International Symposium on Scheduling 2004
May 24-26, 2004, Awaji-Yumebutai, Japan
2B4

PREEMPTIVE SCHEDULING ON PARALLEL PROCESSORS WITH DUE DATES

Yakov Zinder
Department of Mathematical Sciences
University of Technology, Sydney
P O Box 123, Broadway, NSW, 2007, Australia
Yakov.Zinder@uts.edu.au

Gaurav Singh
School of Quantitative Methods and Mathematical Sciences
University of Western Sydney
Locked Bag 1797, Penrith South DC, NSW, 1797, Australia
ga.singh@uws.edu.au

Abstract

A priority algorithm is presented for the maximum lateness problem with parallel identical processors, precedence constraints, and preemptions. The algorithm calculates a task's priority by constructing a schedule for the set of its successors. It is shown that the presented algorithm constructs an optimal schedule for the problem with two processors and arbitrary precedence constraints, and for the problem with an arbitrary number of processors and precedence constraints in the form of an in-tree. The proof also indicates that this algorithm allows the best worst-case ratio currently known for the problems with precedence constraints.
Keywords: parallel identical processors, precedence constraints, maximum lateness, preemptions.

1. Introduction

The priority algorithm presented in this paper is intended for the maximum lateness problem which can be stated as follows. A set $N = \{1, 2, \ldots, n\}$ of n tasks (jobs, operations) is to be processed on $m > 1$ identical parallel processors (machines) subject to precedence constraints in the form of an anti-reflexive, anti-symmetric and transitive relation on N. If task i precedes task j, denoted $i \rightarrow j$, then the processing of i must be completed before the processing of j begins. All processors are available from time $t = 0$. Each processor can process at most one task at a time, and each task can be processed by any processor. The processing time of task j is denoted by p_j and is an arbitrary real number. The processing of any task can be interrupted at any time and resumed later on the same or another processor, i.e. the preemptions are allowed.

A schedule s is a vector function $s(t) = (s_1(t), \ldots, s_n(t))$, where each $s_j(t)$ is a continuous from the left piece-wise constant function, which has a finite number of points of discontinuity, and is equal to 1 if task j is processed at time t, and is equal to 0 otherwise. The completion time of task j in schedule s, denoted by $C_j(s)$, is the smallest τ satisfying the equality $\int_0^\tau s_j(t)dt = p_j$. It is necessary to find a schedule, which minimizes the criterion of maximum lateness

$$L_{max}(s) = \max_{j \in N}[C_j(s) - d_j],$$

where d_j is a due date associated with task j.

The problem above is usually denoted by $P|prec, prmp|L_{max}$, where P specifies that the tasks are to be processed on several parallel identical processors, *prec* indicates the presence of precedence constraints, and the term *prmp* indicates that the preemptions are allowed. The non-preemptive counterpart of $P|prec, prmp|L_{max}$ is the problem $P|prec, p_j = 1|L_{max}$, which differs from $P|prec, prmp|L_{max}$ only by two assumptions: all tasks have the same processing time of one time unit (indicated by the term $p_j = 1$) and the preemptions are not allowed, i.e. if a processor begins executing a task, then it continues processing until the completion of this task. If all due dates are equal to zero, then $P|prec, prmp|L_{max}$ and $P|prec, p_j = 1|L_{max}$ convert into the makespan problems $P|prec, prmp|C_{max}$ and $P|prec, p_j = 1|C_{max}$ with the criterion

$$C_{max}(s) = \max_{j \in N} C_j(s).$$

Although $P|prec, prmp|C_{max}$ and $P|prec, p_j = 1|C_{max}$ are NP-hard (Ullman, 1975), and therefore more general $P|prec, prmp|L_{max}$ and $P|prec, p_j = 1|L_{max}$ problems are also NP-hard, there are several important particular cases allowing polynomial-time algorithms. Thus, the Brucker-Garey-Johnson algorithm (Brucker, *et al.*, 1977) solves the

