Preemptive Scheduling in Overloaded Systems

Marek Chrobak [a,1,4], Leah Epstein [b,2], John Noga [c], Jiří Sgall [d,*,3,4], Rob van Stee [e,5], Tomáš Tichý [d,3,4], Nodari Vakhania [f,6]

[a] *Department of Computer Science, University of California, Riverside, CA 92521, U.S.A.*

[b] *School of Computer Science, The Interdisciplinary Center, P.O.B. 167, 46150 Herzliya, Israel*

[c] *Department of Computer Science, California State University, Northridge, CA 91330, U.S.A.*

[d] *Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic*

[e] *Institut für Informatik, Albert-Ludwigs-Universität, Georges-Köhler-Allee, 79110 Freiburg, Germany*

[f] *Facultad de Ciencias, Universidad Autonoma del Estado de Morelos, 62251 Cuernavaca, Morelos, Mexico*

Abstract

The following scheduling problem is studied: We are given a set of tasks with release times, deadlines, and profit rates. The objective is to determine a 1-processor preemptive schedule of the given tasks that maximizes the overall profit. In the standard model, each completed task brings profit, while non-completed tasks do not. In the metered model, a task brings profit proportional to the execution time even if not completed.

For the metered task model, we present an efficient offline algorithm and improve both the lower and upper bounds on the competitive ratio of online algorithms. Furthermore, we prove three lower bound results concerning resource augmentation in both models.

Key words: Scheduling, online algorithms, deadline, resource augmentation.

1 Introduction

In most task scheduling problems the objective is to minimize some function related to the completion time. This approach is not useful in overloaded systems, where the number of tasks and their processing times exceed the capacity of the processor and not all tasks can be completed. In such systems, the goal is usually to maximize the number of executed tasks or, more generally, to maximize their value or profit.

The problem can be formalized as follows: we have a set of n tasks, each task j is specified by its release time r_j, deadline d_j, processing time p_j, and weight w_j representing its profit rate. Preemption is allowed, i.e., each task can be divided into any number of intervals, with arbitrary granularity. The objective is to determine a 1-processor preemptive schedule that maximizes the overall profit. The profit gained from processing task j can be defined in two ways. In the *standard model*, each completed task j brings profit $w_j p_j$, but non-completed tasks do not bring any profit. In the *metered model*, a task w_j executed for time $t \leq p_j$ brings profit $w_j t$ even if it is not completed.

In many real-world applications, algorithms for task scheduling are required to be *online*. This means that, at any given time, the scheduling algorithm needs to choose the task to process based only on the specification of the tasks that have already been released. In general, due to the incomplete information about the input data, online algorithms cannot compute an optimal solution. It turns out, however, that for some scheduling problems it is possible to compute in an online fashion a solution that is within a constant factor of the optimum.

An online algorithm that approximates the optimal solution within a factor R is called *R-competitive*. Online algorithms are also studied in the framework

* Corresponding author.
Email addresses: marek@cs.ucr.edu (Marek Chrobak), lea@idc.ac.il (Leah Epstein), jnoga@ecs.csun.edu (John Noga), sgall@math.cas.cz (Jiří Sgall), vanstee@informatik.uni-freiburg.de (Rob van Stee), tichy@math.cas.cz (Tomáš Tichý), nodari@servm.fc.uaem.mx (Nodari Vakhania).
[1] Supported by NSF grant CCR-9988360.
[2] Supported by the Israel Science Foundation, grant No. 250/01-1.
[3] Supported by Institute for Theoretical Computer Science, Prague (project LN00A056 of MŠMT ČR), grant 201/01/1195 of GA ČR, and grant A1019901 of GA AV ČR.
[4] Supported by cooperative grant KONTAKT-ME476/CCR-9988360-001 from MŠMT ČR and NSF.
[5] Supported by the Deutsche Forschungsgemeinschaft, Project AL 464/3-1, and by the European Community, Projects APPOL and APPOL II.
[6] Supported by NSF-CONACyT grant E120.1914.

called *resource augmentation*. The idea is to allow an online algorithm to use more resources (a faster processor or more processors) and then to compare its performance to the optimum solution (with no additional resources). For the scheduling problems, we then ask what competitive ratio can be achieved for a given speed-up factor s, or what speed-up is necessary to achieve 1-competitiveness. See [13,2] for more information on competitive analysis in scheduling and other areas.

The standard model. This problem has been extensively studied. Koren and Shasha [9] give a $(\sqrt{\xi}+1)^2$-competitive algorithm, where $\xi = \max_j w_j / \min_j w_j$ is called the *importance factor*. This ratio is in fact optimal [1,9]. Since no constant-competitive algorithms are possible in this model, it is natural to study this problem under the resource augmentation framework. Kalyanasundaram and Pruhs [6] present an online algorithm that uses a processor with speed 32 and achieves a constant competitive ratio. Lam and To [11] show an online algorithm with speed-up $O(\log \xi)$ and competitive ratio 1. One natural special case of this problem is when the tasks are *tight*, that is, for each j we have $d_j = r_j + p_j$. For this case, Koo et al. [8] give a 1-competitive algorithm with speed-up $O(1)$, and Lam et al. [10] show that in order to achieve 1-competitiveness the speed-up must be at least $\phi \approx 1.618$.

The metered model. This version was introduced (in a different terminology) by Chang and Yap [3] in the context of thinwire visualization. In their application, a user viewing a low-resolution image moves the cursor across the screen, generating requests for higher resolution data at the cursor positions. Due to limited bandwidth not all requests can be fully satisfied. However, even partial improvements of resolution may be beneficial to the viewer. Thus the profit represents overall *quality of service*. Metered preemptive tasks also provide a natural model for various decision making processes where an entity with limited resources needs to choose between engaging in several profitable activities. Chang and Yap proved that two online algorithms called FirstFit and EndFit have competitive ratio 2. They also proved that no online algorithm can achieve a competitive ratio better than $2(2-\sqrt{2}) \approx 1.17$.

