Mechanism Design for Online Real-Time Scheduling

Ryan Porter*
Computer Science Department
Stanford University
Stanford, CA 94305
rwporter@stanford.edu

ABSTRACT

For the problem of online real-time scheduling of jobs on a single processor, previous work presents matching upper and lower bounds on the competitive ratio that can be achieved by a deterministic algorithm. However, these results only apply to the non-strategic setting in which the jobs are released directly to the algorithm. Motivated by emerging areas such as grid computing, we instead consider this problem in an economic setting, in which each job is released to a separate, self-interested agent. The agent can then delay releasing the job to the algorithm, inflate its length, and declare an arbitrary value and deadline for the job, while the center determines not only the schedule, but the payment of each agent. For the resulting mechanism design problem (in which we also slightly strengthen an assumption from the non-strategic setting), we present a mechanism that addresses each incentive issue, while only increasing the competitive ratio by one. We then show a matching lower bound for deterministic mechanisms that never pay the agents.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; J.4 [**Social and Behavioral Sciences**]: Economics; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Online computation*

General Terms

Algorithms, Economics, Design, Theory

Keywords

Mechanism Design, Game Theory, Online Algorithms, Scheduling

*The author would like to acknowledge Yoav Shoham for his influential comments on drafts of the paper, and Yossi Azar for a useful discussion. This work was supported in part by DARPA grant F30602-00-2-0598.

EC'04, May 17–20, 2004, New York, New York, USA.

1. INTRODUCTION

We consider the problem of online scheduling of jobs on a single processor. Each job is characterized by a release time, a deadline, a processing time, and a value for successful completion by its deadline. The objective is to maximize the sum of the values of the jobs completed by their respective deadlines. The key challenge in this online setting is that the schedule must be constructed in real-time, even though nothing is known about a job until its release time.

Competitive analysis [6, 10], with its roots in [12], is a well-studied approach for analyzing online algorithms by comparing them against the optimal offline algorithm, which has full knowledge of the input at the beginning of its execution. One interpretation of this approach is as a game between the designer of the online algorithm and an adversary. First, the designer selects the online algorithm. Then, the adversary observes the algorithm and selects the sequence of jobs that maximizes the competitive ratio: the ratio of the value of the jobs completed by an optimal offline algorithm to the value of those completed by the online algorithm.

Two papers paint a complete picture in terms of competitive analysis for this setting, in which the algorithm is assumed to know k, the maximum ratio between the value densities (value divided by processing time) of any two jobs. For $k = 1$, [4] presents a 4-competitive algorithm, and proves that this is a lower bound on the competitive ratio for deterministic algorithms. The same paper also generalizes the lower bound to $(1 + \sqrt{k})^2$ for any $k \geq 1$, and [15] then presents a matching $(1 + \sqrt{k})^2$-competitive algorithm.

The setting addressed by these papers is completely non-strategic, and the algorithm is assumed to always know the true characteristics of each job upon its release. However, in domains such as grid computing (see, for example, [7, 8]) this assumption is invalid, because "buyers" of processor time choose when and how to submit their jobs. Furthermore, "sellers" not only schedule jobs but also determine the amount that they charge buyers, an issue not addressed in the non-strategic setting.

Thus, we consider an extension of the setting in which each job is owned by a separate, self-interested agent. Instead of being released to the algorithm, each job is now released only to its owning agent. Each agent now has four different ways in which it can manipulate the algorithm: it decides when to submit the job to the algorithm after the true release time, it can artificially inflate the length of the job, and it can declare an arbitrary value and deadline for the job. Because the agents are self-interested, they will choose to manipulate the algorithm if doing so will cause their job to be completed; and, indeed, one can find examples in which agents have incentive to manipulate the algorithms presented in [4] and [15].

The addition of self-interested agents moves the problem from the area of algorithm design to that of mechanism design [17], the science of crafting protocols for self-interested agents. Recent years have seen much activity at the interface of computer science and mechanism design (see, e.g., [9, 18, 19]). In general, a mechanism defines a protocol for interaction between the agents and the center that culminates with the selection of an outcome. In our setting, a mechanism will take as input a job from each agent, and return a schedule for the jobs, and a payment to be made by each agent to the center. A basic solution concept of mechanism design is incentive compatibility, which, in our setting, requires that it is always in each agent's best interests to immediately submit its job upon release, and to truthfully declare its value, length, and deadline.

In order to evaluate a mechanism using competitive analysis, the adversary model must be updated. In the new model, the adversary still determines the sequence of jobs, but it is the self-interested agents who determine the observed input of the mechanism. Thus, in order to achieve a competitive ratio of c, an online mechanism must both be incentive compatible, and always achieve at least $\frac{1}{c}$ of the value that the optimal offline mechanism achieves on the same sequence of jobs.

The rest of the paper is structured as follows. In Section 2, we formally define and review results from the original, non-strategic setting. After introducing the incentive issues through an example, we formalize the mechanism design setting in Section 3. In Section 4 we present our first main result, a $((1+\sqrt{k})^2+1)$-competitive mechanism, and formally prove incentive compatibility and the competitive ratio. We also show how we can simplify this mechanism for the special case in which $k=1$ and each agent cannot alter the length of its job. Returning the general setting, we show in Section 5 that this competitive ratio is a lower bound for deterministic mechanisms that do not pay agents. Finally, in Section 6, we discuss related work other than the directly relevant [4] and [15], before concluding with Section 7.

2. NON-STRATEGIC SETTING

In this section, we formally define the original, non-strategic setting, and recap previous results.

2.1 Formulation

There exists a single processor on which jobs can execute, and N jobs, although this number is not known beforehand. Each job i is characterized by a tuple $\theta_i = (r_i, d_i, l_i, v_i)$, which denotes the release time, deadline, length of processing time required, and value, respectively. The space Θ_i of possible tuples is the same for each job and consists of all θ_i such that $r_i, d_i, l_i, v_i \in \Re_+$ (thus, the model of time is continuous). Each job is released at time r_i, at which point its three other characteristics are known. Nothing is known about the job before its arrival. Each deadline is firm (or, hard), which means that no value is obtained for a job that is completed after its deadline. Preemption of jobs is allowed, and it takes no time to switch between jobs. Thus, job i is completed if and only if the total time it executes on the processor before d_i is at least l_i.

Let $\theta = (\theta_1, \ldots, \theta_N)$ denote the vector of tuples for all jobs, and let $\theta_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$ denote the same vector without the tuple for job i. Thus, (θ_i, θ_{-i}) denotes a complete vector of tuples.

Define the *value density* $\rho_i = \frac{v_i}{l_i}$ of job i to be the ratio of its value to its length. For an input θ, denote the maximum and minimum value densities as $\rho_{min} = \min_i \rho_i$ and $\rho_{max} = \max_i \rho_i$. The *importance ratio* is then defined to be $\frac{\rho_{max}}{\rho_{min}}$, the maximal ratio of value densities between two jobs. The algorithm is assumed to always know an upper bound k on the importance ratio. For simplicity, we normalize the range of possible value densities so that $\rho_{min} = 1$.

An online algorithm is a function $f : \Theta_1 \times \ldots \times \Theta_N \to O$ that maps the vector of tuples (for any number N) to an outcome o. An outcome $o \in O$ is simply a schedule of jobs on the processor, recorded by the function $\mathcal{S} : \Re_+ \to \{0, 1, \ldots, N\}$, which maps each point in time to the active job, or to 0 if the processor is idle.

To denote the total elapsed time that a job has spent on the processor at time t, we will use the function $e_i(t) = \int_0^t \mu(\mathcal{S}(x) = i)dx$, where $\mu(\cdot)$ is an indicator function that returns 1 if the argument is true, and zero otherwise. A job's *laxity* at time t is defined to be $\big(d_i - t - l_i + e_i(t)\big)$, the amount of time that it can remain inactive and still be completed by its deadline. A job is *abandoned* if it cannot be completed by its deadline (formally, if $d_i - t + e_i(t) < l_i$). Also, overload $\mathcal{S}(\cdot)$ and $e_i(\cdot)$ so that they can also take a vector θ as an argument. For example, $\mathcal{S}(\theta, t)$ is shorthand for the $\mathcal{S}(t)$ of the outcome $f(\theta)$, and it denotes the active job at time t when the input is θ.