problem $P|in-tree, p_j = 1|L_{max}$, where the term *in-tree* indicates that the partially ordered set of tasks in $P|prec, p_j = 1|L_{max}$ is restricted to an in-tree. The Garey-Johnson algorithm (Garey and Johnson, 1976) solves the problem $P2|prec, p_j = 1|L_{max}$, where the term $P2$ specifies that the number of processors in $P|prec, p_j = 1|L_{max}$ is restricted to two. Both $P2|prec, p_j = 1|L_{max}$ and $P|in-tree, p_j = 1|L_{max}$ are also amenable for the algorithm presented in (Zinder and Roper, 1998). In the case of arbitrary number of processors the algorithm presented in (Zinder and Roper, 1998) satisfies the following performance guarantee

$$L_{max}(s') \leq \left(2 - \frac{2}{m}\right) L_{max}(s^*) + \left(1 - \frac{2}{m}\right) \max_{a \in N} d_a - r(m), \tag{1}$$

where

$$r(m) = \begin{cases} \frac{m-3}{m} & \text{for } m \text{ odd} \\ \frac{m-2}{m} & \text{for } m \text{ even,} \end{cases}$$

s' is a schedule constructed by this algorithm, and s^* is an optimal schedule for the $P|prec, p_j = 1|L_{max}$ problem. In particular case, when all due dates are equal to zero, this performance guarantee coincides with the best currently known performance guarantee for the $P|prec, p_j = 1|C_{max}$ problem obtained in (Braschi and Trystram, 1994) for the algorithm of Coffman and Graham (1972). On the other hand, both classical algorithms presented in (Brucker, *et al.*, 1977) and (Garey and Johnson, 1976) violate (1). The worst-case performance of the algorithm from (Brucker, *et al.*, 1977) was analyzed in (Singh and Zinder, 2000b). Examples can also be presented to show that there are arbitrary large instances of the maximum lateness problem satisfying the inequality

$$L_{max}(s') > \left(2 - \frac{2}{m}\right) L_{max}(s^*) + \left(1 - \frac{2}{m}\right) \max_{a \in N} d_a, \tag{2}$$

where s' is a schedule constructed by the algorithm from (Garey and Johnson, 1976) and s^* is an optimal schedule for the maximum lateness problem

The above observation can be viewed as a motivation for the development of a preemptive counterpart of the algorithm presented in (Zinder and Roper, 1998) and its comparison with the preemptive versions of the algorithms in (Brucker, *et al.*, 1977), (Garey and Johnson, 1976), and (Coffman and Graham, 1972), described in (Lawler, 1982) and (Muntz and Coffman, 1969;1970). In particular, the preemptive versions of the algorithms in (Brucker, *et al.*, 1977) and (Garey and Johnson, 1976) solve $P|in-tree, prmp|L_{max}$ and $P2|prec, prmp|L_{max}$, respectively. It will be shown that the algorithm presented in this paper also constructs an optimal schedule for both problems. Moreover, the corresponding proof shows that this algorithm has also a good worst-case performance. More precisely, usually performance guarantees for the maximum lateness problem are given in the form

$$L_{max}(s') \leq \gamma L_{max}(s^*) + (\gamma - 1) \max_{a \in N} d_a - \delta,$$

where s' is a schedule constructed by the considered algorithm, s^* is an optimal schedule, and γ and δ are constants (see for example (1)). Correspondingly, for the makespan problem

$$C_{max}(s') \leq \gamma C_{max}(s^*) - \delta.$$

To the authors knowledge, for the problems with preemptions and precedence constraints, the best currently known value of γ is $2 - \frac{2}{m}$. This γ was obtained in (Lam and Sethi, 1977) for the Muntz-Coffman algorithm (1969,1970). The algorithm presented in this paper also allows this γ. The worst-case analysis of the preemptive version of the Brucker-Garey-Johnson algorithm can be found in (Singh and Zinder, 2000a).

