A Note on Scheduling Tall/Small Multiprocessor Tasks with Unit Processing Time to Minimize Maximum Tardiness

Philippe Baptiste and Baruch Schieber
IBM T.J. Watson Research Center
P.O. Box 218, Yorktown Heights, NY 10598
{*baptiste,sbar*}*@us.ibm.com*

February 8, 2009

Abstract

We study the scheduling situation where n tasks, subjected to release dates and due dates, have to be scheduled on m parallel processors. We show that, when tasks have unit processing times and either require 1 or m processors simultaneously, the minimum maximal tardiness can be computed in polynomial time. Two algorithms are described. The first one is based on a linear programming formulation of the problem while the second one is a combinatorial algorithm. The complexity status of this "tall/small" task scheduling problem $P|r_i, p_i = 1, size_i \in \{1, m\}|T_{\max}$ was unknown before, even for 2 processors.

Keywords: Multiprocessor Task Scheduling, Linear Programming, Unit Processing Times

1 Introduction

We study the following scheduling problem. The input consists of n unit processing time tasks that have to be scheduled on m parallel identical processors. Each task i is associated with a release date r_i before which it cannot start and a due date d_i by which it needs to be completed. Each task requires simultaneously a fixed number $size_i$ of processors, yet the processors required are not specified. In the "tall/small" problem, there are two possible sizes, either 1 (small tasks) or m (tall tasks), while in the arbitrary size problem, $size_i \in \{1, \ldots, m\}$. In the following, $\mathcal{T}_1$ and $\mathcal{T}_m$ denote the sets of tasks of size 1 and m, respectively.

Throughout this paper, we consider the maximum tardiness objective function. The tardiness of task i is defined as $T_i = \max\{0, C_i - d_i\}$, where C_i is the completion time of i, and the maximum tardiness is $T_{\max} = \max_i T_i$. Following the classical scheduling notation (see for instance [1, 2]), this problem is referred to as $P|r_i, p_i = 1, size_i \in \{1, m\}|T_{\max}$.

When all release dates are equal, the arbitrary size problem for any *fixed* number of processors, denoted $Pm|p_i = 1, size_i|T_{\max}$, can be solved in polynomial time [3]. Similarly, the problem for any *fixed* number of sizes, denoted $P|p_i = 1, |\{size_i\}| = c|T_{\max}$, can be solved in polynomial time. Both problem can be solved using dynamic programming. However, the arbitrary size problem is NP-Hard in the strong sense [4].

When preemption is allowed, even with arbitrary processing times and release dates, the problem is easy to solve. Existing algorithms are based on a Linear Programming formulation where a variable is associated to each subset of tasks whose total resource requirement is no more than m (see for instance [1]). Unfortunately, there are some instances for which the non-preemptive maximum tardiness is strictly larger than the preemptive maximum tardiness. In the preemptive schedule of Figure 1, $T_{\max} = 0$, while the value of $T_{\max}$ is at least 1 for any non-preemptive schedule.

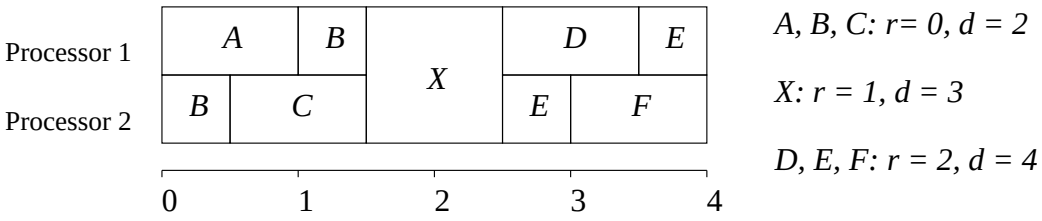

Figure 1: An optimal preemptive schedule.

On two processors, the tall/small problem and the arbitrary size problem are exactly the same and the complexity status of $P2|r_i, p_i = 1, size_i|T_{\max}$ has been open. In

this paper we show that $P|r_i, p_i = 1, size_i \in \{1, m\}|T_{\max}$ can be solved in polynomial time.

Without loss of generality, we can assume that that $\max_{i=1}^{n}\{d_i\} - \min_{i=1}^{n}\{r_i\}$ is $O(n)$. Indeed, consider any schedule built chronologically as follows: schedule at each time t an unscheduled task among $\{i|r_i \leq t\}$. In such a schedule, at most one task is scheduled in each time slot and it is easy to see that if no task is scheduled at t then the tasks can be split in two subsets $\{i : r_i \leq t\}$ and $\{i : r_i > t\}$ that can be scheduled independently. Hence we can assume that $\max_{i=1}^{n}\{r_i\} - \min_{i=1}^{n}\{r_i\} \leq n$. We can also assume that the largest due date is not larger than the largest release date plus n. Hence $\max_{i=1}^{n}\{d_i\} - \min_{i=1}^{n}\{r_i\}$ is $O(n)$.