Since a job cannot be executed before its release time, the space of possible outcomes is restricted in that $\mathcal{S}(\theta, t) = i$ implies $r_i \leq t$. Also, because the online algorithm must produce the schedule over time, without knowledge of future inputs, it must make the same decision at time t for inputs that are indistinguishable at this time. Formally, let $\theta(t)$ denote the subset of the tuples in θ that satisfy $r_i \leq t$. The constraint is then that $\theta(t) = \theta'(t)$ implies $\mathcal{S}(\theta, t) = \mathcal{S}(\theta', t)$.

The objective function is the sum of the values of the jobs that are completed by their respective deadlines: $W(o, \theta) = \sum_i \big(v_i \cdot \mu(e_i(\theta, d_i) \geq l_i)\big)$. Let $W^*(\theta) = \max_{o \in O} W(o, \theta)$ denote the maximum possible total value for the profile θ.

In competitive analysis, an online algorithm is evaluated by comparing it against an optimal offline algorithm. Because the offline algorithm knows the entire input θ at time 0 (but still cannot start each job i until time r_i), it always achieves $W^*(\theta)$. An online algorithm $f(\cdot)$ is (strictly) *c-competitive* if there does not exist an input θ such that $c \cdot W(f(\theta), \theta) < W^*(\theta)$. An algorithm that is c-competitive is also said to achieve a *competitive ratio* of c.

We assume that there does not exist an overload period of infinite duration. A period of time $[t^s, t^f]$ is overloaded if the sum of the lengths of the jobs whose release time and deadline both fall within the time period exceeds the duration of the interval (formally, if $t^f - t^s \leq \sum_{i|(t^s \leq r_i, d_i \leq t^f)} l_i$). Without such an assumption, it is not possible to achieve a finite competitive ratio [15].

2.2 Previous Results

In the non-strategic setting, [4] presents a 4-competitive algorithm called TD_1 (version 2) for the case of $k = 1$, while [15] presents a $(1+\sqrt{k})^2$-competitive algorithm called D^{over} for the general case of $k \geq 1$. Matching lower bounds for deterministic algorithms for both of these cases were shown in [4]. In this section we provide a high-level description of TD_1 (version 2) using an example.

TD_1 (version 2) divides the schedule into intervals, each of which begins when the processor transitions from idle to busy (call this time t^b), and ends with the completion of a job. The first active job of an interval may have laxity; however, for the remainder of the interval, preemption of the active job is only considered when some other job has zero laxity. For example, when the input is the set of jobs listed in Table 1, the first interval is the complete execution of job 1 over the range $[0.0, 0.9]$. No preemption is considered during this interval, because job 2 has laxity until time 1.5. Then, a new interval starts at $t^b = 0.9$ when job 2 becomes active. Before job 2 can finish, preemption is considered at time 4.8, when job 3 is released with zero laxity.

In order to decide whether to preempt the active job, TD_1 (version 2) uses two more variables: t^e and p_loss. The former records the latest deadline of a job that would be abandoned if the active job executes to completion (or, if no such job exists, the time that the active job will finish if it is not preempted). In this case, $t^e = 17.0$. The value $t^e - t^b$ represents the an upper bound on the amount of possible execution time "lost" to the optimal offline algorithm due to the completion of the active job. The other variable, p_loss, is equal to the length of the first active job of the current interval. Because in general this job could have laxity, the offline algorithm may be able to complete it outside of the range $[t^b, t^e]$.[1] If the algorithm completes the active job and this job's length is at least $\frac{t^e - t^b + p\_loss}{4}$, then the algorithm is guaranteed to be 4-competitive for this interval (note that $k = 1$ implies that all jobs have the same value density and thus that lengths can used to compute the competitive ratio). Because this is not case at time 4.8 (since $\frac{t^e - t^b + p\_loss}{4} = \frac{17.0 - 0.9 + 4.0}{4} > 4.0 = l_2$), the algorithm preempts job 2 for job 3, which then executes to completion.

Job	r_i	d_i	l_i	v_i
1	0.0	0.9	0.9	0.9
2	0.5	5.5	4.0	4.0
3	4.8	17.0	12.2	12.2

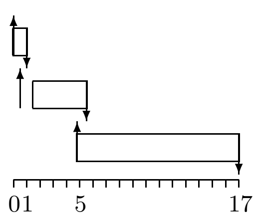

Table 1: Input used to recap TD_1 (version 2) [4]. The up and down arrows represent r_i and d_i, respectively, while the length of the box equals l_i.

3. MECHANISM DESIGN SETTING

However, false information about job 2 would cause TD_1 (version 2) to complete this job. For example, if job 2's deadline were declared as $\hat{d}_2 = 4.7$, then it would have zero laxity at time 0.7. At this time, the algorithm would preempt job 1 for job 2, because $\frac{t^e - t^b + p\_loss}{4} = \frac{4.7 - 0.0 + 1.0}{4} > 0.9 = l_1$. Job 2 would then complete before the arrival of job 3.[2]

In order to address incentive issues such as this one, we need to formalize the setting as a mechanism design problem. In this section we first present the mechanism design formulation, and then define our goals for the mechanism.

3.1 Formulation

There exists a center, who controls the processor, and N agents, where the value of N is unknown by the center beforehand. Each job i is owned by a separate agent i. The characteristics of the job define the agent's type $\theta_i \in \Theta_i$. At time r_i, agent i privately observes its type θ_i, and has no information about job i before r_i. Thus, jobs are still released over time, but now each job is revealed only to the owning agent.

Agents interact with the center through a direct mechanism $\Gamma = (\Theta_1, \ldots, \Theta_N, g(\cdot))$, in which each agent declares a job, denoted by $\hat{\theta}_i = (\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}_i)$, and $g : \Theta_1 \times \ldots \times \Theta_N \to O$ maps the declared types to an outcome $o \in O$. An outcome $o = (\mathcal{S}(\cdot), p_1, \ldots, p_N)$ consists of a schedule and a payment from each agent to the mechanism.

In a standard mechanism design setting, the outcome is enforced at the end of the mechanism. However, since the end is not well-defined in this online setting, we choose to model returning the job if it is completed and collecting a payment from each agent i as occurring at $\hat{d}_i$, which, according to the agent's declaration, is the latest relevant point of time for that agent. That is, even if job i is completed before $\hat{d}_i$, the center does not return the job to agent i until that time. This modelling decision could instead be viewed as a decision by the mechanism designer from a larger space of possible mechanisms. Indeed, as we will discuss later, this decision of when to return a completed job is crucial to our mechanism.

Each agent's utility, $u_i(g(\hat{\theta}), \theta_i) = v_i \cdot \mu(e_i(\hat{\theta}, d_i) \geq l_i) \cdot \mu(\hat{d}_i \leq d_i) - p_i(\hat{\theta})$, is a quasi-linear function of its value for its job (if completed and returned by its true deadline) and the payment it makes to the center. We assume that each agent is a rational, expected utility maximizer.

Agent declarations are restricted in that an agent cannot declare a length shorter than the true length, since the center would be able to detect such a lie if the job were completed. On the other hand, in the general formulation we will allow agents to declare longer lengths, since in some settings it may be possible add unnecessary work to a job. However, we will also consider a restricted formulation in which this type of lie is not possible. The declared release time $\hat{r}_i$ is the time that the agent chooses to submit job i to the center, and it cannot precede the time r_i at which the job is revealed to the agent. The agent can declare an arbitrary deadline or value. To summarize, agent i can declare any type $\hat{\theta}_i = (\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}_i)$ such that $\hat{l}_i \geq l_i$ and $\hat{r}_i \geq r_i$.

While in the non-strategic setting it was sufficient for the algorithm to know the upper bound k on the ratio $\frac{\rho_{max}}{\rho_{min}}$, in the mechanism design setting we will strengthen this assumption so that the mechanism also knows ρ_{min} (or, equivalently, the range $[\rho_{min}, \rho_{max}]$ of possible value densities).[3]

[1]While it would be easy to alter the algorithm to recognize that this is not possible for the jobs in Table 1, our example does not depend on the use of p_loss.

[2]While we will not describe the significantly more complex D^{over}, we note that it is similar in its use of intervals and its preference for the active job. Also, we note that the lower bound we will show in Section 5 implies that false information can also benefit a job in D^{over}.

[3]Note that we could then force agent declarations to satisfy $\rho_{min} \leq \frac{\hat{v}_i}{\hat{l}_i} \leq \rho_{max}$. However, this restriction would not

While we feel that it is unlikely that a center would know k without knowing this range, we later present a mechanism that does not depend on this extra knowledge in a restricted setting.