Our results. We first focus on the metered profit model. In Section 3, we consider offline algorithms. We characterize the structure of optimal solutions and provide a polynomial time algorithm based on bipartite matchings and maximal flows. This addresses a problem stated in [3].

The online metered case is studied in Section 4 to 6. In Section 4 we present an algorithm with competitive ratio $e/(e-1) \approx 1.5820$. In Section 5 we prove a lower lower bound of $\sqrt{5}-1 \approx 1.236$ on the competitive ratio of algorithms for this problem. These results improve both the lower and upper bounds from [3]. (The algorithm FirstEndFit, conjectured in [3] to be 1.5-competitive, is only

2-competitive, as we have shown in the conference version of this paper [5].)

In Section 6 we study the resource augmentation version of this problem, and prove that no online algorithm with constant speed-up can be 1-competitive, neither in the metered profit model, nor in the standard model. In fact, we prove that the minimal speed-up needed to achieve 1-competitiveness is $\Omega(\log\log\xi)$. Thus we disprove a conjecture from [8] by showing that the problem with general deadlines is provably harder than the special case of tight deadlines, and the constant speed-up 1-competitive algorithm for tight tasks from [8] cannot be extended to general tasks.

Furthermore, we prove some lower bounds for the restricted case of tight tasks in the standard model. We improve the lower bound from [10], by proving that, in order to achieve 1-competitiveness, an online algorithm needs speed-up at least 2. Our last result concerns the model where an online algorithm is allowed to use m processors of speed 1, rather than a single faster processor. For this case we prove that the competitive ratio is $\Omega(\sqrt[m]{\xi}/m)$, even if all tasks are restricted to be tight. For tight tasks constant speed-up is sufficient for 1-competitiveness, so the lower bound shows that increasing the speed of a single processor is more powerful than increasing the number of processors of speed 1.

This paper extends the conference version [5]. The conference version contains a 1.8-competitive algorithm for metered tasks which alway schedules at most two tasks. Our new $e/(e-1)$-competitive algorithm is a natural extension allowing to schedule more tasks concurrently. The same algorithm was also recently discovered by Chin and Fung [4] (independently of our work). Chin and Fung [4] also give a new lower bound of 1.25 for metered tasks.

2 Preliminaries

Let $J = \{1, 2, \ldots, n\}$ be the given set of tasks, with task j specified by the values (r_j, d_j, p_j, w_j), where r_j is its release time, d_j is the deadline, p_j is the processing time, and w_j is the weight of task j representing its *profit rate*. (In the literature, w_j is sometimes called the *value density*, and the product $w_j p_j$ is called the *value* of task j.) We assume $\min_j r_j = 0$ and we denote by $D = \max_j d_j$ the latest deadline. If $r_j \leq t < d_j$, then we say that task j is *feasible at time* t.

Schedules. We define a *schedule* for J to be a measurable function $S : \mathbb{R} \to J \cup \{\perp\}$ such that, for each j and t, $|S^{-1}(j)| \leq p_j$ and $S(t) \neq j$ for $t \notin [r_j, d_j)$. In this definition, $S(t)$ denotes the task that is scheduled at time t, and $S(t) = \perp$

if no task is scheduled. For a set $X \subseteq \mathbb{R}$, $|X|$ denotes the size (measure) of X.

The *profit* of a schedule S depends on the model: In the *standard model*, the profit is the sum of the profits of the completed tasks, that is $\mathit{profit}_S(J) = \sum_j w_j p_j$, where the sum is taken over all j for which $|S^{-1}(j)| = p_j$. In the *metered model*, even partially executed tasks count, that is, the profit of the schedule S is $\mathit{profit}_S(J) = \sum_j w_j |S^{-1}(j)|$. The optimal profit is $\mathit{profit}_{\mathrm{OPT}}(J) = \sup_S \mathit{profit}_S(J)$. It is easy to see that this supremum is achieved. Moreover, each schedule can be transformed into a piece-wise constant schedule without changing the total profit (see [3]). The profit of a schedule generated by an algorithm $\mathcal{A}$ on the instance J is denoted by $\mathit{profit}_{\mathcal{A}}(J)$.

For the metered model, it is important to keep in mind that the optimum profit is not changed if any task is divided into several tasks with the same release times, deadlines, and weights, and whose total processing time is equal to the processing time of the original task. (For this reason it is more natural to define the weight as the profit rate instead of the total profit.)

For a schedule S, let $\mathit{done}_{S,j}(t) = |S^{-1}(j) \cap [0, t)|$ be the amount of task j that has been processed in S by time t. We define a task j to be *active* in S at time t if $r_j \leq t < d_j$ and $\mathit{done}_{S,j}(t) < p_j$. In other words, the active tasks are those that are feasible at time t and have not been completely processed before time t.

We say that a schedule S is *canonical* if for any two times $t_1 < t_2$, if $j_2 = S(t_2) \neq \perp$, then either $r_{j_2} > t_1$, or $j_1 = S(t_1) \neq \perp$ and $d_{j_1} \leq d_{j_2}$. One way to think about canonical schedules is this: at each time t, if j is the earliest-deadline task among the active tasks at time t, then we either process j at time t, or discard j irrevocably so that it will never be processed in the future. Any schedule S, including an optimal one, can be converted into a canonical schedule as follows. Consider the instance J' consisting of the portions of tasks that are processed in S. Reschedule the tasks in J' so that at each time we schedule the active task with the earliest deadline. Using a standard exchange argument, it is easy to verify that all tasks are fully processed.

Online algorithms. A scheduling algorithm $\mathcal{A}$ is *online* if, at any time t, its schedule depends only on the tasks that have been released before or at time t. An online algorithm $\mathcal{A}$ is called *R-competitive* if $\mathit{profit}_{\mathcal{A}}(J) \geq \mathit{profit}_{\mathrm{OPT}}(J)/R$ for every instance J. The *competitive ratio* of $\mathcal{A}$ is the smallest R for which $\mathcal{A}$ is R-competitive.