In the following, we focus on the decision variant of the maximal tardiness problem. For a fixed value of $T_{\max}$, it is easy to compute a deadline $\delta_i = d_i + T_{\max}$ for each task i and a schedule is said to be feasible if tasks are completed by their deadlines. To compute the minimal maximal tardiness, one can find the smallest value of $T_{\max}$ for which there is a feasible schedule. Since one can easily build a feasible schedule with $T_{\max} = n$, there are no more than n values to test. A dichotomic search could be used to reduce the number of iterations. So, the $T_{\max}$ problem can be solved in polynomial time provided that the deadline scheduling problem is solvable in polynomial time.

We describe two independent techniques to solve the tall/small problem. In Section 2, we introduce a linear programming formulation of the problem and in Section 3 we describe a combinatorial algorithm to solve the problem in $O(n^4)$. Finally, we draw some conclusions in Section 4.

2 An LP Formulation

In Section 2.1 we describe some linear constraints that must be met by any feasible schedule. In Section 2.2 we show that if these constraints hold, we can build a preemptive schedule of tall tasks that "implicitly" takes into account the small tasks. This schedule is transformed in Section 2.3 into a non-preemptive schedule of both small and tall tasks.

2.1 Necessary Conditions

To simplify the presentation of the algorithm we will introduce some time indexed variables. Since $\max_{i=1}^{n}\{d_i\} - \min_{i=1}^{n}\{r_i\}$ is $O(n)$, there are few relevant time points and the total number of variables is linear in n. In the following, unless precisely stated, time points and time slots are integral.

Consider a feasible schedule and, for each tall task i and for each integer time point, let t, $x_i^t \in [0,1]$ be the total time during which i executes in the time-slot $[t, t+1)$. Each tall task has to be scheduled somewhere between its release date and its deadline so,

$$\forall i \in \mathcal{T}_m, \sum_{t=r_i}^{\delta_i - 1} x_i^t = 1. \tag{1}$$

On top of that, the total time during which tall tasks are processed in a single time slot does not exceed the size of the time slot, *i.e.*,

$$\forall t, \sum_{i \in \mathcal{T}_m} x_i^t \leq 1. \tag{2}$$

Now let us focus on small tasks. For any time interval $[t_1, t_2)$, and any size $u \in \{1, m\}$, let $\mathcal{T}_u(t_1, t_2)$ be the set of small ($u = 1$) or tall ($u = m$) tasks that have to execute in the time interval $[t_1, t_2)$, *i.e.*,

$$\mathcal{T}_u(t_1, t_2) = \{i \in \mathcal{T}_u : t_1 \leq r_i < \delta_i \leq t_2\}.$$

Note that in a non-preemptive schedule, q small tasks cannot be scheduled in less than $\lceil \frac{q}{m} \rceil$ time units. Since in a time interval $[t_1, t_2)$ there are only $t_2 - t_1 - \sum_{i \in \mathcal{T}_m} \sum_{t=t_1}^{t_2-1} x_i^t$ time units available to schedule tall tasks, we get

$$\forall [t_1, t_2), \sum_{i \in \mathcal{T}_m} \sum_{t=t_1}^{t_2-1} x_i^t + \left\lceil \frac{|\mathcal{T}_1(t_1, t_2)|}{m} \right\rceil \leq t_2 - t_1. \tag{3}$$

Hence, if there is feasible schedule, there is a feasible solution of the Linear Program (4).

$$\begin{cases} \forall i \in \mathcal{T}_m, \sum_{t=r_i}^{\delta_i - 1} x_i^t = 1 \\ \forall t, \sum_{i \in \mathcal{T}_m} x_i^t \leq 1 \\ \forall [t_1, t_2), \sum_{i \in \mathcal{T}_m} \sum_{t=t_1}^{t_2-1} x_i^t + \left\lceil \frac{|\mathcal{T}_1(t_1, t_2)|}{m} \right\rceil \leq t_2 - t_1 \\ \forall i \in \mathcal{T}_m, \forall t, x_i^t \geq 0 \end{cases} \tag{4}$$

In the following, we show that a feasible schedule exists if there is a feasible solution of (4).