The restriction on the schedule is now that $\mathcal{S}(\hat{\theta}, t) = i$ implies $\hat{r}_i \leq t$, to capture the fact that a job cannot be scheduled on the processor before it is declared to the mechanism. As before, preemption of jobs is allowed, and job switching takes no time.

The constraints due to the online mechanism's lack of knowledge of the future are that $\hat{\theta}(t) = \hat{\theta}'(t)$ implies $\mathcal{S}(\hat{\theta}, t) = \mathcal{S}(\hat{\theta}', t)$, and $\hat{\theta}(\hat{d}_i) = \hat{\theta}'(\hat{d}_i)$ implies $p_i(\hat{\theta}) = p_i(\hat{\theta}')$ for each agent i. The setting can then be summarized as follows.

Overview of the Setting:

for all t **do**
  The center instantiates $\mathcal{S}(\hat{\theta}, t) \leftarrow i$, for some i s.t. $\hat{r}_i \leq t$
  if $\exists i, (r_i = t)$ **then**
    θ_i is revealed to agent i
  if $\exists i, (t \geq r_i)$ and agent i has not declared a job **then**
    Agent i can declare any job $\hat{\theta}_i$, s.t. $\hat{r}_i = t$ and $\hat{l}_i \geq l_i$
  if $\exists i, (\hat{d}_i = t) \wedge (e_i(\hat{\theta}, t) \geq l_i)$ **then**
    Completed job i is returned to agent i
  if $\exists i, (\hat{d}_i = t)$ **then**
    Center sets and collects payment $p_i(\hat{\theta})$ from agent i

3.2 Mechanism Goals

Our aim as mechanism designer is to maximize the value of completed jobs, subject to the constraints of incentive compatibility and individual rationality.

The condition for (dominant strategy) incentive compatibility is that for each agent i, regardless of its true type and of the declared types of all other agents, agent i cannot increase its utility by unilaterally changing its declaration.

Definition 1. *A direct mechanism* Γ *satisfies incentive compatibility (IC) if* $\forall i, \theta_i, \theta'_i, \hat{\theta}_{-i}$:

$$u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq u_i(g(\theta'_i, \hat{\theta}_{-i}), \theta_i)$$

From an agent perspective, dominant strategies are desirable because the agent does not have to reason about either the strategies of the other agents or the distribution from the which other agent's types are drawn. From a mechanism designer perspective, dominant strategies are important because we can reasonably assume that an agent who has a dominant strategy will play according to it. For these reasons, in this paper we require dominant strategies, as opposed to a weaker equilibrium concept such as Bayes-Nash, under which we could improve upon our positive results.[4]

decrease the lower bound on the competitive ratio.

[4]A possible argument against the need for incentive compatibility is that an agent's lie may actually improve the schedule. In fact, this was the case in the example we showed for the false declaration $\hat{d}_2 = 4.7$. However, if an agent lies due to incorrect beliefs over the future input, then the lie could instead make the schedule the worse (for example, if job 3 were never released, then job 1 would have been unnecessarily abandoned). Furthermore, if we do not know the beliefs of the agents, and thus cannot predict how they will lie, then we can no longer provide a competitive guarantee for our mechanism.

While restricting ourselves to incentive compatible direct mechanisms may seem limiting at first, the *Revelation Principle for Dominant Strategies* (see, e.g., [17]) tells us that if our goal is dominant strategy implementation, then we can make this restriction without loss of generality.

The second goal for our mechanism, individual rationality, requires that agents who truthfully reveal their type never have negative utility. The rationale behind this goal is that participation in the mechanism is assumed to be voluntary.

Definition 2. *A direct mechanism* Γ *satisfies individual rationality (IR) if* $\forall i, \theta_i, \hat{\theta}_{-i},\ u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq 0$.

Finally, the social welfare function that we aim to maximize is the same as the objective function of the non-strategic setting: $W(o, \theta) = \sum_i \big(v_i \cdot \mu(e_i(\theta, d_i) \geq l_i)\big)$. As in the non-strategic setting, we will evaluate an online mechanism using competitive analysis to compare it against an optimal offline mechanism (which we will denote by $\Gamma_{offline}$). An offline mechanism knows all of the types at time 0, and thus can always achieve $W^*(\theta)$.[5]

Definition 3. *An online mechanism* Γ *is (strictly) c-competitive if it satisfies IC and IR, and if there does not exist a profile of agent types* θ *such that* $c \cdot W(g(\theta), \theta) < W^*(\theta)$.

4. RESULTS

In this section, we first present our main positive result: a $\big((1+\sqrt{k})^2+1\big)$-competitive mechanism ($\Gamma_1$). After providing some intuition as to why Γ_1 satisfies individual rationality and incentive compatibility, we formally prove first these two properties and then the competitive ratio. We then consider a special case in which $k = 1$ and agents cannot lie about the length of their job, which allows us to alter this mechanism so that it no longer requires either knowledge of ρ_{min} or the collection of payments from agents.

Unlike TD_1 (version 2) and D^{over}, Γ_1 gives no preference to the active job. Instead, it always executes the available job with the highest *priority*: $(\hat{v}_i + \sqrt{k} \cdot e_i(\hat{\theta}, t) \cdot \rho_{min})$. Each agent whose job is completed is then charged the lowest value that it could have declared such that its job still would have been completed, holding constant the rest of its declaration.

By the use of a payment rule similar to that of a second-price auction, Γ_1 satisfies both IC with respect to values and IR. We now argue why it satisfies IC with respect to the other three characteristics. Declaring an "improved" job (i.e., declaring an earlier release time, a shorter length, or a later deadline) could possibly decrease the payment of an agent. However, the first two lies are not possible in our setting, while the third would cause the job, if it is completed, to be returned to the agent after the true deadline. This is the reason why it is important to always return a completed job at its declared deadline, instead of at the point at which it is completed.

[5]Another possibility is to allow only the agents to know their types at time 0, and to force $\Gamma_{offline}$ to be incentive compatible so that agents will truthfully declare their types at time 0. However, this would not affect our results, since executing a VCG mechanism (see, e.g., [17]) at time 0 both satisfies incentive compatibility and always maximizes social welfare.

Mechanism 1 Γ_1

Execute $\mathcal{S}(\hat{\theta}, \cdot)$ according to Algorithm 1
for all i **do**
  if $e_i(\hat{\theta}, \hat{d}_i) \geq \hat{l}_i$ $\{$*Agent i's job is completed*$\}$ **then**
    $p_i(\hat{\theta}) \leftarrow \arg\min_{v_i' \geq 0}(e_i(((\hat{r}_i, \hat{d}_i, \hat{l}_i, v_i'), \hat{\theta}_{-i}), \hat{d}_i) \geq \hat{l}_i)$
  else
    $p_i(\hat{\theta}) \leftarrow 0$

Algorithm 1

for all t **do**
  $Avail \leftarrow \{i | (t \geq \hat{r}_i) \wedge (e_i(\hat{\theta}, t) < \hat{l}_i) \wedge (e_i(\hat{\theta}, t) + \hat{d}_i - t \geq \hat{l}_i)\}$
    $\{$*Set of all released, non-completed, non-abandoned jobs*$\}$
  if $Avail \neq \emptyset$ **then**
    $\mathcal{S}(\hat{\theta}, t) \leftarrow \arg\max_{i \in Avail}(\hat{v}_i + \sqrt{k} \cdot e_i(\hat{\theta}, t) \cdot \rho_{min})$
      $\{$*Break ties in favor of lower* $\hat{r}_i\}$
  else
    $\mathcal{S}(\hat{\theta}, t) \leftarrow 0$

It remains to argue why an agent does not have incentive to "worsen" its job. The only possible effects of an inflated length are delaying the completion of the job and causing it to be abandoned, and the only possible effects of an earlier declared deadline are causing to be abandoned and causing it to be returned earlier (which has no effect on the agent's utility in our setting). On the other hand, it is less obvious why agents do not have incentive to declare a later release time. Consider a mechanism Γ_1' that differs from Γ_1 in that it does not preempt the active job i unless there exists another job j such that $(\hat{v}_i + \sqrt{k} \cdot l_i(\hat{\theta}, t) \cdot \rho_{min}) < \hat{v}_j$. Note that as an active job approaches completion in Γ_1, its condition for preemption approaches that of Γ_1'.