Timesharing and randomizaton. The online algorithms are easier to formulate if we allow timesharing of tasks. This means that several tasks may be processed simultaneously at appropriately reduced speeds. As explained below, this does not change the power of the model of metered tasks.

Formally, a *generalized schedule* is a function V that, for each task j and time $t \in [0, D)$, specifies the speed $V(j,t)$ at which we perform task j at time t. We impose the following restrictions on $V(j,t)$:

$$\sum_j V(j,t) \leq 1, \quad \int_0^\infty V(j,t)\,dt \leq p_j, \quad \text{and} \quad V(j,t) = 0 \text{ for } t \notin [r_j, d_j)$$

The first condition states that the sum of the processing speeds assigned to different tasks cannot exceed the processor speed, and the second condition states that the total time spent on executing task j does not exceed p_j.

The profit of a generalized schedule V is

$$\mathit{profit}_V(J) \;=\; \sum_j w_j \int_0^\infty V(j,t)\,dt \;=\; \int_0^\infty \sum_j w_j V(j,t)\,dt.$$

Clearly, this definition generalizes the previous one. Both definitions are equivalent in the offline case.

In the online case, any generalized schedule V can be transformed into a schedule S which simulates the time-sharing in V by alternating the tasks. It is easy to see that if the tasks are alternated with sufficiently high frequency (compared to the processing times), this transformation increases the competitive ratio only by an arbitrarily small $\varepsilon > 0$. So both definitions are equivalent in the online case as well, in the sense that the infima of achievable competitive ratios are the same. Throughout the paper we slightly abuse terminology and refer to the function V simply as a *schedule*.

It is also easy to see that randomized (online) algorithms are no more powerful than deterministic ones for metered tasks. Any randomized algorithm can be transformed into a deterministic one by generating a generalized schedule in which at a given time, each task is processed with speed equal to the probability that the randomized algorithm schedules it. It is easy to see that any scheduled task is active, since when the randomized algorithm schedules it with a non-zero probability, it is not completed and feasible at the given time. Moreover, the profit of the deterministic algorithm is exactly equal to the expected profit of the randomized algorithm. (Note that this fails in the standard model: The randomized algorithm may complete a task with probability $1/2$, achieving one half of its profit on average, in which case the deterministic algorithm schedules only a part of the task and achieves no profit.)

Resource augmentation. As mentioned in the introduction, we also study two variants of the problem where the online algorithms are given more resources than the optimal schedule used as the basis of comparison. In the first variant, with speed-up s, the online algorithm uses a single machine of speed $s \geq 1$. A schedule for J is a measurable function $S : \mathbb{R} \to J \cup \{\perp\}$ such that, for each j and t, $|S^{-1}(j)| \leq p_j/s$ and $S(t) \neq j$ for $t \notin [r_j, d_j)$. A profit of a job is $w_j s|S^{-1}(j)|$. In the standard model, a job is completed if $s|S^{-1}(j)| = p_j$ and the profit of the schedule is the sum of the profits of all completed jobs. In the metered model, the profit is the sum of profits of all jobs, even those only partially completed.

In the second variant, the online algorithm uses m processors of speed 1. A schedule for J is then given by an m-tuple of measurable functions $S_i : \mathbb{R} \to J \cup \{\perp\}$, $i = 1, \dots, m$, such that $\sum_{i=1}^{m} |S_i^{-1}(j)| \leq p_j$ for each j and t, $S_i(t) \neq j$ for each i and $t \notin [r_j, d_j)$, and, if $S_i(t) = S_{i'}(t)$ for some j and t, then $S_i(t) = S_{i'}(t) = \perp$ (i.e., no job is scheduled on two machines at the same time). We consider only the standard model for this variant. A job is completed if $\sum_{i=1}^{m} |S_i^{-1}(j)| = p_j$. If j is completed, its profit is $w_j p_j$, and otherwise it's 0. The total profit is the sum of the profits of all completed jobs.

In both variants, we compare a schedule generated by the online algorithm that uses additional resources to the optimal schedule with no speed-up and no additional machines. We are mainly interested in 1-competitive algorithms, that is, in algorithms that always achieve at least the optimal profit. (Due to the additional resources such an algorithm can achieve a larger profit on some instances.)

3 An Offline Algorithm for Metered Tasks

In this section we give an efficient algorithm for computing the optimal solution for metered tasks, addressing a problem posed in [3]. First we observe that the problem can be cast as a linear programming problem, and thus it can be solved in polynomial time. The main goal of this section is to present a more efficient algorithm based on bipartite matchings and flows.

The release times and deadlines partition the range $[0, D)$ into $2n-1$ intervals that we call *stages*. We number the stages $1, 2, \dots, 2n-1$. If stage s is $[a, b)$, we say that task j is *feasible in stage s* if it is feasible at any time $t \in [a, b)$.

Linear programming. By ℓ_s we denote the length of stage s. Let $\delta_{j,s} = p_j$ if j is feasible in s and 0 otherwise. With each stage s and each task j we associate the variable $x_{j,s}$ whose value is the amount of task j processed in stage s. Any schedule can be described by the values of the $x_{j,s}$, since the ordering of the tasks scheduled within a stage is arbitrary. Then the linear program is:

$$
\begin{array}{ll}
\text{maximize } \sum_{j,s} w_j x_{j,s} & \\
\quad\text{s.t.} \quad \sum_s x_{j,s} \le p_j \quad \forall j & \\
\qquad\qquad \sum_j x_{j,s} \le \ell_s \quad \forall s & (1)\\
\qquad\qquad\quad x_{j,s} \le \delta_{j,s} \; \forall j,s & \\
\qquad\qquad\quad x_{j,s} \ge 0 \quad \forall j,s &
\end{array}
$$

Thus we can compute an optimal schedule using linear programming, see, e.g., [7,12]. This is not fully satisfactory, since the running time of polynomial-time algorithms for linear programming depends on the size of the numbers on input.