2.2 Preemptive Schedule of Tall Tasks

From now on, we assume that tasks $1, \ldots, |\mathcal{T}_m|$ are the tall ones and that they are sorted in non-decreasing order of due dates, *i.e.*, $d_1 \leq \ldots \leq d_{|\mathcal{T}_m|}$.

A solution x of (4), specifies the duration x_i^t a tall task i is scheduled in $[t, t+1)$. To precisely build a preemptive schedule of tall tasks, it remains to decide how pieces of tall tasks are scheduled inside each time slot $[t, t+1)$. Let $\mathcal{S}(x)$ be the schedule where, in each time slot, pieces of tall tasks are scheduled from left to right according to their initial numbering (*i.e.*, in non-decreasing order of deadlines). Now let us consider the solution $\bar{x}$ that lexicographically minimizes the vector of average completion times $(C_1, \ldots, C_{|\mathcal{T}_m|})$, where $C_i = \sum_{t=r_i}^{\delta_i - 1} t \cdot x_i^t$ (*i.e.*, C_1 is minimum, C_2 is minimum given the previous value of C_1, ..., and finally C_n is minimum given the previous values of $C_1, \ldots, C_{n-1}$). The next proposition shows that in any such solution x_i^t is either 0 or 1.

Proposition 1. *In $\mathcal{S}(\bar{x})$ tall tasks are not preempted and they start at integer time points.*

Proof. Let k be the first task for which the proposition does not hold (all tasks with smaller indices are not preempted and start at integer time points). Let $[t, t+1)$ and $[t', t'+1)$ be the time slots in which k respectively starts and is completed in $\mathcal{S}(\bar{x})$. First, we show that in $[t, t+1)$, k is the only tall task to execute. Indeed, if there was another tall task l that executes there, we would have $l > k$ (because of our assumption on k). Therefore, we could exchange a small piece of l that executes in $[t, t+1)$ with a small piece of k that executes in $[t', t'+1)$. In other terms, we could build a vector $\hat{x}$ that equals $\bar{x}$ except for the following values:

$$\begin{cases} \hat{x}_k^t = \bar{x}_k^t + \varepsilon \\ \hat{x}_k^{t'} = \bar{x}_k^{t'} - \varepsilon \\ \hat{x}_l^{t'} = \bar{x}_l^{t'} + \varepsilon \\ \hat{x}_l^t = \bar{x}_l^t - \varepsilon \end{cases},$$

where $\varepsilon = \min\{\bar{x}_k^{t'}, \bar{x}_l^t\} > 0$. In the resulting schedule $\mathcal{S}(\hat{x})$ the average completion time of k is smaller than its average completion time in $\mathcal{S}(\bar{x})$ and task l is completed before its deadline (because $\delta_l \geq \delta_k$). Moreover, the "load" of each time slot $[\tau, \tau+1)$ is the same in both schedule, *i.e.*,

$$\forall \tau, \sum_{i \in \mathcal{T}_m} \bar{x}_i^\tau = \sum_{i \in \mathcal{T}_m} \hat{x}_i^\tau.$$

It follows that $\hat{x}$ is a feasible solution of (4) and it is smaller lexicographically than $\bar{x}$. This contradicts our hypothesis on $\bar{x}$.
Next, we examine what prevents us from obtaining a "lower" feasible solution in the defined lexicographical order by increasing $\bar{x}_k^t$ by ε and decreasing $\bar{x}_k^{t'}$ by ε. The reason for this must be some constraints of (4) that are tight for $\bar{x}$. Constraints (1) do not prevent us from making this exchange. Neither do Constraints (2) since k is the only tall task to execute in time slot t. Now, let us examine Constraints (3). We are going to show that a slightly more complex exchange is possible, hence leading to a contradiction. In the following, the notation $\mathcal{I}(t_1, t_2)$ refers to the constraint (3) over the interval $[t_1, t_2)$. Let Ω be the set of constraints (3) with $t_1 \leq t$ and $t < t_2 \leq t'$ that are tight for $\bar{x}$. It is easy to see that Constraints (3) that do not belong to Ω do not prevent us from increasing $\bar{x}_k^t$. Among the constraints of Ω let us pick one with maximum t_1. Since the constraint is tight, we have

$$\sum_{i \in \mathcal{T}_m} \sum_{t=t_1}^{t_2-1} \bar{x}_i^t = t_2 - t_1 - \left\lceil \frac{|\mathcal{T}_1(t_1, t_2)|}{m} \right\rceil .$$