However, the types in Table 2 for the case of $k = 1$ show why an agent may have incentive to delay the arrival of its job under Γ_1'. Job 1 becomes active at time 0, and job 2 is abandoned upon its release at time 6, because $10 + 10 = v_1 + l_1 > v_2 = 13$. Then, at time 8, job 1 is preempted by job 3, because $10 + 10 = v_1 + l_1 < v_3 = 22$. Job 3 then executes to completion, forcing job 1 to be abandoned. However, job 2 had more "weight" than job 1, and would have prevented job 3 from being executed if it had been the active job at time 8, since $13 + 13 = v_2 + l_2 > v_3 = 22$. Thus, if agent 1 had falsely declared $\hat{r}_1 = 20$, then job 3 would have been abandoned at time 8, and job 1 would have completed over the range $[20, 30]$.

Job	r_i	d_i	l_i	v_i
1	0	30	10	10
2	6	19	13	13
3	8	30	22	22

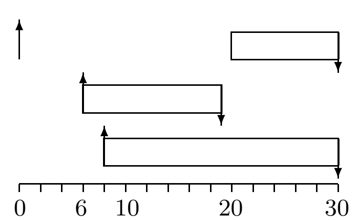

Table 2: Jobs used to show why a slightly altered version of Γ_1 would not be incentive compatible with respect to release times.

Intuitively, Γ_1 avoids this problem because of two properties. First, when a job becomes active, it must have a greater priority than all other available jobs. Second, because a job's priority can only increase through the increase of its elapsed time, $e_i(\hat{\theta}, t)$, the rate of increase of a job's priority is independent of its characteristics. These two properties together imply that, while a job is active, there cannot exist a time at which its priority is less than the priority that one of these other jobs would have achieved by executing on the processor instead.

4.1 Proof of Individual Rationality and Incentive Compatibility

After presenting the (trivial) proof of IR, we break the proof of IC into lemmas.

THEOREM 1. *Mechanism* Γ_1 *satisfies individual rationality.*

PROOF. For arbitrary $i, \theta_i, \hat{\theta}_{-i}$, if job i is not completed, then agent i pays nothing and thus has a utility of zero; that is, $p_i(\theta_i, \hat{\theta}_{-i}) = 0$ and $u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) = 0$. On the other hand, if job i is completed, then its value must exceed agent i's payment. Formally, $u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) = v_i - \arg\min_{v_i' \geq 0}(e_i(((r_i, d_i, l_i, v_i'), \hat{\theta}_{-i}), d_i) \geq l_i) \geq 0$ must hold, since $v_i' = v_i$ satisfies the condition. $\square$

To prove IC, we need to show that for an arbitrary agent i, and an arbitrary profile $\hat{\theta}_{-i}$ of declarations of the other agents, agent i can never gain by making a false declaration $\hat{\theta}_i \neq \theta_i$, subject to the constraints that $\hat{r}_i \geq r_i$ and $\hat{l}_i \geq l_i$.

We start by showing that, regardless of $\hat{v}_i$, if truthful declarations of r_i, d_i, and l_i do not cause job i to be completed, then "worse" declarations of these variables (that is, declarations that satisfy $\hat{r}_i \geq r_i$, $\hat{l}_i \geq l_i$ and $\hat{d}_i \leq d_i$) can never cause the job to be completed. We break this part of the proof into two lemmas, first showing that it holds for the release time, regardless of the declarations of the other variables, and then for length and deadline.

LEMMA 2. *In mechanism* Γ_1*, the following condition holds for all* $i, \theta_i, \hat{\theta}_{-i}$*:* $\forall\ \hat{v}_i,\ \hat{l}_i \geq l_i,\ \hat{d}_i \leq d_i,\ \hat{r}_i \geq r_i$,

$$\left[e_i\big(((\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq \hat{l}_i\right] \implies \left[e_i\big(((r_i, \hat{d}_i, \hat{l}_i, \hat{v}_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq \hat{l}_i\right]$$

PROOF. Assume by contradiction that this condition does not hold– that is, job i is not completed when r_i is truthfully declared, but is completed for some false declaration $\hat{r}_i \geq r_i$. We first analyze the case in which the release time is truthfully declared, and then we show that job i cannot be completed when agent i delays submitting it to the center.

Case I: Agent i declares $\hat{\theta}_i' = (r_i, \hat{d}_i, \hat{l}_i, \hat{v}_i)$.

First, define the following three points in the execution of job i.

- Let $t^s = \arg\min_t \big(\mathcal{S}((\hat{\theta}_i', \hat{\theta}_{-i}), t) = i\big)$ be the time that job i first starts execution.
- Let $t^p = \arg\min_{t > t^s} \big(\mathcal{S}((\hat{\theta}_i', \hat{\theta}_{-i}), t) \neq i\big)$ be the time that job i is first preempted.
- Let $t^a = \arg\min_t \big(e_i((\hat{\theta}_i', \hat{\theta}_{-i}), t) + \hat{d}_i - t < \hat{l}_i\big)$ be the time that job i is abandoned.

If t^s and t^p are undefined because job i never becomes active, then let $t^s = t^p = t^a$.

Also, partition the jobs declared by other agents before t^a into the following three sets.

- $X = \{j | (\hat{r}_j < t^p) \wedge (j \neq i)\}$ consists of the jobs (other than i) that arrive before job i is first preempted.
- $Y = \{j | (t^p \leq \hat{r}_j \leq t^a) \wedge (\hat{v}_j > \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), \hat{r}_j))\}$ consists of the jobs that arrive in the range $[t^p, t^a]$ and that when they arrive have higher priority than job i (note that we are make use of the normalization).
- $Z = \{j | (t^p \leq \hat{r}_j \leq t^a) \wedge (\hat{v}_j \leq \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), \hat{r}_j))\}$ consists of the jobs that arrive in the range $[t^p, t^a]$ and that when they arrive have lower priority than job i.

We now show that all active jobs during the range $(t^p, t^a]$ must be either i or in the set Y. Unless $t^p = t^a$ (in which case this property trivially holds), it must be the case that job i has a higher priority than an arbitrary job $x \in X$ at time t^p, since at the time just preceding t^p job x was available and job i was active. Formally, $\hat{v}_x + \sqrt{k} \cdot e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^p) < \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^p)$ must hold.[6] We can then show that, over the range $[t^p, t^a]$, no job $x \in X$ runs on the processor. Assume by contradiction that this is not true. Let $t^f \in [t^p, t^a]$ be the earliest time in this range that some job $x \in X$ is active, which implies that $e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^f) = e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^p)$. We can then show that job i has a higher priority at time t^f as follows: $\hat{v}_x + \sqrt{k} \cdot e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^f) = \hat{v}_x + \sqrt{k} \cdot e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^p) < \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^p) \leq \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^f)$, contradicting the fact that job x is active at time t^f.

A similar argument applies to an arbitrary job $z \in Z$, starting at it release time $\hat{r}_z > t^p$, since by definition job i has a higher priority at that time. The only remaining jobs that can be active over the range $(t^p, t^a]$ are i and those in the set Y.

Case II: Agent i declares $\hat{\theta}_i = (\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}_i)$, where $\hat{r}_i > r_i$.

We now show that job i cannot be completed in this case, given that it was not completed in case I. First, we can restrict the range of $\hat{r}_i$ that we need to consider as follows. Declaring $\hat{r}_i \in (r_i, t^s]$ would not affect the schedule, since t^s would still be the first time that job i executes. Also, declaring $\hat{r}_i > t^a$ could not cause the job to be completed, since $d_i - t^a < \hat{l}_i$ holds, which implies that job i would be abandoned at its release. Thus, we can restrict consideration to $\hat{r}_i \in (t^s, t^a]$.

In order for declaring $\hat{\theta}_i$ to cause job i to be completed, a necessary condition is that the execution of some job $y^c \in Y$ must change during the range $(t^p, t^a]$, since the only jobs other than i that are active during that range are in Y. Let $t^c = \arg\min_{t \in (t^p, t^a]} [\exists y^c \in Y, (\mathcal{S}((\hat{\theta}'_i, \hat{\theta}_{-i}), t) = y^c) \wedge (\mathcal{S}((\hat{\theta}_i, \hat{\theta}_{-i}), t) \neq y^c)]$ be the first time that such a change occurs. We will now show that for any $\hat{r}_i \in (t^s, t^a]$, there cannot exist a job with higher priority than y^c at time t^c, contradicting $(\mathcal{S}((\hat{\theta}_i, \hat{\theta}_{-i}), t) \neq y^c)$.