Matchings and flows. Now we present a more efficient algorithm, whose running time is a polynomial function of n alone (assuming unit time for arithmetic operations and comparisons of the real numbers representing times).

Before giving the algorithm, we prove the following property: any optimal schedule, restricted to a subset of jobs with weights larger than some threshold, is an optimal schedule for this subset of jobs. Thus, perhaps surprisingly, the set of optimal schedules depends only on the ordering of the weights but not on their values, and every optimal schedule contains an optimal schedule for any instance restricted to heavy tasks. In particular, all the optimal schedules include the same portion of the tasks of any given weight.

We prove this property first for the discrete version and then discretize and take a limit for the general case.

Order the tasks in an instance J so that $w_1 \ge w_2 \ge \cdots \ge w_n$. Without loss of generality, $w_n > 0$. For convenience, write $w_{n+1} = 0$. Let J_k denote the sub-instance consisting of tasks $1, \ldots, k$. Given a schedule S for J, let S_k be the restriction of S to J_k. In particular, $S_n = S$. Let $busy(S) = |S^{-1}(J)|$ be the total time when any task is scheduled in S.

Assume now that all the release times and deadlines are integers and that we only have unit tasks (with $p_j = 1$). Recall that the first release time is 0 and the latest deadline is D. In this scenario preemptions are not necessary in the offline case. Construct a bipartite graph G with vertices $X = \{x_1, \ldots, x_n\}$ corresponding to tasks and $Y = \{y_1, \ldots, y_D\}$ corresponding to the unit time slots. If task j is feasible in time unit t then connect x_j and y_t with an edge of weight w_j. Let G_k denote the subgraph of G induced by $\{x_1, \ldots, x_k\} \cup Y$. Any schedule defines a matching in G and any matching is a schedule. So computing an optimal schedule is equivalent to computing a maximum-weight matching.

Lemma 3.1 (a) *There exists a maximum-weight matching M in G such that, for each $k = 1, \ldots, n$, M contains a maximum-cardinality matching of G_k.*

(b) *If M is any maximum-weight matching in G then, for each $k = 1, \ldots, n$ such that $w_k > w_{k+1}$, M contains a maximum-cardinality matching of G_k.*

Proof: Let M be a maximum-weight matching in G. Denote by M_k the sub-matching of M restricted to G_k, and suppose that $\hat{M}$ is a matching in G_k with cardinality larger than M_k. Consider the symmetric difference of M_k and $\hat{M}$. It consists of disjoint alternating cycles and paths. Moreover, it contains at least one odd-length alternating path P with more edges from $\hat{M}$ than from M_k. Let y be the endpoint of P in Y. Since the weights of the edges depend only on the endpoints in X, the consecutive pairs of adjacent edges on P have equal weights (1st and 2nd, 3rd and 4th, etc., numbering from y), and the last edge is from $\hat{M}$ and has weight $w_p \geq w_k$. If y were not matched in M, we could swap the edges on P in M (i.e., remove from M the edges in $M \cap P = M_k \cap P$ and add the edges in $\hat{M} \cap P$), creating a matching with weight $w(M) + w_p$, contradicting the maximality of M. Thus y has to be matched in M. Then the matching edge containing y has weight w_l, for some $l > k$, and thus $w_p > w_l$. Let M' be the matching obtained from M by unmatching y and swapping the edges on P. We have $w(M') = w(M) + w_p - w_l \geq w(M)$. This implies (b): if $w_k > w_{k+1}$ then $w_l < w_p$ and $w(M') > w(M)$, which contradicts the maximality of M. To obtain (a), modify all the weights to $w'_i = w_i - i\varepsilon$, for $\varepsilon > 0$. For a sufficiently small ε, the maximality of any matching is preserved and all the weights are distinct. Using (b) with the new weights, it follows that any maximum-weight matching satisfies (a) with the original weights. □

For a general instance, round each processing time, release time, and deadline to the nearest multiple of $\varepsilon > 0$. Taking a limit for $\varepsilon \to 0$, Lemma 3.1 implies the following theorem.

Theorem 3.2 (a) *There exists an optimal schedule S for J such that, for each $k = 1, \ldots, n$, S_k is optimal for J_k and busy(S_k) is maximized.*

(b) *If S is any optimal schedule then, for each $k = 1, \ldots, n$ such that $w_k > w_{k+1}$, S_k is optimal for J_k and busy(S_k) is maximized.*

Proof: Discretize the range $[0, D)$ into small intervals of length $\varepsilon > 0$, rounding the release times and deadlines to the nearest multiple of ε. Replace each processing time p_j is replaced by the nearest multiple of ε. Let J_ε be the resulting discrete instance. Each schedule can be replaced by a canonical schedule without changing *busy*(S_k). A canonical schedule S for J consists of $O(n^2)$ intervals, it is easy to see that there are optimal schedules S_ε for each J_ε, that converge to S with $\varepsilon \to 0$. By Lemma 3.1 and taking the limit, the theorem follows. □

Algorithm Opt.

(i) Construct a flow network H with source s, sink t, and vertices x_j for each task j and z_i for each stage i. The edges are: (s, x_j) for each task j, (x_j, z_i) for each stage i and each task j feasible in stage i, and (z_i, t) for each stage i. The capacity of each edge (x_j, z_i) is p_j and the capacity of each edge (z_i, t) is ℓ_i, the length of stage i. The capacities of edges (s, x_j) are initialized to 0. Initialize f'_0 to be the zero flow.

(ii) For $j = 1, \ldots, n$, do the following:
Set the capacity of (s, x_j) to p_j.
Compute the maximal flow, denote it f_j. Let $\delta_j = |f_j| - |f_{j-1}|$ be the increase in the flow value.
Set the capacity of (s, x_j) to δ_j. Recompute the maximal flow, denote it f'_j.