Hence, $\sum_{i \in \mathcal{T}_m} \sum_{t=t_1}^{t_2-1} \bar{x}_i^t$ takes an integer value and consequently, there is another tall task u that is partially executed between t_1 and t_2. If u were partially executed between t and t_2 then we could exchange a piece of it with the last piece of k that executes in $[t', t'+1)$ (we have $\delta_u \geq \delta_k$ because u is preempted and so the exchange is feasible). We would have decreased the average completion time of k; which would contradict our initial hypothesis. So u is partially executed in a time slot $[\tau, \tau+1)$ between t_1 and t. Let $\tilde{x}$ be the vector that equals $\bar{x}$ except of the following values:

$$\begin{cases} \tilde{x}_u^{\tau} = \bar{x}_u^{\tau} - \varepsilon \\ \tilde{x}_u^{t'} = \bar{x}_u^{t'} + \varepsilon \\ \tilde{x}_k^{t'} = \bar{x}_k^{t'} - \varepsilon \\ \tilde{x}_k^{t} = \bar{x}_k^{t} + \varepsilon \end{cases} ,$$

for a small positive value ε. Note that a piece of u can be scheduled at t' because $\delta_u \geq \delta_k$. Moreover, the only constraints that could be violated by the exchange are those in the set Ω (the value of $\tilde{x}_k^t$ is consistent with (2) because k is the only tall task that executes in $[t, t+1)$). Let $\mathcal{I}(t_1', t_2')$ be a violated constraint of Ω. Because of our hypothesis on t_1, we have $t_1' \leq t_1$. Hence, it is easy to verify that the load induced by the tall tasks between t_1' and t_2' does not increase. Therefore, the resulting schedule $\mathcal{S}(\tilde{x})$ is lower than $\mathcal{S}(\bar{x})$ in the lexicographical order. This contradicts our hypothesis. □

2.3 From Preemptive to Non-Preemptive Schedules

In Section 2.2, we have shown that there is a solution $\bar{x}$ of (4) such that in $\mathcal{S}(\bar{x})$, tall tasks are not preempted and start at integer time points. In Proposition 2 we show that small tasks can be scheduled in $\mathcal{S}(\bar{x})$ too.

Proposition 2. *Small tasks can be scheduled in* $\mathcal{S}(\bar{x})$.

Proof. Let us sort small tasks in non-decreasing order of deadlines and let us add them one after the other into $\mathcal{S}(\bar{x})$. Each time, the current task starts at the first time point after its release date where one processor is available. Note that because tall tasks are not preempted and start at integer time points, small tasks are not preempted either and also start at integer time points.
Suppose that we do not obtain a feasible schedule. Let then k be the first small task that is completed after its deadline and let t be the earliest time point such that all processors are full between t and δ_k. Note that since k is not completed by its deadline $t \leq r_k$. Let Ψ be the set of small tasks that are scheduled in $[t, \delta_k)$. Tasks in Ψ have a release date greater than or equal to t (otherwise they would be scheduled in $[t-1, t)$) and a deadline smaller than or equal to δ_k (because tasks are sorted in non-decreasing order of deadlines). Since all processors are full, there are exactly $m(\delta_k - t - \sum_{i \in \mathcal{T}_m} \sum_{t'=t}^{\delta_k - 1} \bar{x}_i^{t'})$ tasks in Ψ. On top of that, $\Psi \subset \mathcal{T}_1(t, \delta_k)$ and $k \notin \Psi$ but $k \in \mathcal{T}_1(t, \delta_k)$. Hence,

$$\mathcal{T}_1(t, \delta_k) > m(\delta_k - t - \sum_{i \in \mathcal{T}_m} \sum_{t'=t}^{\delta_k - 1} \bar{x}_i^{t'}).$$

This contradicts (3). □

3 A Combinatorial Algorithm

For each time interval $[t_1, t_2)$, we define recursively the *slack* of the interval, denoted $\Delta(t_1, t_2)$. We show later that there exists a feasible schedule if and only if all the intervals have non-negative slack. We also show how to use the slack to actually construct a schedule is such exists. Intuitively, $\Delta(t_1, t_2)$ is an upper bound on the number of *free* time slots in $[t_1, t_2)$ in any schedule of the tasks in $\mathcal{T}(t_1, t_2)$. (For a given schedule a free time slot is a time unit in which all the m processors are idle.) To define the slack we define *static* slack and *dynamic* slack. In the recursive definition the computation the dynamic slack (and thus the slack) of an interval $[t_1, t_2)$ uses the values of the slack of intervals contained in $[t_1, t_2)$.