First note that job i cannot have a higher priority, since there would have to exist a $t \in (t^p, t^c)$ such that $\exists y \in Y, (\mathcal{S}((\hat{\theta}'_i, \hat{\theta}_{-i}), t) = y) \wedge (\mathcal{S}((\hat{\theta}_i, \hat{\theta}_{-i}), t) = i)$, contradicting the definition of t^c.

Now consider an arbitrary $y \in Y$ such that $y \neq y^c$. In case I, we know that job y has lower priority than y^c at time t^c; that is, $\hat{v}_y + \sqrt{k} \cdot e_y((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c) < \hat{v}_{y^c} + \sqrt{k} \cdot e_{y^c}((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c)$. Thus, moving to case II, job y must replace some other job before t^c. Since $\hat{r}_y \geq t^p$, the condition is that there must exist some $t \in (t^p, t^c)$ such that $\exists w \in Y \cup \{i\}, (\mathcal{S}((\hat{\theta}'_i, \hat{\theta}_{-i}), t) = w) \wedge (\mathcal{S}((\hat{\theta}_i, \hat{\theta}_{-i}), t) = y)$. Since $w \in Y$ would contradict the definition of t^c, we know that $w = i$. That is, the job that y replaces must be i. By definition of the set Y, we know that $\hat{v}_y > \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), \hat{r}_y)$. Thus, if $\hat{r}_y \leq t$, then job i could not have executed instead of y in case I. On the other hand, if $\hat{r}_y > t$, then job y obviously could not execute at time t, contradicting the existence of such a time t.

Now consider an arbitrary job $x \in X$. We know that in case I job i has a higher priority than job x at time t^s, or, formally, that $\hat{v}_x + \sqrt{k} \cdot e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^s) < \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^s)$. We also know that $\hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c) < \hat{v}_{y^c} + \sqrt{k} \cdot e_{y^c}((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c)$. Since delaying i's arrival will not affect the execution up to time t^s, and since job x cannot execute instead of a job $y \in Y$ at any time $t \in (t^p, t^c]$ by definition of t^c, the only way for job x's priority to increase before t^c as we move from case I to II is to replace job i over the range $(t^s, t^c]$. Thus, an upper bound on job x's priority when agent i declares $\hat{\theta}_i$ is: $\hat{v}_x + \sqrt{k} \cdot \left[e_x((\hat{\theta}'_i, \hat{\theta}_{-i}), t^s) + e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c) - e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^s)\right] < \hat{v}_i + \sqrt{k} \cdot \left[e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^s) + e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c) - e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^s)\right] = \hat{v}_i + \sqrt{k} \cdot e_i((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c) < \hat{v}_{y^c} + \sqrt{k} \cdot e_{y^c}((\hat{\theta}'_i, \hat{\theta}_{-i}), t^c)$.

Thus, even at this upper bound, job y^c would execute instead of job x at time t^c. A similar argument applies to an arbitrary job $z \in Z$, starting at it release time $\hat{r}_z$. Since the sets $\{i\}, X, Y, Z$ partition the set of jobs released before t^a, we have shown that no job could execute instead of job y^c, contradicting the existence of t^c, and completing the proof. $\square$

LEMMA 3. *In mechanism Γ_1, the following condition holds for all $i, \theta_i, \hat{\theta}_{-i}$:* $\forall\ \hat{v}_i,\ \hat{l}_i \geq l_i,\ \hat{d}_i \leq d_i,$

$$\left[e_i\big(((r_i, \hat{d}_i, \hat{l}_i, \hat{v}_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq \hat{l}_i\right] \implies \left[e_i\big(((r_i, d_i, l_i, \hat{v}_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq l_i\right]$$

PROOF. Assume by contradiction there exists some instantiation of the above variables such that job i is not completed when l_i and d_i are truthfully declared, but is completed for some pair of false declarations $\hat{l}_i \geq l_i$ and $\hat{d}_i \leq d_i$.

Note that the only effect that $\hat{d}_i$ and $\hat{l}_i$ have on the execution of the algorithm is on whether or not $i \in Avail$. Specifically, they affect the two conditions: $(e_i(\hat{\theta}, t) < \hat{l}_i)$ and $(e_i(\hat{\theta}, t) + \hat{d}_i - t \geq \hat{l}_i)$. Because job i is completed when $\hat{l}_i$ and $\hat{d}_i$ are declared, the former condition (for completion) must become false before the latter. Since truthfully declaring $l_i \leq \hat{l}_i$ and $d_i \geq \hat{d}_i$ will only make the former condition become false earlier and the latter condition become false later, the execution of the algorithm will not be affected when moving to truthful declarations, and job i will be completed, a contradiction. $\square$

We now use these two lemmas to show that the payment for a completed job can only increase by falsely declaring "worse" $\hat{l}_i$, $\hat{d}_i$, and $\hat{r}_i$.

[6] For simplicity, when we give the formal condition for a job x to have a higher priority than another job y, we will assume that job x's priority is strictly greater than job y's, because, in the case of a tie that favors x, future ties would also be broken in favor of job x.

LEMMA 4. *In mechanism Γ_1, the following condition holds for all $i, \theta_i, \hat{\theta}_{-i}$:* $\quad \forall\, \hat{l}_i \geq l_i,\ \hat{d}_i \leq d_i,\ \hat{r}_i \geq r_i$,

$$\arg\min_{v'_i \geq 0} \left[e_i\big(((\hat{r}_i, \hat{d}_i, \hat{l}_i, v'_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq \hat{l}_i\right] \quad \geq$$
$$\arg\min_{v'_i \geq 0} \left[e_i\big(((r_i, d_i, l_i, v'_i), \hat{\theta}_{-i}), d_i\big) \geq l_i\right]$$

PROOF. Assume by contradiction that this condition does not hold. This implies that there exists some value v'_i such that the condition $(e_i(((\hat{r}_i, \hat{d}_i, \hat{l}_i, v'_i), \hat{\theta}_{-i}), \hat{d}_i) \geq \hat{l}_i)$ holds, but $(e_i(((r_i, d_i, l_i, v'_i), \hat{\theta}_{-i}), d_i) \geq l_i)$ does not. Applying Lemmas 2 and 3: $(e_i(((\hat{r}_i, \hat{d}_i, \hat{l}_i, v'_i), \hat{\theta}_{-i}), \hat{d}_i) \geq \hat{l}_i) \implies$ $(e_i(((r_i, \hat{d}_i, \hat{l}_i, v'_i), \hat{\theta}_{-i}), \hat{d}_i) \geq \hat{l}_i) \implies$
$(e_i(((r_i, d_i, l_i, v'_i), \hat{\theta}_{-i}), d_i) \geq l_i)$, a contradiction. $\square$

Finally, the following lemma tells us that the completion of a job is monotonic in its declared value.

LEMMA 5. *In mechanism Γ_1, the following condition holds for all $i, \hat{\theta}_i, \hat{\theta}_{-i}$:* $\quad \forall\, \hat{v}'_i \geq \hat{v}_i$,

$$\left[e_i\big(((\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq \hat{l}_i\right] \quad \implies$$
$$\left[e_i\big(((\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}'_i), \hat{\theta}_{-i}), \hat{d}_i\big) \geq \hat{l}_i\right]$$

The proof, by contradiction, of this lemma is omitted because it is essentially identical to that of Lemma 2 for $\hat{r}_i$. In case I, agent i declares $(\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}'_i)$ and the job is not completed, while in case II he declares $(\hat{r}_i, \hat{d}_i, \hat{l}_i, \hat{v}_i)$ and the job is completed. The analysis of the two cases then proceeds as before– the execution will not change up to time t^s because the initial priority of job i decreases as we move from case I to II; and, as a result, there cannot be a change in the execution of a job other than i over the range $(t^p, t^a]$.

We can now combine the lemmas to show that no profitable deviation is possible.

THEOREM 6. *Mechanism Γ_1 satisfies incentive compatibility.*

PROOF. For an arbitrary agent i, we know that $\hat{r}_i \geq r_i$ and $\hat{l}_i \geq l_i$ hold by assumption. We also know that agent i has no incentive to declare $\hat{d}_i > d_i$, because job i would never be returned before its true deadline. Then, because the payment function is non-negative, agent i's utility could not exceed zero. By IR, this is the minimum utility it would achieve if it truthfully declared θ_i. Thus, we can restrict consideration to $\hat{\theta}_i$ that satisfy $\hat{r}_i \geq r_i$, $\hat{l}_i \geq l_i$, and $\hat{d}_i \leq d_i$. Again using IR, we can further restrict consideration to $\hat{\theta}_i$ that cause job i to be completed, since any other $\hat{\theta}_i$ yields a utility of zero.