The resulting maximum flow f'_n defines a schedule: we take $f_n(x_j, z_i)$ to be the amount of task j scheduled during stage i. Theorem 3.2 implies that $|f_j| = |f'_j|$ and after iteration j, the flow on (s, x_j) will remain δ_j until the end. Thus Algorithm Opt computes step by step $busy(S_j) = |f_j|$, $j = 1, \ldots, n$, for an optimal schedule S. Therefore f'_n defines an optimal schedule. The running time of Algorithm Opt is no worse than $O(n)$ times the complexity of the maximum flow, which is not worse than $O(n^4)$.

4 A Competitive Online Algorithm for Metered Tasks

We now present our $e/(e-1)$-competitive online algorithm. Chang and Yap introduced a greedy algorithm FirstFit that always processes the heaviest task. The drawback of FirstFit is that it may schedule a task with a distant deadline, discarding an only slightly less profitable task with a tight deadline. To avoid this, our algorithm Mixed schedules concurrently several tasks: the heaviest task, the heaviest task among those with an earlier deadline, and so on, up to a task with earliest deadline among those with weight at least $1/e$ of the largest weight.

Algorithm Mixed. Construct a sequence of jobs $h_1, \ldots, h_k$ as follows. Let h_1 be an active task j with maximum w_j (break ties arbitrarily). Given $h_1, \ldots, h_i$, choose the next job h_{i+1} as the heaviest active job j with deadline $d_j < d_{h_i}$ (breaking ties arbitrarily); if there is no such job or the job has weight $w_j \leq w_{h_1}/e$, set $k = i$ and finish the construction. Denote the weights of the chosen jobs $v_i = w_{h_i}$ and set $v_{k+1} = v_1/e$. Schedule all jobs h_i, $i = 1, \ldots, k$, with speed $V(h_i, t) = \ln v_i - \ln v_{i+1}$. For $j \notin \{h_1, \ldots, h_k\}$ set $V(j, t) = 0$.

At any given time, $v_1 \geq v_2 \geq \ldots \geq v_k \geq v_{k+1}$, thus $V(h_i, t) \geq 0$, and the sum of speeds of all scheduled jobs is $\sum_{i=1}^{k}(\ln v_i - \ln v_{i+1}) = \ln v_1 - \ln v_{k+1} = \ln v_1 - \ln(v_1/e) = 1$. Thus the schedule is well-defined. Note also that Mixed keeps processing the same tasks at the same speeds in-between any of following at most $2n$ events: task arrivals, deadlines or task completions.

Theorem 4.1 *Algorithm* Mixed *has competitive ratio* $e/(e-1) \approx 1.5820$.

Proof: Let V be the schedule generated by Mixed and let S be some canonical optimal schedule.

We devise an appropriate charging scheme, described by a function $C : \mathbb{R} \to \mathbb{R}$ which maps each time in S to a time in V. The intention is that any profit achieved at time t in S is "charged" to the time $C(t)$ in V. We then argue that each time u in V is charged at most $e/(e-1)$ times the profit in V at time u. Further, all profit from S is charged. These two facts imply $e/(e-1)$-competitiveness. Since the "profit at time t" is infinitesimally small, in the formal proof we need to express our argument in terms of "charged profit rates". We define another function $F : \mathbb{R} \to [0, 1]$ with the intended meaning that $F(t)$ is the fraction of profit at time t in S charged to time $C(t)$ in V.

Consider a time t, and let $j = S(t)$. If $done_{V,j}(t) \leq done_{S,j}(t)$, define $C(t) = t$ and $F(t) = 1$, i.e., the charged profit rate of j at time t is v_j. Otherwise, let $C(t) = u < t$, where u is the maximum time such that $done_{V,j}(u) = done_{S,j}(t)$, and $F(t) = V(j, u)$; i.e., the charged profit rate of j at time u is $v_j V(j, u)$. Choosing u as maximal such time implies that $V(j, u) > 0$. It is easy to check that the total charged profit (i.e., the charged profit rate integrated over the whole schedule V) equals the total profit of S.

Let j be the task scheduled in S at time t. Let $h_1, \ldots, h_k$ be the tasks scheduled in V at time t, as chosen by Mixed. Denote their weights $v_i = w_{h_i}$ and set $v_{k+1} = v_1/e$. The set $C^{-1}(t)$ consists of at most $k+1$ points, t and the maximal times $t_1, \ldots, t_k$ such that $done_{V,h_i}(t) = done_{S,h_i}(t_i)$, if such t_i exists and satisfies $t_i > t$. If $C(t) = t$ then the charged profit rate of j at t is w_j. If t_i is present in $C^{-1}(t)$ then the charged profit rate of h_i at t is $V(h_i, t)v_i$. The combined charged profit rate at t is the sum of these contributions over all points in $C^{-1}(t)$. We show that it is at most $e/(e-1)$ times the profit rate of V at time t. The theorem then follows by integrating over all times t.

Case 1: $C(t) < t$. Then the combined charged profit rate at t is at most the profit rate of V at t (it is equal if all t_i are present in $C^{-1}(t)$).

Case 2: $C(t) = t$. Consider any i such that the point t_i is present in $C^{-1}(t)$, we show that $w_j \leq v_{i+1}$. Both tasks j and h_i are active at time t both in V and S: both are released before t, as one of V and S schedules schedules them at t; both are scheduled at t or later in S, so they are not completed in S at time t; h_i is scheduled in V, so it is not completed; finally j is not completed in V by definition of C and $C(t) = t$. Since S is a canonical schedule, we have $d_j \le d_{h_i}$. If $w_j > v_{i+1}$, Mixed would choose j (or a job with even larger weight) as h_{i+1} (note that this holds even for $i = k$). It follows that $w_j \le v_{i+1}$.

