RC22179 (W0109-034) September 19, 2001
Computer Science

IBM Research Report

A note on scheduling tall/small multiprocessor tasks with unit processing time to minimize maximum tardiness

Philippe J. Baptiste
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

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

Philippe Baptiste
IBM T. J. Watson Research Center
Mathematical Sciences, Office 35-214
P. O. Box 218, Yorktown Heights, NY 10598
baptiste@us.ibm.com

September 19, 2001

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. 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 scheduling situation where n tasks with unit processing times 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 before which one would like it 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 no more than 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 $\mathcal{T}_m$ respectively denote the sets of tasks of size 1 and m.

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 J_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 can be solved in polynomial time for any fixed number of different sizes [3]. Hence, $Pm|p_i = 1, size_i|T_{\max}$ and $P|p_i = 1, size_i \in \{1, m\}|T_{\max}$ can be solved in polynomial time. When m is part of the input data, 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 less 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.

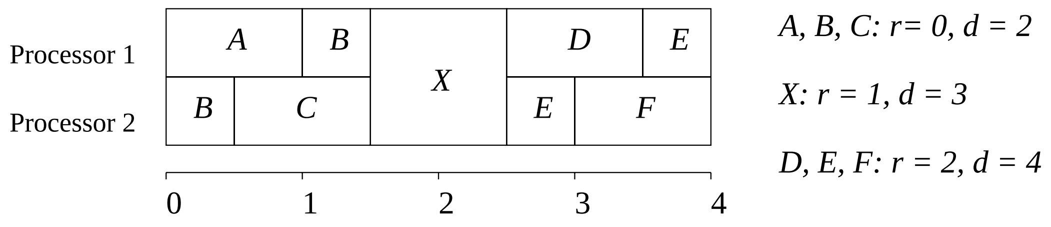

Figure 1: An optimal preemptive schedule.

On two machines, the tall/small problem and the arbitrary size problem are exactly the same and the complexity status of $P2|r_i, p_i = 1, size_i|T_{\max}$ is still 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 by Linear Programming.

In Section 2 we focus on the decision version of the problem and we describe some linear constraints that must be met by any feasible schedule. In Section 3 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 4 into a non-preemptive schedule of both small and tall tasks. Finally we draw some conclusions in Section 5.

2 Necessary Conditions

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 before 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.

To simplify the presentation of the algorithm we will introduce some time indexed variables. There are few relevant time points so the total number of variables remains polynomial in n. Indeed, we can assume that the distance between two consecutive release dates r_x, r_y is not larger than n (otherwise, the jobs could be split in two subsets $\{i : r_i \leq r_x\}$ and $\{i : r_i \geq r_y\}$ that could be scheduled independently). On top of that, we can also assume that the largest deadline is not larger than the largest release date plus n. Thanks to these assumptions, we have a polynomial number of relevant integer time points t to consider. 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 $[t_1, t_2)$, let $\mathcal{T}_1(t_1, t_2)$ be the set of small tasks that have to execute in the time interval $[t_1, t_2)$, *i.e.*,

$$\mathcal{T}_1(t_1, t_2) = \{i \in \mathcal{T}_1 : t_1 \leq r_i \leq \delta_i \leq t_2\}.$$

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

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

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).

3 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 during which the 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 completion times $(C_1, \ldots, C_{|\mathcal{T}_m|})$.

Proposition 1. *On $\mathcal{S}(\bar{x})$, tall tasks are not interrupted 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 interrupted 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 were another tall task l that would execute 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 $\mathcal{S}(\hat{x})$, k is completed earlier than 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.*,

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

So, $\hat{x}$ is a feasible solution of (4) and it is better than $\bar{x}$ for the lexicographical criterion. This contradicts our hypothesis on $\bar{x}$.

Second, note that some constraints of (4) must be tight for $\bar{x}$ and prevent us to increase $\bar{x}_k^t$ of ε and decrease $\bar{x}_k^{t'}$ of ε. Indeed, if we could perform this exchange, we would obtain a "better" feasible solution for the lexicographical criterion. Constraints (1) do not prevent us to make 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. In the following, the notation (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 to increase $\bar{x}_k^t$. Among the constraints of Ω let us pick one with maximum t_1. Since the constraint is tight, we have

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

Hence, $\sum_{i \in \mathcal{T}_m} \sum_{t=t_1}^{t_2-1} x_i^t$ 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 interrupted and so the exchange is feasible). We would decrease the 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 for the following values:

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

Where ε is a small positive value. 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 there is k is the only task that executes in $[t, t+1)$). Let (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. $\square$

4 From Preemptive to Non-Preemptive Schedules

In Section 3, we have shown that there is a solution $\bar{x}$ of (4) such that in $\mathcal{S}(\bar{x})$, tall tasks are not interrupted 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 tall tasks are not interrupted and start at integer time points, small tasks are not interrupted either and also start at integer time points.

Let k be the first small task that is completed after its deadline and let t be the earliest time point lower than or equal to r_k such that all processors are full between t and 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 $2(\delta_k - t - \sum_{i \in \mathcal{T}_m} \sum_{t'=t}^{\delta_k - 1} x_i^t)$ 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,

$$\mathcal{T}_1(t, \delta_k) > 2(\delta_k - t - \sum_{i \in \mathcal{T}_m} \sum_{t'=t}^{\delta_k - 1} x_i^t).$$

This contradicts (3). $\square$

5 Conclusion

We have presented a polynomial algorithm for scheduling tall/small multiprocessor tasks with unit processing time to minimize maximum tardiness. Our approach relies on a Linear Programming formulation with implicit constraints. An interesting open question is whether the formulation presented in this paper could be extended to solve a larger class of problem such as $P|r_i, p_i = 1, size_i|T_{\max}$.

Acknowledgment

The author would like to thank Professor Peter Brucker who introduced the 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.