If truthful declaration of θ_i causes job i to be completed, then by Lemma 4 any such false declaration $\hat{\theta}_i$ could not decrease the payment of agent i. On the other hand, if truthful declaration does not cause job i to be completed, then declaring such a $\hat{\theta}_i$ will cause agent i to have negative utility, since $v_i < \arg\min_{v'_i \geq 0} \left[e_i(((r_i, d_i, l_i, v'_i), \hat{\theta}_{-i}), \hat{d}_i) \geq l_i\right] \leq \arg\min_{v'_i \geq 0} \left[e_i(((\hat{r}_i, \hat{d}_i, \hat{l}_i, v'_i), \hat{\theta}_{-i}), \hat{d}_i) \geq \hat{l}_i\right]$ holds by Lemmas 5 and 4, respectively. $\square$

4.2 Proof of Competitive Ratio

The proof of the competitive ratio, which makes use of techniques adapted from those used in [15], is also broken into lemmas. Having shown IC, we can assume truthful declaration ($\hat{\theta} = \theta$). Since we have also shown IR, in order to prove the competitive ratio it remains to bound the loss of social welfare against $\Gamma_{offline}$.

Denote by $(1, 2, \ldots, F)$ the sequence of jobs completed by Γ_1. Divide time into intervals $I_f = (t_f^{open}, t_f^{close}]$, one for each job f in this sequence. Set t_f^{close} to be the time at which job f is completed, and set $t_f^{open} = t_{f-1}^{close}$ for $f \geq 2$, and $t_1^{open} = 0$ for $f = 1$. Also, let t_f^{begin} be the first time that the processor is not idle in interval I_f.

LEMMA 7. *For any interval I_f, the following inequality holds: $t_f^{close} - t_f^{begin} \leq (1 + \frac{1}{\sqrt{k}}) \cdot v_f$*

PROOF. Interval I_f begins with a (possibly zero length) period of time in which the processor is idle because there is no available job. Then, it continuously executes a sequence of jobs $(1, 2, \ldots, c)$, where each job i in this sequence is preempted by job $i+1$, except for job c, which is completed (thus, job c in this sequence is the same as job f is the global sequence of completed jobs). Let t_i^s be the time that job i begins execution. Note that $t_1^s = t_f^{begin}$.

Over the range $[t_f^{begin}, t_f^{close}]$, the priority $(v_i + \sqrt{k} \cdot e_i(\theta, t))$ of the active job is monotonically increasing with time, because this function linearly increases while a job is active, and can only increase at a point in time when preemption occurs. Thus, each job $i > 1$ in this sequence begins execution at its release time (that is, $t_i^s = r_i$), because its priority does not increase while it is not active.

We now show that the value of the completed job c exceeds the product of $\sqrt{k}$ and the time spent in the interval on jobs 1 through $c-1$, or, more formally, that the following condition holds: $v_c \geq \sqrt{k} \sum_{h=1}^{c-1} (e_h(\theta, t_{h+1}^s) - e_h(\theta, t_h^s))$. To show this, we will prove by induction that the stronger condition $v_i \geq \sqrt{k} \sum_{h=1}^{i-1} e_h(\theta, t_{h+1}^s)$ holds for all jobs i in the sequence.

Base Case: For $i = 1$, $v_1 \geq \sqrt{k} \sum_{h=1}^{0} e_h(\theta, t_{h+1}^s) = 0$, since the sum is over zero elements.

Inductive Step: For an arbitrary $1 \leq i < c$, we assume that $v_i \geq \sqrt{k} \sum_{h=1}^{i-1} e_h(\theta, t_{h+1}^s)$ holds. At time t_{i+1}^s, we know that $v_{i+1} \geq v_i + \sqrt{k} \cdot e_i(\theta, t_{i+1}^s)$ holds, because $t_{i+1}^s = r_{i+1}$. These two inequalities together imply that $v_{i+1} \geq \sqrt{k} \sum_{h=1}^{i} e_h(\theta, t_{h+1}^s)$, completing the inductive step.

We also know that $t_f^{close} - t_c^s \leq l_c \leq v_c$ must hold, by the simplifying normalization of $\rho_{min} = 1$ and the fact that job c's execution time cannot exceed its length. We can thus bound the total execution time of I_f by: $t_f^{close} - t_f^{begin} = (t_f^{close} - t_c^s) + \sum_{h=1}^{c-1} (e_h(\theta, t_{h+1}^s) - e_h(\theta, t_h^s)) \leq (1 + \frac{1}{\sqrt{k}}) v_f$. $\square$

We now consider the possible execution of uncompleted jobs by $\Gamma_{offline}$. Associate each job i that is not completed by Γ_1 with the interval during which it was abandoned. All jobs are now associated with an interval, since there are no gaps between the intervals, and since no job i can be abandoned after the close of the last interval at t_F^{close}. Because the processor is idle after t_F^{close}, any such job i would become active at some time $t \geq t_F^{close}$, which would lead to the completion of some job, creating a new interval and contradicting the fact that I_F is the last one.

The following lemma is equivalent to Lemma 5.6 of [15], but the proof is different for our mechanism.

LEMMA 8. *For any interval I_f and any job i abandoned in I_f, the following inequality holds: $v_i \leq (1+\sqrt{k})v_f$.*

PROOF. Assume by contradiction that there exists a job i abandoned in I_f such that $v_i > (1+\sqrt{k})v_f$. At t_f^{close}, the priority of job f is $v_f + \sqrt{k}\cdot l_f < (1+\sqrt{k})v_f$. Because the priority of the active job monotonically increases over the range $[t_f^{begin}, t_f^{close}]$, job i would have a higher priority than the active job (and thus begin execution) at some time $t \in [t_f^{begin}, t_f^{close}]$. Again applying monotonicity, this would imply that the priority of the active job at t_f^{close} exceeds $(1+\sqrt{k})v_f$, contradicting the fact that it is $(1+\sqrt{k})v_f$. □

As in [15], for each interval I_f, we give $\Gamma_{offline}$ the following "gift": k times the amount of time in the range $[t_f^{begin}, t_f^{close}]$ that it does not schedule a job. Additionally, we "give" the adversary v_f, since the adversary may be able to complete this job at some future time, due to the fact that Γ_1 ignores deadlines. The following lemma is Lemma 5.10 in [15], and its proof now applies directly.

LEMMA 9. *[15] With the above gifts the total net gain obtained by the clairvoyant algorithm from scheduling the jobs abandoned during I_f is not greater than $(1+\sqrt{k})\cdot v_f$.*

The intuition behind this lemma is that the best that the adversary can do is to take almost all of the "gift" of $k\cdot(t_f^{close} - t_f^{begin})$ (intuitively, this is equivalent to executing jobs with the maximum possible value density over the time that Γ_1 is active), and then begin execution of a job abandoned by Γ_1 right before t_f^{close}. By Lemma 8, the value of this job is bounded by $(1+\sqrt{k})\cdot v_f$. We can now combine the results of these lemmas to prove the competitive ratio.