The algorithm presented in this paper assigns to each task j some value μ_j and calculates the task's priority as a sum of the remaining processing time and μ_j. The tasks are assigned to processing according to their priorities. The main idea of the considered approach is to calculate μ_j by constructing a schedule for the set of the successors of task j. Each such schedule as well as the resultant schedule is constructed by a priority algorithm described in Section 2. An iterative procedure calculating μ_j, for each $j \in N$, is presented in Section 3. Section 4. is concerned with the analysis of the algorithm.

2. Priority Algorithm

The priority algorithm described below will be used for constructing a schedule for the entire partially ordered set of tasks as well as schedules for partially ordered sets induced by different subsets of N. Let $M \subseteq N$ be an arbitrary subset of N and suppose that a non-negative number μ_j is associated with each task $j \in M$. Consider the partially ordered set of tasks induced by M, i.e. for a moment we ignore all tasks that do not belong to M and preserve all precedence constraints which exist between tasks from M. We will refer to any schedule for this partially ordered set simply as a schedule for M. In constructing a schedule for M, the priority algorithm determines an increasing sequence of points in time, which will be referred to as points of allocation. The first point of allocation is $t = 0$. At each point of allocation, the algorithm selects tasks which will be processed in the time interval between this and the next point of allocation; determines the amount of processing time, which each of these tasks will receive in this interval; and determines the next point of allocation.

The selection of tasks for processing and the allocation of processing times are based on the tasks priorities. The priority of any task j at time $t = 0$ is $p_j + \mu_j$. For any other point in time, the priorities are calculated as follows. Let s be a schedule for M constructed by the priority algorithm, and let $t_1 < \ldots < t_q$ be the corresponding points of allocation. We define two functions $\eta_j(t,s)$ and $p_j(t,s)$, where for each $i < q$

$$\eta_j(t,s) = \frac{1}{t_{i+1} - t_i} \int_{t_i}^{t_{i+1}} s_j(x)dx, \qquad \text{for any } t_i \leq t < t_{i+1},$$

and

$$p_j(t,s) = p_j - \int_0^t \eta_j(x,s)dx, \qquad \text{for any } t_i \le t \le t_{i+1}.$$

The priority of any task j at any point t is $p_j(t,s)+\mu_j$, where larger $p_j(t,s)+\mu_j$ means higher priority. It is easy to see that for any point of allocation t_i, $p_j(t_i,s)$ is the remaining processing time of task j at time t_i.

At each point of allocation t_i, all tasks, which are available for processing, are split into several subsets. Each subset is comprised of all tasks with the same priority. The subsets are assigned to processors in the decreasing order of the corresponding priorities. If the number of tasks in the current subset is greater than or equal to the number of remaining processors, then these tasks occupy all these processors. Otherwise, the tasks from the current subset are assigned one task per processor. The allocation terminates when either no subsets of tasks or no processors have left for allocation. Then the next point of allocation is selected, all processors are released at this point, and the allocation procedure repeats.

More specifically, let t_i be a point of allocation, Δ be the length of the time interval between this and the next point of allocation, $M' \subseteq M$ be a set of all tasks of the same priority available for processing at t_i, and $\delta_j(t_i,s)$ be the amount of processor time that a task $j \in M'$ receives in the time interval $[t_i, t_i+\Delta]$. Then $\delta_j(t_i,s) = \frac{\Delta}{|M'|}k$, where k is the number of processors allocated to M'. Consequently, if no processors are assigned to the set M' then, $k=0$ and $\delta_j(t_i,s)=0$ for all $j \in M'$. Observe that, in the time interval $[t_i, t_i+\Delta]$, all tasks from M' receive the same amount of processing time, and that $|M'| \ge k$. The next point of allocation is chosen by calculating the largest Δ satisfying the following two conditions:

$$\delta_j(t_i,s) \le p_j(t_i,s),$$

for any task j assigned for processing at the point of allocation t_i; and

$$p_{j'}(t_i,s)+\mu_{j'} - p_{j''}(t_i,s) - \mu_{j''} \ge \delta_{j'}(t_i,s) - \delta_{j''}(t_i,s),$$

for any two tasks j' and j'' available for processing at the point of allocation t_i and such that $p_{j'}(t_i,s)+\mu_{j'} > p_{j''}(t_i,s)+\mu_{j''}$. The first condition ensures that no task receives the amount of processing time that exceeds time required for its completion. The second condition guarantees that if at one point of allocation task j' has a priority higher than task j'', then at the next point of allocation the priority of j' is not less than that of j''.

If a subset M' is assigned to k processors, where $|M'| > k$, then the actual schedule for these tasks is obtained by McNaughton's algorithm (McNaughton, 1959). In accord with this algorithm we select a processor and allocate to this processor from time t_i an arbitrary task $j_1 \in M'$. After that we select an arbitrary task $j_2 \in M'$ and allocate this task to the same processor from time $t_i + \delta_{j_1}(t_i,s)$. We continue to allocate tasks one after another to the selected processor until we reach a task j_r, which cannot be allocated entirely to this processor. Then we allocate task j_r to the selected processor only till the time point $t_i+\Delta$. After that we select another processor and allocate task j_r to this new processor from time t_i in such a way that the total processing time for this task on both processors becomes $\delta_{j_r}(t_i,s)$. We continue to allocate tasks to this second processor until we encounter a situation that next task cannot be allocated entirely to this processor by time $t_i+\Delta$. Then we allocate this task only till the time $t_i+\Delta$, again select a new processor and allocate to this new processor the considered task from time t_i for the remaining processing time, and so on.

In what follows, the priority algorithm described in this section will be denoted by P.

3. Calculation of μ's

For any task j let $K(j)$ be the set of all successors of j, i.e. $K(j)$ is the set of all tasks i such that $j \to i$. The approach presented in (Zinder and Roper, 1998) suggests to calculate each μ_j using a schedule for $K(j)$. Moreover, (Zinder and Roper, 1998) suggests to construct this schedule using μ_i which have been already assigned for all $i \in K(j)$. In this case, the value of μ_j depends on the algorithm A used in the construction of the corresponding schedule. To reflect this fact we will use the notation $\mu_j(A)$. In what follows we will assume that all μ's are calculated using the same algorithm A. More rigorously, the calculation of μ's can be described as follows.

(1) For each task $j \in N$ such that $K(j) = \emptyset$, let $\mu_j(A) = \max_{i \in N} d_i - d_j$.

(2) Select any task $j \in N$ satisfying the following two conditions:
   (a) The value of $\mu_j(A)$ has not been specified.
   (b) The values of $\mu_i(A)$ have been specified for all $i \in K(j)$.

   Ignore for a moment all tasks from $N - K(j)$ and preserve all precedence constraints between tasks from $K(j)$. Construct a schedule s^j for the set $K(j)$ using the algorithm A. Set

$$\mu_j(A) = \max\{\max_{i \in K(j)}[C_i(s^j)+\mu_i(A)], \max_{i \in N} d_i - d_j\}.$$

   Repeat step 2 until values of $\mu_j(A)$ have been specified for all tasks $j \in N$.

We will refer to the above algorithm as μ-algorithm. Let A^* be an algorithm, which constructs for each task $j \in N$ with $K(j) \neq \emptyset$ a schedule for $K(j)$ with the smallest value of the criterion $\max_{i \in K(j)}[C_i(s)+\mu_i(A^*)]$. The following lemma explains why values $\mu_j(A^*)$ play an important role in the following analysis.