Definition 1. *For any interval* $[t_1, t_2)$*, with* $t_2 > t_1 + 1$*, the slack* $\Delta(t_1, t_2)$ *is the minimum of the static slack* $\Delta_s(t_1, t_2)$ *and of the dynamic slack* $\Delta_d(t_1, t_2)$*. For any interval* $[t_1, t_1 + 1)$*,* $\Delta(t_1, t_1 + 1) = \Delta_s(t_1, t_1 + 1)$*.*

Definition 2. *The static slack over* $[t_1, t_2)$*,* $\Delta_s(t_1, t_2)$*, equals*

$$t_2 - t_1 - \left(|\mathcal{T}_m(t_1, t_2)| + \left\lceil \frac{|\mathcal{T}_1(t_1, t_2)|}{m} \right\rceil \right)$$

Definition 3. *The dynamic slack over* $[t_1, t_2)$*, with* $t_2 > t_1 + 1$*,* $\Delta_d(t_1, t_2)$*, is the minimum of*

$$\Delta(t_1, t'_2) + \Delta(t'_1, t_2) - |\mathcal{T}_m(t_1, t_2) \setminus (\mathcal{T}_m(t_1, t'_2) \cup \mathcal{T}_m(t'_1, t_2))|$$

over t'_1, t'_2 *with* $t_1 < t'_1 \leq t'_2 < t_2$*.*

We claim that all slacks can be computed in $O(n^4)$. Indeed, there are $O(n)$ time points to consider and thus there are $O(n^2)$ values $\Delta(t_1, t_2)$ to compute. We observe that all the values of $|\mathcal{T}_1(t_1, t_2)|$ and $|\mathcal{T}_m(t_1, t_2)|$ can be computed inductively in $O(n^2)$ time. To see this, note that all the $O(n)$ values $|\mathcal{T}_1(t_1, t_1 + 1)|$ and $|\mathcal{T}_m(t_1, t_1 + 1)|$ can be computed in linear time. Now, for any $x > 1$, and for $i \in \{1, m\}$,

$$|\mathcal{T}_i(t_1, t_1 + x)| = |\mathcal{T}_i(t_1, t_1 + x - 1)| + |\mathcal{T}_i(t_1 + 1, t_1 + x)| - |\mathcal{T}_i(t_1 + 1, t_1 + x - 1)|.$$

Thus, given all values $|\mathcal{T}_1(t_1, t_1 + y)|$ and $|\mathcal{T}_m(t_1, t_1 + y)|$, for $y < x$, $|\mathcal{T}_1(t_1, t_1 + x)|$ and $|\mathcal{T}_m(t_1, t_1 + x)|$ can be computed in linear time. Clearly, this implies that all static slacks can be computed in $O(n^2)$ time. For each of the $O(n^2)$ computations of the dynamic slack we need to find the minimum of $O(n^2)$ quantities. Since

$$|\mathcal{T}_m(t_1, t_2) \setminus (\mathcal{T}_m(t_1, t'_2) \cup \mathcal{T}_m(t'_1, t_2))| = |\mathcal{T}_m(t_1, t_2)| - |\mathcal{T}_m(t_1, t'_2)| - |\mathcal{T}_m(t'_1, t_2)| + |\mathcal{T}_m(t'_1, t'_2)|,$$

each of these $O(n^2)$ quantities can be computed in constant time totalling to $O(n^4)$ time for the whole computation.

The next proposition shows that if there is a feasible schedule, then all intervals have non-negative slack.

Proposition 3. *There are at most* $\Delta(t_1, t_2)$ *idle time slots between* t_1 *and* t_2 *in any feasible schedule of* $\mathcal{T}(t_1, t_2)$*.*