THEOREM 10. *Mechanism Γ_1 is $((1+\sqrt{k})^2+1)$-competitive.*

PROOF. Using the fact that the way in which jobs are associated with the intervals partitions the entire set of jobs, we can show the competitive ratio by showing that Γ_1 is $((1+\sqrt{k})^2+1)$-competitive for each interval in the sequence $(1,\ldots,F)$. Over an arbitrary interval I_f, the offline algorithm can achieve at most $(t_f^{close} - t_f^{begin})\cdot k + v_f + (1+\sqrt{k})v_f$, from the two gifts and the net gain bounded by Lemma 9. Applying Lemma 7, this quantity is then bounded from above by $(1+\frac{1}{\sqrt{k}})\cdot v_f\cdot k + v_f + (1+\sqrt{k})v_f = ((1+\sqrt{k})^2+1)\cdot v_f$. Since Γ_1 achieves v_f, the competitive ratio holds. □

4.3 Special Case: Unalterable length and k=1

While so far we have allowed each agent to lie about all four characteristics of its job, lying about the length of the job is not possible in some settings. For example, a user may not know how to alter a computational problem in a way that both lengthens the job and allows the solution of the original problem to be extracted from the solution to the altered problem. Another restriction that is natural in some settings is uniform value densities ($k=1$), which was the case considered by [4]. If the setting satisfies these two conditions, then, by using Mechanism Γ_2, we can achieve a competitive ratio of 5 (which is the same competitive ratio as Γ_1 for the case of $k=1$) without knowledge of ρ_{min} and without the use of payments. The latter property may be necessary in settings that are more local than grid computing (e.g., within a department) but in which the users are still self-interested.[7]

Mechanism 2 Γ_2

Execute $\mathcal{S}(\hat{\theta}, \cdot)$ according to Algorithm 2
for all i **do**
  $p_i(\hat{\theta}) \leftarrow 0$

Algorithm 2

for all t **do**
  $Avail \leftarrow \{i | (t \geq \hat{r}_i) \wedge (e_i(\hat{\theta}, t) < l_i) \wedge (e_i(\hat{\theta}, t) + \hat{d}_i - t \geq l_i)\}$
  if $Avail \neq \emptyset$ **then**
    $\mathcal{S}(\hat{\theta}, t) \leftarrow \arg\max_{i \in Avail}(l_i + e_i(\hat{\theta}, t))$
      {*Break ties in favor of lower* $\hat{r}_i$}
  else
    $\mathcal{S}(\hat{\theta}, t) \leftarrow 0$

THEOREM 11. *When $k=1$, and each agent i cannot falsely declare l_i, Mechanism Γ_2 satisfies individual rationality and incentive compatibility.*

THEOREM 12. *When $k=1$, and each agent i cannot falsely declare l_i, Mechanism Γ_2 is 5-competitive.*

Since this mechanism is essentially a simplification of Γ_1, we omit proofs of these theorems. Basically, the fact that $k=1$ and $\hat{l}_i = l_i$ both hold allows Γ_2 to substitute the priority $(l_i + e_i(\hat{\theta}, t))$ for the priority used in Γ_1; and, since $\hat{v}_i$ is ignored, payments are no longer needed to ensure incentive compatibility.

5. COMPETITIVE LOWER BOUND

We now show that the competitive ratio of $(1+\sqrt{k})^2 + 1$ achieved by Γ_1 is a lower bound for deterministic online mechanisms. To do so, we will appeal to third requirement on a mechanism, *non-negative payments* (NNP), which requires that the center never pays an agent (formally, $\forall i, \hat{\theta},\ p_i(\hat{\theta}_i) \geq 0$). Unlike IC and IR, this requirement is not standard in mechanism design. We note, however, that both Γ_1 and Γ_2 satisfy it trivially, and that, in the following proof, zero only serves as a baseline utility for an agent, and could be replaced by any non-positive function of $\hat{\theta}_{-i}$.

The proof of the lower bound uses an adversary argument similar to that used in [4] to show a lower bound of $(1+\sqrt{k})^2$ in the non-strategic setting, with the main novelty lying in the perturbation of the job sequence and the related incentive compatibility arguments. We first present a lemma relating to the recurrence used for this argument, with the proof omitted due to space constraints.

LEMMA 13. *For any $k \geq 1$, for the recurrence defined by $l_{i+1} = \lambda\cdot l_i - k\cdot\sum_{h=1}^{i} l_h$ and $l_1 = 1$, where $(1+\sqrt{k})^2 - 1 < \lambda < (1+\sqrt{k})^2$, there exists an integer $m \geq 1$ such that $\frac{l_m + k\cdot\sum_{h=1}^{m-1} l_h}{l_m} > \lambda$.*

[7]While payments are not required in this setting, Γ_2 can be changed to collect a payments without affecting incentive compatibility by charging some fixed fraction of l_i for each job i that is completed.

THEOREM 14. *There does not exist a deterministic online mechanism that satisfies NNP and that achieves a competitive ratio less than $(1+\sqrt{k})^2+1$.*

PROOF. Assume by contradiction that there exists a deterministic online mechanism Γ that satisfies NNP and that achieves a competitive ratio of $c = (1+\sqrt{k})^2 + 1 - \epsilon$ for some $\epsilon > 0$ (and, by implication, satisfies IC and IR as well). Since a competitive ratio of c implies a competitive ratio of $c + x$, for any $x > 0$, we assume without loss of generality that $\epsilon < 1$. First, we will construct a profile of agent types θ using an adversary argument. After possibly slightly perturbing θ to assure that a strictness property is satisfied, we will then use a more significant perturbation of θ to reach a contradiction.

We now construct the original profile θ. Pick an α such that $0 < \alpha < \epsilon$, and define $\delta = \frac{\alpha}{ck+3k}$. The adversary uses two sequences of jobs: minor and major. Minor jobs i are characterized by $l_i = \delta$, $v_i = k \cdot \delta$, and zero laxity. The first minor job is released at time 0, and $r_i = d_{i-1}$ for all $i > 1$. The sequence stops whenever Γ completes any job.

Major jobs also have zero laxity, but they have the smallest possible value ratio (that is, $v_i = l_i$). The lengths of the major jobs that may be released, starting with $i = 1$, are determined by the following recurrence relation.

$$\begin{aligned} l_{i+1} &= (c-1+\alpha) \cdot l_i - k \cdot \sum_{h=1}^{i} l_h \\ l_1 &= 1 \end{aligned}$$

The bounds on α imply that $(1+\sqrt{k})^2 - 1 < c - 1 + \alpha < (1+\sqrt{k})^2$, which allows us to apply Lemma 13. Let m be the smallest positive number such that $\frac{l_m + k \cdot \sum_{h=1}^{m-1} l_h}{l_m} > c - 1 + \alpha$.

The first major job has a release time of 0, and each major job $i > 1$ has a release time of $r_i = d_{i-1} - \delta$, just before the deadline of the previous job. The adversary releases major job $i \le m$ if and only if each major job $j < i$ was executed continuously over the range $[r_i, r_{i+1}]$. No major job is released after job m.

In order to achieve the desired competitive ratio, Γ must complete some major job f, because $\Gamma_{offline}$ can always at least complete major job 1 (for a value of 1), and Γ can complete at most one minor job (for a value of $\frac{\alpha}{c+3} < \frac{1}{c}$). Also, in order for this job f to be released, the processor time preceding r_f can only be spent executing major jobs that are later abandoned. If $f < m$, then major job $f+1$ will be released and it will be the final major job. Γ cannot complete job $f+1$, because $r_f + l_f = d_f > r_{f+1}$. Therefore, θ consists of major jobs 1 through $f+1$ (or, f, if $f = m$), plus minor jobs from time 0 through time d_f.

We now possibly perturb θ slightly. By IR, we know that $v_f \ge p_f(\theta)$. Since we will later need this inequality to be strict, if $v_f = p_f(\theta)$, then change θ_f to θ'_f, where $r'_f = r_f$, but v'_f, l'_f, and d'_f are all incremented by δ over their respective values in θ_f. By IC, job f must still be completed by Γ for the profile (θ'_f, θ_{-f}). If not, then by IR and NNP we know that $p_f(\theta'_f, \theta_{-f}) = 0$, and thus that $u_f(g(\theta'_f, \theta_{-f}), \theta'_f) = 0$. However, agent f could then increase its utility by falsely declaring the original type of θ_f, receiving a utility of: $u_f(g(\theta'_f, \theta_{-f}), \theta'_f) = v'_f - p_f(\theta) = \delta > 0$, violating IC. Furthermore, agent f must be charged the same amount (that is, $p_f(\theta'_f, \theta_{-f}) = p_f(\theta)$), due to a similar incentive compatibility argument. Thus, for the remainder of the proof, assume that $v_f > p_f(\theta)$.

We now use a more substantial perturbation of θ to complete the proof. If $f < m$, then define θ''_f to be identical to θ_f, except that $d''_f = d_{f+1} + l_f$, allowing job f to be completely executed after job $f+1$ is completed. If $f = m$, then instead set $d''_f = d_f + l_f$. IC requires that for the profile $(\theta''_f, \theta_{-f})$, Γ still executes job f continuously over the range $[r_f, r_f + l_f]$, thus preventing job $f+1$ from being completed.