Let z be the largest index (in $\{1, \dots, k+1\}$) such that $w_j \le v_z$. Since j is active at t in V, it also follows that $w_j \le v_1$, and thus such a z exists. By the previous paragraph, only j and $h_1, \dots, h_{z-1}$ contribute to the charged profit rate at t. Thus the combined charged profit rate at t is at most $X = v_z + \sum_{i=1}^{z-1} V(h_i, t) v_i$, using also $w_j \le v_z$. The profit rate of V at t is equal to $Y = \sum_{i=1}^{k} V(h_i, t) v_i$. We need to show that $X \le Y \cdot e/(e-1)$, or equivalently $X - Y \le Y/(e-1)$.

First we derive an auxiliary inequality for any $t = 1, \dots, k$:

$$
\begin{aligned}
\sum_{i=t}^{k} V(h_i,t) v_i &= \sum_{i=t}^{k} \int_{\ln v_{i+1}}^{\ln v_i} v_i \, dx \;\ge\; \sum_{i=t}^{k} \int_{\ln v_{i+1}}^{\ln v_i} e^x \, dx \\
&= \int_{\ln v_{k+1}}^{\ln v_t} e^x \, dx \;=\; v_t - v_{k+1} \;=\; v_t - v_1/e.
\end{aligned}
$$

As a special case for $t = 1$ we have $Y \ge v_1(1 - 1/e) = v_1(e-1)/e$. Now we have

$$
X - Y = v_z - \sum_{i=z}^{k} V(h_i,t) v_i \le v_z - (v_z - v_1/e) = v_1/e \le Y/(e-1).
$$

To see that the bound of $e/(e-1)$ on the competitive ratio of Mixed is tight, consider a small $\varepsilon > 0$ and an instance with jobs $(0, 1 - i\varepsilon^2, 1, 1 - i\varepsilon)$, for $i = 0, 1, \dots, \lfloor 1/\varepsilon \rfloor$. The deadlines are all very close to 1 (they serve only to break ties in the algorithm in the desired way), and the weights cover with high density the interval $[0, 1]$. Thus it is easy to check that, as the ε tends to 0, the profit of Mixed converges to $1 - 1/e$, while the optimum is e. □

5 A Lower Bound for Metered Tasks

The idea of the lower bound we prove now is as follows. At each integral time the algorithm has a choice of a job with unit processing time and tight deadline and another unit job with higher weight and longer deadline. If the algorithm schedules at most one half of the tight job, the sequence ends, and the weights are set so that in this case the competitive ratio is too large. Otherwise the sequence continues for a sufficiently long time. During this time the weights increase exponentially and in the limit the competitive ratio is large, too. This lower bound was recently improved by Chin and Fung [4] to 1.25, using an analysis based on a random distribution of similar input instances.

Theorem 5.1 *The competitive ratio of any online algorithm for scheduling metered tasks is at least* $\sqrt{5}-1 \approx 1.236$.

Proof: Fix an online algorithm $\mathcal{A}$ and $\varepsilon > 0$ arbitrarily small. We show that the competitive ratio of $\mathcal{A}$ is at least $\sqrt{5}-1-\varepsilon$, which proves the theorem.

Let $\sigma = \sqrt{5}-2$ and let $\phi = (\sqrt{5}+1)/2$ be the golden ratio. Define the sequence $\{v_i\}_{i=0}^{\infty}$ by $v_0 = 1$, $v_1 = \phi + \varepsilon$, and $v_{i+1} = (v_i - v_{i-1})/\sigma$ for $i > 1$. We solve the recurrence: $\phi + 1$ and ϕ are the roots of the characteristic equation $\sigma x^2 - x + 1 = 0$, and we have $v_i = (1-\varepsilon)\phi^i + \varepsilon(\phi+1)^i$.

The adversary strategy is this: Pick some large integer n. For each time $i = 0, 1, 2, \ldots$ the following two tasks arrive

task i: $(i, i+1, 1, v_i)$, task i': $(i, i+2, 1, v_{i+1})$.

If there is an integer time $1 \le j < n$ when $\mathcal{A}$ has completed at most half of task $j-1$, the adversary terminates the sequence (prior to releasing tasks j and j'). If this case occurs, $\mathcal{A}$ earns at most $\mathit{profit}_{\mathcal{A}}(J) \le \frac{1}{2}v_0 + v_1 + v_2 + \cdots + v_{j-1} + v_j$, and the optimal profit is $\mathit{profit}_{\mathrm{OPT}}(J) = (v_1 + v_2 + \cdots + v_{j-1} + v_j) + v_{j-1}$. Using the recurrence and $v_{j-1} \ge 1$ (in the last inequality) we obtain

$$
\begin{aligned}
\frac{\mathit{profit}_{\mathrm{OPT}}}{\mathit{profit}_{\mathcal{A}}(J)} &\ge \frac{(1 + 2\sum_{i=1}^{j} v_i) + 2v_{j-1} - 1}{1 + 2\sum_{i=1}^{j} v_i} \\
&= 1 + \frac{2v_{j-1} - 1}{1 + 2v_1 + 2\sum_{i=2}^{j} \frac{v_{i-1} - v_{i-2}}{\sigma}} \\
&= 1 + \frac{\sigma(2v_{j-1} - 1)}{\sigma + 2\sigma v_1 + 2v_{j-1} - 2} \\
&= 1 + \frac{\sigma(2v_{j-1} - 1)}{2\varepsilon\sigma + (2v_{j-1} - 1)} \; > \; 1 + \sigma - \varepsilon.
\end{aligned}
$$

Otherwise, the adversary issues all tasks up to time $n-1$, and at time n he releases task n only. Now $\mathit{profit}_{\mathcal{A}}(J) \le \frac{1}{2}v_0 + v_1 + v_2 + \ldots + v_{n-1} + \frac{3}{2}v_n$ and $\mathit{profit}_{\mathrm{OPT}}(J) = v_1 + v_2 + \ldots + v_{n-1} + 2v_n$. Using the recurrence and letting $n \to \infty$

$$
\begin{aligned}
\frac{\mathit{profit}_{\mathrm{OPT}}}{\mathit{profit}_{\mathcal{A}}(J)} &= \frac{2v_n + 2\sum_{i=1}^{n} v_i}{1 + v_n + 2\sum_{i=1}^{n} v_i} = 1 + \frac{v_n - 1}{1 + v_1 + v_n + 2\frac{v_{n-1} - 1}{\sigma}} \\
&\longrightarrow 1 + \frac{\phi + 1}{(\phi + 1) + \frac{2}{\sigma}} = 1 + \sigma.
\end{aligned}
$$