Proof. Consider an interval $[t_1, t_2)$ such that there is a feasible schedule of $\mathcal{T}(t_1, t_2)$ with more than $\Delta(t_1, t_2)$ idle time slots and assume that $t_2 - t_1$ is minimal among such intervals. There are $|\mathcal{T}_m(t_1, t_2)|$ tall tasks and $|\mathcal{T}_1(t_1, t_2)|$ small tasks and at least $\left(|\mathcal{T}_m(t_1, t_2)| + \left\lceil \frac{|\mathcal{T}_1(t_1,t_2)|}{m} \right\rceil\right)$ time units are required to schedule these tasks. It follows that the number of idle time slots in a feasible schedule is no more than the static slack $\Delta_s(t_1, t_2)$. Hence the slack equals the dynamic slack; that is, $\Delta(t_1, t_2) = \Delta_d(t_1, t_2)$. Suppose that $\Delta_d(t_1, t_2)$ is determined by the two time points t'_1 and t'_2; that is, $\Delta(t_1, t_2) = \Delta(t_1, t'_2) + \Delta(t'_1, t_2) - |\mathcal{T}_m(t_1, t_2) \setminus (\mathcal{T}_m(t_1, t'_2) \cup \mathcal{T}_m(t'_1, t_2))|$. Consider the feasible schedule $\mathcal{S}$ of $\mathcal{T}(t_1, t_2)$ and let $\mathcal{S}_r$ and $\mathcal{S}_l$ be the restriction of $\mathcal{S}$ to the tasks of $\mathcal{T}(t_1, t'_2)$ and to the tasks of $\mathcal{T}(t'_1, t_2)$. The number of idle time slots in $\mathcal{S}$ is less than or equal to the number of those in $\mathcal{S}_r$ plus the number of those in $\mathcal{S}_l$. Given our hypothesis on $t_2 - t_1$ this is lower than or equal to $\Delta(t_1, t'_2) + \Delta(t'_1, t_2)$. On top of that, large tasks in $\mathcal{T}_m(t_1, t_2) \setminus (\mathcal{T}_m(t_1, t'_2) \cup \mathcal{T}_m(t'_1, t_2))$ are not scheduled in $\mathcal{S}_l$ nor in $\mathcal{S}_r$ so the number of idle time slots in $\mathcal{S}$ is less than or equal to

$$\Delta(t_1, t'_2) + \Delta(t'_1, t_2) - |\mathcal{T}_m(t_1, t_2) \setminus (\mathcal{T}_m(t_1, t'_2) \cup \mathcal{T}_m(t'_1, t_2))|,$$

which equals $\Delta(t_1, t_2)$. This contradicts our initial hypothesis. □

The more complicated part is to show that if there is no feasible schedule then there must be at least one interval with negative slack. In order to prove this we first prove two propositions. In the proofs we consider several instances simultaneously. Given an instance I, $\Delta(I, t_1, t_2)$, $\mathcal{T}(I, t_1, t_2)$, $\mathcal{T}_1(I, t_1, t_2)$ and $\mathcal{T}_m(I, t_1, t_2)$ denote the slack and the sets associated to I for a given interval $[t_1, t_2)$. From now on, to simplify notation we also assume that the smallest release time is 0.

Proposition 4. *Consider an instance I, and let J be and instance given by removing a tall task i, with $r_i = 0$, if such exists. For all $t < \delta_i$, $\Delta(J, 0, t) = \Delta(I, 0, t)$, and for all $t \geq \delta_i$, $\Delta(J, 0, t) = \Delta(I, 0, t) + 1$.*

Proof. The first part follows directly from the definition of slack. Consider the second part and to obtain a contradiction assume that the equality does not hold. Let t be the minimum time such that that $\Delta(J, 0, t) \neq \Delta(I, 0, t) + 1$. We first show that $\Delta(J, 0, t) \leq \Delta(I, 0, t) + 1$. We consider two cases.
Case 1: $\Delta(I, 0, t) = \Delta_s(I, 0, t)$. Since $\Delta_s(J, 0, t) = \Delta_s(I, 0, t)+1$, we have $\Delta(J, 0, t) \leq \Delta(I, 0, t) + 1$.
Case 2: $\Delta(I, 0, t) = \Delta_d(I, 0, t)$. Let t_1, t_2 be two time points that determine $\Delta_d(I, 0, t)$. Note that $\Delta(I, t_1, t) = \Delta(J, t_1, t)$ (because $0 < t_1$). If $\delta_i \leq t_2$ then i

does not belong to $\mathcal{T}_m(I,0,t) \setminus (\mathcal{T}_m(I,0,t_2) \cup \mathcal{T}_m(I,t_1,t))$. Hence, the slack of J is not larger than

$$\Delta(J,0,t_2) + \Delta(I,t_1,t) - |\mathcal{T}_m(I,0,t) \setminus (\mathcal{T}_m(I,0,t_2) \cup \mathcal{T}_m(I,t_1,t))|\,.$$

Since $t_2 < t$, by our assumption $\Delta(J,0,t_2) = \Delta(I,0,t_2) + 1$. Hence, $\Delta(J,0,t) \leq \Delta(I,0,t) + 1$. If $\delta_i > t_2$ then

$$\begin{aligned}\Delta(J,0,t) &\leq \Delta(I,0,t_2) + \Delta(I,t_1,t) - (|\mathcal{T}_m(I,0,t) \setminus (\mathcal{T}_m(I,0,t_2) \cup \mathcal{T}_m(I,t_1,t))| - 1)\\ &= \Delta(I,0,t) + 1.\end{aligned}$$