Assume by contradiction that this were not true. Then, at the original deadline of d_f, job f is not completed. Consider the possible profile $(\theta''_f, \theta_{-f}, \theta_x)$, which differs from the new profile only in the addition of a job x which has zero laxity, $r_x = d_f$, and $v_x = l_x = max(d''_f - d_f, (c+1) \cdot (l_f + l_{f+1}))$. Because this new profile is indistinguishable from $(\theta''_f, \theta_{-f})$ to Γ before time d_f, it must schedule jobs in the same way until d_f. Then, in order to achieve the desired competitive ratio, it must execute job x continuously until its deadline, which is by construction at least as late as the new deadline d''_f of job f. Thus, job f will not be completed, and, by IR and NNP, it must be the case that $p_f(\theta''_f, \theta_{-f}, \theta_x) = 0$ and $u_f(g(\theta''_f, \theta_{-f}, \theta_x), \theta''_f) = 0$. Using the fact that θ is indistinguishable from $(\theta_f, \theta_{-f}, \theta_x)$ up to time d_f, if agent f falsely declared his type to be the original θ_f, then its job would be completed by d_f and it would be charged $p_f(\theta)$. Its utility would then increase to $u_f(g(\theta_f, \theta_{-f}, \theta_x), \theta''_f) = v_f - p_f(\theta) > 0$, contradicting IC.

While Γ's execution must be identical for both (θ_f, θ_{-f}) and $(\theta''_f, \theta_{-f})$, $\Gamma_{offline}$ can take advantage of the change. If $f < m$, then Γ achieves a value of at most $l_f + \delta$ (the value of job f if it were perturbed), while $\Gamma_{offline}$ achieves a value of at least $k \cdot (\sum_{h=1}^{f} l_h - 2\delta) + l_{f+1} + l_f$ by executing minor jobs until r_{f+1}, followed by job $f+1$ and then job f (we subtract two δ's instead of one because the last minor job before r_{f+1} may have to be abandoned). Substituting in for l_{f+1}, the competitive ratio is then at least: $\frac{k \cdot (\sum_{h=1}^{f} l_h - 2\delta) + l_{f+1} + l_f}{l_f + \delta} = \frac{k \cdot (\sum_{h=1}^{f} l_h) - 2k \cdot \delta + (c-1+\alpha) \cdot l_f - k \cdot (\sum_{h=1}^{f} l_h) + l_f}{l_f + \delta} = \frac{c \cdot l_f + (\alpha \cdot l_f - 2k \cdot \delta)}{l_f + \delta} \ge \frac{c \cdot l_f + ((ck+3k)\delta - 2k \cdot \delta)}{l_f + \delta} > c$.

If instead $f = m$, then Γ achieves a value of at most $l_m + \delta$, while $\Gamma_{offline}$ achieves a value of at least $k \cdot (\sum_{h=1}^{m} l_h - 2\delta) + l_m$ by completing minor jobs until $d_m = r_m + l_m$, and then completing job m. The competitive ratio is then at least: $\frac{k \cdot (\sum_{h=1}^{m} l_h - 2\delta) + l_m}{l_m + \delta} = \frac{k \cdot (\sum_{h=1}^{m-1} l_h) - 2k \cdot \delta + k l_m + l_m}{l_m + \delta} > \frac{(c-1+\alpha) \cdot l_m - 2k \cdot \delta + k l_m}{l_m + \delta} = \frac{(c+k-1) \cdot l_m + (\alpha l_m - 2k \cdot \delta)}{l_m + \delta} > c$. □

6. RELATED WORK

In this section we describe related work other than the two papers ([4] and [15]) on which this paper is based. Recent work related to this scheduling domain has focused on competitive analysis in which the online algorithm uses a faster processor than the offline algorithm (see, e.g., [13, 14]). Mechanism design was also applied to a scheduling problem in [18]. In their model, the center owns the jobs in an offline setting, and it is the agents who can execute them. The private information of an agent is the time it will require to execute each job. Several incentive compatible mechanisms are presented that are based on approximation algorithms for the computationally infeasible optimization problem. This paper also launched the area of algorithmic mechanism design, in which the mechanism must satisfy computational requirements in addition to the standard incentive requirements. A growing sub-field in this area is multicast cost-sharing mechanism design (see, e.g., [1]), in which the mechanism must efficiently determine, for each agent in a multicast tree, whether the agent receives the transmission and the price it must pay. For a survey of this and other topics in distributed algorithmic mechanism design, see [9].

Online execution presents a different type of algorithmic challenge, and several other papers study online algorithms or mechanisms in economic settings. For example, [5] considers an online market clearing setting, in which the auctioneer matches buy and sells bids (which are assumed to be exogenous) that arrive and expire over time. In [2], a general method is presented for converting an online algorithm into an online mechanism that is incentive compatible with respect to values. Truthful declaration of values is also considered in [3] and [16], which both consider multi-unit online auctions. The main difference between the two is that the former considers the case of a digital good, which thus has unlimited supply. It is pointed out in [16] that their results continue to hold when the setting is extended so that bidders can delay their arrival.

The only other paper we are aware of that addresses the issue of incentive compatibility in a real-time system is [11], which considers several variants of a model in which the center allocates bandwidth to agents who declare both their value and their arrival time. A dominant strategy IC mechanism is presented for the variant in which every point in time is essentially independent, while a Bayes-Nash IC mechanism is presented for the variant in which the center's current decision affects the cost of future actions.

7. CONCLUSION

In this paper, we considered an online scheduling domain for which algorithms with the best possible competitive ratio had been found, but for which new solutions were required when the setting is extended to include self-interested agents. We presented a mechanism that is incentive compatible with respect to release time, deadline, length and value, and that only increases the competitive ratio by one. We also showed how this mechanism could be simplified when $k = 1$ and each agent cannot lie about the length of its job. We then showed a matching lower bound on the competitive ratio that can be achieved by a deterministic mechanism that never pays the agents.

Several open problems remain in this setting. One is to determine whether the lower bound can be strengthened by removing the restriction of non-negative payments. Also, while we feel that it is reasonable to strengthen the assumption of knowing the maximum possible ratio of value densities (k) to knowing the actual range of possible value densities, it would be interesting to determine whether there exists a $((1+\sqrt{k})^2+1)$-competitive mechanism under the original assumption. Finally, randomized mechanisms provide an unexplored area for future work.

8. REFERENCES

[1] A. Archer, J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker, *Approximation and collusion in multicast cost sharing*, Games and Economic Behavior (to appear).

[2] B. Awerbuch, Y. Azar, and A. Meyerson, *Reducing truth-telling online mechanisms to online optimization*, Proceedings on the 35th Symposium on the Theory of Computing, 2003.

[3] Z. Bar-Yossef, K. Hildrum, and F. Wu, *Incentive-compatible online auctions for digital goods*, Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[4] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, *On the competitiveness of on-line real-time task scheduling*, Journal of Real-Time Systems **4** (1992), no. 2, 125–144.

[5] A. Blum, T. Sandholm, and M. Zinkevich, *Online algorithms for market clearing*, Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[6] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[7] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, *Economic models for resource management and scheduling in grid computing*, The Journal of Concurrency and Computation: Practice and Experience **14** (2002), 1507–1542.

[8] N. Camiel, S. London, N. Nisan, and O. Regev, *The popcorn project: Distributed computation over the internet in java*, 6th International World Wide Web Conference, 1997.

[9] J. Feigenbaum and S. Shenker, *Distributed algorithmic mechanism design: Recent results and future directions*, Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, 2002, pp. 1–13.

[10] A. Fiat and G. Woeginger (editors), *Online algorithms: The state of the art*, Springer Verlag, 1998.

[11] E. Friedman and D. Parkes, *Pricing wifi at starbucks-issues in online mechanism design*, EC'03, 2003.

[12] R. L. Graham, *Bounds for certain multiprocessor anomalies*, Bell System Technical Journal **45** (1966), 1563–1581.

[13] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, Journal of the ACM **47** (2000), 617–643.

[14] C. Koo, T. Lam, T. Ngan, and K. To, *On-line scheduling with tight deadlines*, Theoretical Computer Science **295** (2003), 251–261.

[15] G. Koren and D. Shasha, *D-over: An optimal on-line scheduling algorithm for overloaded real-time systems*, SIAM Journal of Computing **24** (1995), no. 2, 318–339.

[16] R. Lavi and N. Nisan, *Competitive analysis of online auctions*, EC'00, 2000.

[17] A. Mas-Colell, M. Whinston, and J. Green, *Microeconomic theory*, Oxford University Press, 1995.

[18] N. Nisan and A. Ronen, *Algorithmic mechanism design*, Games and Economic Behavior **35** (2001), 166–196.

[19] C. Papadimitriou, *Algorithms, games, and the internet*, STOC, 2001, pp. 749–753.