In both cases, the competitive ratio is at least $1 + \sigma - \varepsilon$, as claimed. $\square$

6 Lower Bounds for Resource Augmentation

In this section we prove several lower bounds on resource augmentation. Recall the definition of the importance factor $\xi = \max_j w_j / \min_j w_j$. We start with a lower bound showing that there exists no 1-competitive speed-up $O(1)$ algorithm, both for metered and standard tasks. This disproves a conjecture of Koo et al. [8]; it is interesting to note in this context that all the tasks used in the lower bound are tight or have laxity 2, i.e., $d_j - r_j = 2p_j$.

Theorem 6.1 *Both in the metered and standard profit model, any online* 1-*competitive algorithm has speedup at least* $\Omega(\log \log \xi)$*, where* ξ *is the importance factor. In particular, there is no constant speed-up* 1*-competitive algorithm.*

Proof: Fix an integer m. We construct an instance such that any online 1-competitive algorithm needs speed-up $m/2$. See Figure 1 for an illustration. All tasks ending at the same deadline t have the same profit rate $h(t) = (2^m)^t$. The tasks are grouped into $m + 1$ classes numbered $k = 0, \dots, m$. For $k = 0$, the tasks in class 0 are

$$(i, i + 1, 1, h(i + 1)), \quad i = 0, \dots, 2^m - 1.$$

For $k = 1, \dots, m$, the tasks in class k are

$$(i2^k, (i + 1)2^k, 2^{k-1}, h((i + 1)2^k)), \quad i = 0, \dots, 2^{m-k} - 1.$$

For each time t consider the sub-instance consisting of the tasks that are released before t. We claim that the optimal solution of this sub-instance schedules exactly all the tasks with deadline t or later. To prove this claim, note that in the sub-instance, there is exactly one task in each class with deadline at least t. All these task can be scheduled from time 0 to 2^m, completely filling the capacity of the processor at any time: Schedule each such job j in classes 1 to m in that half of the interval $[r_j, d_j)$ which does not contain the interval $[t - 1, t)$; the interval $[t - 1, t)$ is used by the job in class 0. Since all the other tasks in the sub-instance have smaller profit rate, this gives the optimal solution (both for metered and standard tasks).

The weights increase so fast that the profit rate $h(t)$ is at least the total profit of all the tasks with deadlines before t: The total processing time of all tasks

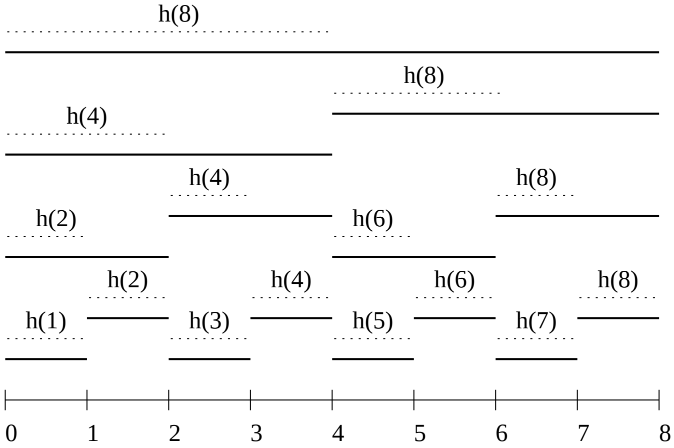

Fig. 1. The instance for $m = 3$. Solid lines represent feasibility ranges, dotted lines represent processing times, and the profit rates $h(x)$ are shown above these lines.

with deadline equal to $t - 1$ is at most 2^{m-1}, thus their total profit is at most $2^{m-1}h(t-1)$. By induction, $h(t-1)$ bounds the total profit of all tasks with deadline before $t-1$. Thus the total profit of all tasks with deadline before t is at most $(2^{m-1}+1)h(t-1) \leq 2^m h(t-1) = h(t)$.

The previous considerations show that to achieve optimal profit, the on-line algorithm has to completely execute all the tasks with deadline t, with the exception of tasks or their parts (in metered model) with processing time bounded by 1 (since only that much can be replaced by the hypothetic profit of tasks with earlier deadlines). Since this holds for any time t, all the tasks must be completed, with a possible exception of tasks with total processing time 2^m. The total processing time of all tasks is $(m+2)2^{m-1}$, so tasks or their parts of total length $m2^{m-1}$ have to be executed by time 2^m. It follows that $\mathcal{A}$ must run at speed at least $m/2$.

Since the deadlines range from 1 to 2^m, the importance ratio is $\xi = (2^m)^{2^m-1}$ and the lower bound is $m/2 = \Omega(\log\log\xi)$. □

Theorem 6.2 *In the standard profit model, there is no online* 1*-competitive algorithm with speed-up* $s < 2$ *for scheduling tight tasks.*

Proof: Let $\mathcal{A}$ be an online 1-competitive algorithm. We show an adversary strategy that, for any given n, forces $\mathcal{A}$ to run at speed $2 - 1/n$. The adversary chooses tasks from among $2n-1$ tasks defined as follows. Task 0 is $(0, n, n, 1)$. For $i = 1, \ldots, n-1$, task i is $(i-1, i, 1, 1)$ and task i' is $(i, n, n-i, n/(n-i))$.