Similarly, it can be shown that $\Delta(I,0,t) \leq \Delta(J,0,t) - 1$, by considering the two cases $\Delta(J,0,t) = \Delta_s(J,0,t)$ and $\Delta(J,0,t) = \Delta_d(J,0,t)$. □

Proposition 5. *Consider an instance I. Let k be the number of small tasks in I with release date 0. Let J be the instance derived from I by removing $\min\{k,m\}$ small tasks with the smallest deadline among tasks with release date 0, and by changing the release date of the rest of the tasks with release date 0 (if any) to 1. Let x be a tall task in I with the earliest deadline among tall tasks with release date 0 (if such exists), and let t be δ_x if x is exists and ∞ otherwise. For all $1 < t_2 < t$,*

$$\min\{\Delta(I,0,t_2), \Delta(I,1,t_2)\} \leq \Delta(J,1,t_2).$$

Proof. To obtain a contradiction assume that t_2 is the minimum time for which the inequality does not hold. If $k \leq m$ then since no tall task in I has release date 0 and deadline at most t_2, we have $\Delta(I,1,t_2) = \Delta(J,1,t_2)$; a contradiction. Suppose that $k > m$. Let δ be the earliest deadline of a task in I with release date 0 that was not removed. If $t_2 < \delta$ then again $\mathcal{T}(J,1,t_2)$ does not contain any task with release time 0 in I, and thus, $\Delta(I,1,t_2) = \Delta(J,1,t_2)$; a contradiction. The only remaining case is $t_2 \geq \delta$. We consider two cases.

Case 1: $\Delta(J,1,t_2) = \Delta_s(J,1,t_2)$. Since $t_2 \geq \delta$ $|\mathcal{T}_1(J,1,t_2)| = |\mathcal{T}_1(I,0,t_2)| - m$ and thus

$$\left\lceil \frac{|\mathcal{T}_1(J,1,t_2)|}{m} \right\rceil = \left\lceil \frac{|\mathcal{T}_1(I,0,t_2)|}{m} \right\rceil - 1.$$

Recall also that since no tall task in I has release date 0 and deadline at most t_2 we have for all $1 < s \leq t_2$, $\mathcal{T}_m(I,0,s) = \mathcal{T}_m(J,1,s)$. We get

$$\begin{aligned}\Delta(J,1,t_2) &= t_2 - 1 - (|\mathcal{T}_m(J,1,t_2)| + \left\lceil \frac{|\mathcal{T}_1(J,1,t_2)|}{m} \right\rceil)\\ &= t_2 - (|\mathcal{T}_m(I,0,t_2)| + \left\lceil \frac{|\mathcal{T}_1(I,0,t_2)|}{m} \right\rceil)\\ &= \Delta_s(I,0,t_2) \geq \Delta(I,0,t_2).\end{aligned}$$

Case 2: $\Delta(J,1,t_2) = \Delta_d(J,1,t_2)$. Let t'_1, t'_2 with $1 < t'_1 \leq t'_2 < t_2$ be time points that determine the dynamic slack. By our assumption $\min\{\Delta(I,0,t'_2), \Delta(I,1,t'_2)\} \leq \Delta(J,1,t'_2)$. Since $t'_1 > 1$, $\Delta(I,t'_1,t_2) = \Delta(J,t'_1,t_2)$. We get

$$\begin{aligned}
\Delta(J,1,t_2) &= \Delta(J,1,t'_2) + \Delta(J,t'_1,t_2) \\
&\quad - |\mathcal{T}_m(J,1,t_2) \setminus (\mathcal{T}_m(J,1,t'_2) \cup \mathcal{T}_m(J,t'_1,t_2))| \\
&\geq \min\{\Delta(I,0,t'_2), \Delta(I,1,t'_2)\} + \Delta(I,t'_1,t_2) \\
&\quad - |\mathcal{T}_m(I,0,t_2) \setminus (\mathcal{T}_m(I,0,t'_2) \cup \mathcal{T}_m(I,t'_1,t_2))| \\
&\geq \min\{\Delta(I,0,t_2), \Delta(I,1,t_2)\}.
\end{aligned}$$

□

Proposition 6. *If all time intervals have non-negative slack then there is a feasible schedule.*

