in Figure 1. However, these tasks are not the only ones that could be chosen to correspond to the maximal slack vector. Tasks $B^{(1)}$ or $B^{(2)}$ could have been chosen to be served in the initial action as well since $s^{(1)} = s^{(2)} = s^{(3)} = s^{(4)} = 1$. Also, it is possible to serve another task, with any slack z, together with the two tasks chosen in this example, as $[1, 1, z]^\intercal \preccurlyeq [1, 1, \infty]^\intercal$ for any z.

B. Example Two

The example in Figure 2 shows that a scalar such as the sum of the slacks does not necessarily determine admissibility of an action. A vector is needed to determine admissibility of an action. This scheduling problem consists of a set of 7 integer tasks with heterogeneous deadlines with $\overline{U} = 3$ where the maximal slack vector is determined to be $\hat{\mathbf{s}} = [0, 1, 3]^\intercal$. Even though $\hat{\mathbf{s}}$ has the same sum as the slack vector $[0, 2, 2]^\intercal$, this slack vector does not correspond to an admissible initial action, and this can be seen in the Figure 2. Actions with slack vectors of $[0, 1, 2]^\intercal$ or $[0, 1, 3]^\intercal$ are both admissible because they are both smaller in slack than $\hat{\mathbf{s}}$.

C. Example Three

After determining the maximal slack vector $\hat{\mathbf{s}}$, all of the admissible actions at time zero are known because every action with a slack vector less than $\hat{\mathbf{s}}$ will also be an admissible action. This information can be used to implement an optimal policy when the cost of the current action is available and the cost of future actions is still to be revealed. For example, we could consider a service that guarantees the completion of a task by deadline T. To participate the user must pay an entry fee, but the user may decide to modify its priority and bid in order to be served at specific time slots. A scheme of this kind can be used to implement a demand response mechanism to supply energy to different flexible loads while at the same time take into account their time

preferences.

The example in Figure 3 is a set of 9 integer tasks with heterogeneous deadlines with $\overline{U} = 4$, and an attempt is made to maximize the profit based on a bidding system assuming no a-priori knowledge of the future bids. By using the Maximal Element Algorithm, the maximal slack vector for this problem at time 0 is computed and $\hat{\mathbf{s}} = [0, 2, 3, 5]^\intercal$. Consequently any action with a smaller slack vector than the maximal slack element is also admissible. In the example, the slack of $B^{(1)}$ is 0 and must be served until it is completed. Therefore, the slack vector $[0, 1, 3, 4]^\intercal$ (corresponding to $B^{(1)}$, $B^{(3)}$, $B^{(6)}$, $B^{(7)}$) would maximize the profit from the bids. Even though $B^{(9)}$ has the highest bid, it is not possible to serve this task initially or schedulability for the problem would be lost.

IV. CONCLUSIONS

For a scheduling problem, this article shows that the set of all admissible actions at a given time can be characterized via a partial order condition on the slack of the tasks to be served. This partial order condition is completely described by a unique maximal slack element. The maximal slack element can be determined using the Maximal Element Algorithm, and then all the admissible actions at the current time step can be found efficiently. Using this algorithm, a scheme can be implemented that optimizes with respect to the current action instead of the global schedule while keeping schedulability. The concepts presented in this article can be exploited to develop fast algorithms for scheduling problems associated with cost functions where future costs are not known a priori. If information about future bids is known, such information could be potentially factored in the cost of a certain action. In such a case, the characterization of the set of all admissible actions presented could potentially provide a more efficient strategy to optimally solve scheduling problems via dynamic programming.

V. ACKNOWLEDGMENTS

The authors D. Materassi and S. Haque acknowledge ARPA-E for supporting this research through the project titled *A Robust Distributed Framework for Flexible Power Grids*.

REFERENCES

[1] M. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer, 2012.

[2] A. Zakariazadeh, S. Jadid, and P. Siano, "Multi-objective scheduling of electric vehicles in smart distribution system," *Energy Conversion and Management*, vol. 79, pp. 43 – 53, 2014.

[3] P. You, Z. Yang, Y. Zhang, S. H. Low, and Y. Sun, "Optimal charging schedule for a battery switching station serving electric buses," *IEEE Transactions on Power Systems*, vol. 31, no. 5, pp. 3473–3483, 2016.

[4] P. P. Varaiya, F. F. Wu, and J. W. Bialek, "Smart operation of smart grid: Risk-limiting dispatch," *Proceedings of the IEEE*, vol. 99, no. 1, pp. 40–57, 2011.

[5] A. Subramanian, M. Garcia, D. Callaway, K. Poolla, and P. Varaiya, "Real-time scheduling of distributed resources," *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 2122–2130, Dec. 2013.

[6] M. Kefayati and C. Caramanis, "Efficient energy delivery management for phevs," in *2010 First IEEE International Conference on Smart Grid Communications*. IEEE, 2010, pp. 525–530.

[7] E. Bitar and Y. Xu, "Deadline differentiated pricing of deferrable electric loads," *IEEE Transactions on Smart Grid*, vol. 8, no. 1, pp. 13–25, 2017.

[8] H. Hao and W. Chen, "Characterizing flexibility of an aggregation of deferrable loads," in *IEEE Conference on Decision and Control*. IEEE, 2014, pp. 4059–4064.

[9] F. Parise, M. Colombino, S. Grammatico, and J. Lygeros, "Mean field constrained charging policy for large populations of plug-in electric vehicles," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 5101–5106.

[10] G. Gupta, V. K. Kumawat, P. R. Laxmi, D. Singh, V. Jain, and R. Singh, "A simulation of priority based earliest deadline first scheduling for cloud computing system," in *2014 First International Conference on Networks Soft Computing (ICNSC2014)*, Aug 2014, pp. 35–39.

[11] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011.

[12] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 4, pp. 553–566, 2009.

[13] B. Andersson, K. Bletsas, and S. Baruah, "Scheduling arbitrary-deadline sporadic task systems on multiprocessors," in *2008 Real-Time Systems Symposium*, Nov 2008, pp. 385–394.

[14] D. Materassi, S. Bolognani, M. Roozbehani, and M. Dahleh, "Deferrable loads in an energy market: coordination under congestion constraints," in *Mediterranean Conference on Control and Automation*. IEEE, 2014, pp. 628–633.

[15] J. Chen, F. N. Lee, A. M. Breipohl, and R. Adapa, "Scheduling direct load control to minimize system opoperation cost," *IEEE Transactions on Power Systems*, vol. 10, no. 4, pp. 1994–2001, Nov. 1995.

[16] K.-H. Ng and G. B. Sheble, "Direct load control - a profit-based load management using linear programming," *IEEE Transactions on Power Systems*, vol. 13, no. 2, pp. 688–695, May 1998.