The adversary strategy is this: issue tasks $0, 1, 2, \ldots$, as long as tasks $1, 2, \ldots, i$ are fully processed by $\mathcal{A}$ by time i. If $\mathcal{A}$ fails to fully process task i, the adversary issues task i' and halts. If this happens, the instance contains tasks $0, 1, \ldots, i, i'$ whose optimal profit is is $n+i$. To gain this profit $\mathcal{A}$ needs to process all tasks other than i. Their total length is $2n-1$, so $\mathcal{A}$'s speed must be at least $2 - 1/n$.

If $\mathcal{A}$ processes all tasks $1, ..., n-1$, the instance is $0, 1, \ldots, n-1$ and its maximum profit is n. To achieve this profit, $\mathcal{A}$ must also process task 0. Once again, this means that $\mathcal{A}$'s speed is at least $2 - 1/n$. $\Box$

Theorem 6.3 *In the standard profit model, any online algorithm with m processors for scheduling tight tasks has a competitive ratio of $\Omega(\sqrt[m]{\xi}/m)$ (against a 1-processor optimum), where ξ is the importance ratio.*

Proof: Let M be large constant. Suppose $\mathcal{A}$ has m machines. The adversary chooses tasks from $m+1$ task classes numbered $0, 1, \ldots, m$. The tasks in class i have all equal processing time $p_j = M^{2i}$, profit rate $w_j = M^{-i}$, and profit $w_j p_j = M^i$; their release times are aM^{2i}, for $a = 0, 1, \ldots, M^{2m-2i} - 1$. The importance ratio is $\xi = M^m$.

The adversary strategy is as follows. Since the tasks are tight and we consider the standard model, we can assume that once $\mathcal{A}$ fails to run a task, it never starts it again. If $\mathcal{A}$ stops executing a task j at time t (where t could be d_j) then from time $t+1$ until the deadline of j no tasks from classes $0, 1, \ldots, j-1$ are released. (In other words, a task arrives if at its release time all the active tasks are running; note that these tasks are only from higher classes.) It follows that at each time there exists at least one task that was released but is not being executed by $\mathcal{A}$. At time t, let j_t be such a task from the smallest class.

Let P be the total profit of all the dropped tasks, i.e., tasks not finished in $\mathcal{A}$. We prove that (i) the optimal solution schedules tasks with profit at least $P/(m+1)$, and (ii) the algorithm $\mathcal{A}$ schedules tasks with profit at most $2P/(M-1)$. The bound on the competitive ratio follows.

The proof of (i) is trivial: The dropped tasks in each class are disjoint, so the dropped tasks in one of the classes have weight at least $P/(m+1)$.

Now we prove (ii). If a task j running in $\mathcal{A}$ at some time t is from a lower-numbered class than j_t, we assign it to j_t. A task executed by $\mathcal{A}$ can be assigned to none, one or even more dropped tasks (as j_t may change). Any running task not assigned at all is always from a higher class than the current j_t. At each time, the total profit rate of all such tasks is at most $1/(M-1)$ fraction of the profit rate of j_t. Thus the overall profit of all unassigned completed tasks is at most $P/(M-1)$. Now consider all the executed tasks assigned to a particular dropped task j from class i. From the definition of the sequence it follows that there is at most one such task from each class $i' < i$. Thus their total profit (not profit rate) is at most $1/(M-1)$ fraction of the profit of j. Hence the overall profit of all assigned completed tasks is at most $P/(M-1)$, and (ii) follows. $\Box$

7 Final Comments

As we have seen, the model of metered tasks has very nice mathematical properties, which also makes it very attractive. The main remaining open problem is to determine the best competitive ratio for the metered profit model. The best current bounds show that this ratio between 1.25 and $e/(e-1) \approx 1.5820$, but the gap between these two bounds is still wide. Similarly, in the standard model, we know that the minimum speedup needed to obtain a 1-competitive algorithm is between $\Omega(\log\log\xi)$ and $O(\log\xi)$. It would be interesting to determine the optimal speedup for this problem.

References

[1] Sanjoy Baruah, Gilad Koren, Decao Mao, Bud Mishra, Arvind Raghunathan, Louis Rosier, Dennis Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4:125–144, 1992.

[2] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[3] Ee-Chien Chang and Chee Yap. Competitive online scheduling with level of service. In *Proc. 7th Annual International Computing and Combinatorics Conference*, volume 2108 of *Lecture Notes in Computer Science*, pages 453–462. Springer, 2001.

[4] Francis Y. L. Chin and Stanley P. Y. Fung On-line scheduling with partial job values: Does timesharing or randomization help? Manuscript, 2002. To appear in *Algorithmica.*

[5] Marek Chrobak, Leah Epstein, John Noga, Jiří Sgall, Rob van Stee, Tomáš Tichý, and Nodari Vakhania. Preemptive scheduling in overloaded systems. In *Proc. of the 28th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 2380*, pages 800–811. Springer-Verlag, 2002.

[6] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):214–221, 2000.

[7] Howard Karloff. *Linear Programming.* Birkhäuser, Boston, 1991.

[8] Chiu-Yuen Koo, Tak-Wah Lam, Tsuen-Wan Ngan, and Kar-Keung To. On-line scheduling with tight deadlines. In *Proc. 26th Symp. on Mathematical Foundations of Computer Science*, volume 2136 of *Lecture Notes in Computer Science*, pages 464–473, 2001.

[9] G. Koren and D. Shasha. d^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24:318–339, 1995.

[10] Tak-Wah Lam, Tsuen-Wan Ngan, and Ker-Keung To. On the speed requirement for optimal deadline scheduling in overloaded systems. In *Proc. 15th International Parallel and Distributed Processing Symposium*, page 202, 2001.

[11] Tak-Wah Lam and Ker-Keung To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proc. 10th Symp. on Discrete Algorithms*, pages 755–764, 1999.

[12] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization: An Interior Point Approach.* John Wiley and Sons, Chichester, 1997.

[13] Jiří Sgall. Online scheduling. In *Online Algorithms: The State of Art*, pages 196–227. Springer-Verlag, 1998.