Lemma 3.1 *For an arbitrary schedule s for N,*

$$\max_{v \in N}[C_v(s)+\mu_v(A^*)] = L_{max}(s) + \max_{i \in N} d_i.$$

By Lemma 3.1, the algorithm A^* constructs a schedule for $K(j)$, which is optimal for the criterion $\max_{i \in K(j)}[C_i(s) - d_i]$. Since the problem of constructing such a schedule is equivalent to the original problem, we will calculate μ_j using the priority algorithm P described in Section 2.

4. Analysis

In this section we assume that all μ's are calculated using the priority algorithm P and that this algorithm constructs the final schedule which will be denoted by s^{μ}. Suppose that the schedule s^{μ} has been constructed using h points of allocation. Let they be points $t_1, \ldots, t_h$, where $0 = t_1 < \ldots < t_h$. For each point of allocation t_i, let S_i be the set of all tasks, which are allocated for processing at point t_i. Note that $S_h = \emptyset$ and the procedure terminates at this point. For each point of allocation t_i, $i > 1$, let F_i be the set of all tasks, which complete their processing in the time interval $[t_{i-1}, t_i]$ at point t_i. Since, for any $j \in F_i$, $C_j(s^{\mu}) \geq t_i + p_j(t_i, s^{\mu})$,

$$\max_{j \in F_i}[t_i + p_j(t_i, s^{\mu}) + \mu_j(P)] \leq \max_{j \in N}[C_j(s^{\mu}) + \mu_j(P)].$$

Consider a task q satisfying the equality $C_q(s^{\mu}) + \mu_q(P) = \max_{j \in N}[C_j(s^{\mu}) + \mu_j(P)]$. Suppose that $t_{i-1} < C_q(s^{\mu}) < t_i$, for some $i > 1$. This means that at point t_{i-1}, task q belonged to a subset, say N', which was allocated to the number of processors less than $|N'|$. According to the algorithm P, there is a task $r \in N'$ that completes its processing in the time interval $[t_{i-1}, t_i]$ at point t_i. Hence, $C_r(s^{\mu}) \geq t_i + p_r(t_i, s^{\mu})$. Because q and r belong at point t_{i-1} to the same subset, they must have the same priority at point t_i, that is $\mu_q(P) = \mu_r(P) + p_r(t_i, s^{\mu})$. We have,

$$\begin{aligned} C_q(s^{\mu}) + \mu_q(P) &< t_i + \mu_q(P) \\ &= t_i + \mu_r(P) + p_r(t_i, s^{\mu}) \leq C_r(s^{\mu}) + \mu_r(P), \end{aligned}$$

which contradicts the selection of task q. Therefore, $C_q(s^{\mu}) = t_i$, for some $i > 1$, and for this point of allocation

$$\max_{j \in F_i}[t_i + p_j(t_i, s^{\mu}) + \mu_j(P)] = \max_{v \in N}[C_v(s^{\mu}) + \mu_v(P)]. \quad (3)$$

Among all t_i satisfying (3) select the smallest, say t_{i^*}, and select $x \in F_{i^*}$ such that

$$t_{i^*} + p_x(t_{i^*}, s^{\mu}) + \mu_x(P) = \max_{j \in F_{i^*}}[t_{i^*} + p_j(t_{i^*}, s^{\mu}) + \mu_j(P)].$$

Let $M_{i^*} = S_{i^*-1}$, and for each $1 < i < i^*$, we denote by M_i the set of all $j \in S_{i-1}$ such that $\mu_j(P) + p_j(t_i, s^{\mu}) \geq \mu_x(P) + p_x(t_{i^*}, s^{\mu})$. We will call an interval $[t_{i-1}, t_i]$ complete if $|M_i| \geq m$. Otherwise we will call the interval incomplete. Note that sets M_i, and therefore the notion of completeness, are defined only for points of allocation t_i with $1 < i \leq i^*$.

The following lemmas and the theorem outline the steps in the proof of the presented results.