Proof. We prove that if there is no feasible schedule then there must be an interval with a negative slack. Define the *size* of an instance I as $\sum_{\mathcal{T}} \delta_i - r_i$. We prove it by induction on the size of the instance. The proposition clearly holds for instances of size 1 since such instances consist of one task. Suppose that the proposition holds for all instances of size at most s. We prove it for instances of size $s+1$. Consider an instance I of size $s+1$ with no feasible schedule. If I has both a tall task and a small task with release date 0 and deadline 1 or more than m small tasks with release time 0 and deadline 1 then $\Delta(I,0,1) < 0$. Suppose that this is not the case. Let k be the number of small tasks in I with release date 0. Consider two instances J_S and J_T defined as follows. The instance J_S is derived from I by removing $\min\{k,m\}$ small tasks with the smallest deadline among tasks with release date 0, and by changing the release date of the rest of the tasks with release date 0 (if any) to 1. The instance J_T is derived from I by removing a tall task x with the smallest deadline among tall tasks with release date 0 (assuming that the task x exists), and by changing the release date of the rest of the tasks with release date 0 (if any) to 1. Note that if any of J_T and J_S has a feasible schedule then this schedule can be extended to a feasible schedule for I. We conclude that both J_S and J_T have no feasible schedule. Since the size of both J_S and J_T is at most s, there is an interval with negative slack in both J_S and J_T. If for $J \in \{J_S, J_T\}$, there exists $1 < t_1 < t_2$ such that $\Delta(J,t_1,t_2) < 0$ then since $\Delta(I,t_1,t_2) = \Delta(J,t_1,t_2)$ we are done. Suppose that this is not the case. Hence, there are $t_S, t_T > 1$ such that $\Delta(J_S,1,t_S) < 0$ and $\Delta(J_T,1,t_T) < 0$. We consider several cases.

Case 1: $t_T \geq \delta_x$. Let I' be the instance given by removing task x from I. Since the sets of tasks in J_T and I' are the same $\Delta(I', 0, t_T) \leq \Delta(J_T, 1, t_T)+1$. By Proposition 4 $\Delta(I, 0, t_T) = \Delta(I', 0, t_T) - 1$. Hence, $\Delta(I, 0, t_T) \leq \Delta(J_T, 1, t_T) < 0$.
Case 2: $t_S < \delta_x$. By Proposition 5 $\min\{\Delta(I, 0, t_S), \Delta(I, 1, t_S)\} \leq \Delta(J_S, 1, t_S) < 0$.
Case 3: The remaining case is $t_S \geq \delta_x$ and $t_T < \delta_x$. Let l be the number of large tasks in I with release date 0 and deadline at most t_S. Let J'_S be the instance given by removing these l tasks from J_S. By proposition 4 $\Delta(J'_S, 1, t_S) = \Delta(J_S, 1, t_S) + l < l$. Let I' be the instance given by removing these l tasks from I. Note that $\Delta(I, 1, t_S) = \Delta(I', 1, t_S)$. By Proposition 5 $\min\{\Delta(I', 0, t_S), \Delta(I', 1, t_S)\} \leq \Delta(J'_S, 1, t_S) < l$. If $\Delta(I', 0, t_S) < l$ then by Proposition 4 $\Delta(I, 0, t_S) = \Delta(I', 0, t_S) - l < 0$. Otherwise, $\Delta(I, 1, t_S) < l$. We get that

$$\begin{aligned} \Delta(I, 0, t_S) &\leq \Delta(I, 0, t_T) + \Delta(I, 1, t_S) - |\mathcal{T}_m(I, 0, t_S) \setminus (\mathcal{T}_m(I, 0, t_T) \cup \mathcal{T}_m(I, 1, t_S))| \\ &< 0 + l - l = 0 \end{aligned}$$

To conclude the proof, we just have to consider the situation where the task x does not exist (*i.e.*, there is no tall task with release date 0). In such a case, the same reasoning as for Case 2 applies. □

Finally, note that the proof of Proposition 6 implies also an algorithm for constructing a schedule. Given an instance I we determine the schedule at time 0 by testing the feasibility of J_S and J_T, and continue in the same manner to construct the rest of the schedule.

4 Conclusion

We have presented two algorithms for scheduling tall/small multiprocessor tasks with unit processing time to minimize maximum tardiness. The first one relies on linear programming and the second one is purely combinatorial. An interesting open question is whether one of them could be extended to solve a larger class of problem such as $Pm|r_i, p_i = 1, size_i|T_{\max}$.

Acknowledgment

The authors would like to thank Professor Peter Brucker who introduced the first author to the small/tall problem in 1998.

References

[1] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer, 1996.

[2] P. Brucker. *Scheduling Algorithms, 3rd edition*. Springer, 2001.

[3] P. Brucker and A. Krämer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90:214–226, 1996.

[4] E. L. Lloyd. Concurrent task systems. *Operations Research* 29:189–201, 1981.