Lemma 4.1 *If all intervals $[t_{i-1}, t_i]$, where $1 < i \leq i^*$, are complete, then for any schedule s*

$$\max_{j \in N}[C_j(s^{\mu}) + \mu_j(P)] \leq \max_{j \in N}[C_j(s) + \mu_j(P)].$$

Lemma 4.2 *For any task $v \in S_{i^*-1}$*

$$\mu_v(P) + p_v(t_{i^*-1}, s^{\mu}) = \mu_x(P) + p_x(t_{i^*-1}, s^{\mu}),$$

and $|M_{i^*}| > m$.

Lemma 4.3 *Let $[t_{i-1}, t_i]$ be an incomplete time interval. Then for any task j, such that $p_j(t_i, s^{\mu}) > 0$ and $\mu_j(P) + p_j(t_i, s^{\mu}) \geq \mu_x(P) + p_x(t_{i^*}, s^{\mu})$, either $j \in M_i$, or there is a task j' such that $j' \in M_i$ and $j' \rightarrow j$.*

Lemma 4.4 *Let $[t_{v-1}, t_v]$ be any incomplete time interval, then $|M_v| \geq 2$.*

Lemma 4.5 *Let s^* be an optimal schedule for the criterion $L_{max}(s)$, then*

$$L_{max}(s^{\mu}) \leq \left(2 - \frac{2}{m}\right) L_{max}(s^*) + \left(1 - \frac{2}{m}\right) \max_{j \in N} d_j.$$

Theorem 4..1 *Schedule s^{μ} is optimal for $P2|prec, prmp|L_{max}$ and $P|in-tree, prmp|L_{max}$.*

References

Braschi, B. and D. Trystram. (1994). A new insight into the Coffman-Graham algorithm. *SIAM J. Comput.*, Vol. 23, pp. 662-669.

Brucker, P., M.R. Garey, and D.S. Johnson, Scheduling equal-length tasks under tree-like precedence constraints to minimise maximum lateness. *Mathematics of Operations Research*, Vol. 2, No. 3, pp. 275-284.

Coffman Jr, E.G. and R.L. Graham. (1972). Optimal scheduling for two-processor systems. *Acta Inform.*, Vol. 1, pp. 200-213.

Garey, M.R. and D.S. Johnson. (1976). Scheduling tasks with nonuniform deadlines on two processors. *Journal of ACM*, Vol. 23, No.6, pp. 461-467.

Lam, S. and R. Sethi. (1977). Worst case analysis of two scheduling algorithms. *SIAM J. Computing,* Vol. 6, pp. 518-536.

Lawler, E.L. (1982). Preemptive scheduling of precedence-constrained jobs on parallel machines, in *Deterministic and stochastic scheduling* (M.A.H. Dempster et al. (eds)), pp. 101-123. D. Reidel Publishing Company.

McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science,* Vol. 6, pp. 1-12.

Muntz, R.R. and E.G. Coffman Jr. (1969). Optimal preemptive scheduling on two-processor systems. *IEEE Trans. Comput.*, Vol. C-18, pp. 1014-1020.

Muntz, R.R. and E.G. Coffman Jr. (1970). Preemptive scheduling of real time tasks on multiprocessor systems. *Journal of Assoc. Comput. Mach.*, Vol. 17, pp. 324-338.

Singh, G. and Y. Zinder. (2000a). Worst-case performance of critical path type algorithms. *International Trans. Oper. Res.*, Vol. 7, pp. 383-399.

Singh, G. and Y. Zinder. (2000b). Worst-case performance of two critical path type algorithms. *Asia Pacific J. Oper. Res.*, Vol. 17, No. 1, pp. 101-121.

Ullman, J.D. (1975). NP-Complete scheduling problems. *J. Comput. System Sci.*, Vol. 10, pp. 384-393.

Zinder, Y. and D. Roper. (1998). An iterative algorithm for scheduling unit-time operations with precedence constraints to minimise the maximum lateness. *Annals of Operations Research*, Vol. 81, pp. 321-340.