PERGAMON

Intl. Trans. in Op. Res. 7 (2000) 383–399

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Worst-case performance of critical path type algorithms

G. Singh, Y. Zinder*

University of Technology, P.O. Box 123, Broadway, NSW 2007, Sydney, Australia

Received 11 February 2000; accepted 24 February 2000

Abstract

The critical path method remains one of the most popular approaches in practical scheduling. Being developed for the makespan problem this method can also be generalized to the maximum lateness problem. For the unit execution time task system and parallel processors this generalization is known as the Brucker–Garey–Johnson algorithm. We characterize this algorithm by introducing an upper bound on the deviation of the criterion from its optimal value. The bound is stated in terms of parameters characterizing the problem, namely number of processors, the length of the longest path, and the total required processing time. We also derive a similar bound for the preemptive version of the Brucker–Garey–Johnson algorithm. © 2000 IFORS. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Deterministic scheduling problems; Parallel identical machines; Precedence constraints; Maximum lateness; Worst-case analysis

1. Introduction

The critical path method remains one of the most popular approaches in practical scheduling. The first theoretical justification of this method has been given by Hu (1961). Hu's algorithm allows the generalization to the case of the maximum lateness problem. This generalization is known as the Brucker–Garey–Johnson algorithm (Brucker et al., 1977). Both the algorithms were developed for the model with a unit execution time (UET) task system, precedence constraints, and parallel processors. Being of considerable practical interest in its own right, this model provides an important insight into the field of deterministic scheduling

* Corresponding author. Tel.: +61-2-9514-2279; fax: +61-2-9514-2248.
E-mail address: y.zinder@maths.uts.edu.au (Y. Zinder).

PII: S0969-6016(00)00023-X

with precedence constraints, and therefore, remains as a subject of intensive research over several decades.

The maximum lateness problem for a UET task system can be stated as follows. Suppose that a set $N = \{1, 2, \ldots, n\}$ of n tasks (jobs, operations) is to be processed on $m > 1$ identical 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 as $i \rightarrow j$, then the processing of i must be completed before the processing of j begins. The processing of the tasks commences at time $t = 0$. Each processor can process at most one task at a time, and each task can be processed by any processor. Once a processor begins executing a task it continues until completion (i.e., no preemptions are allowed). The processing time of task j is denoted by p_j. All processing times are equal, and therefore, without loss of generality, can be taken as the unit of time. Let $C_j(s)$ be the completion time of task j in schedule s. It is necessary to find a schedule, which minimizes the criterion of maximum lateness

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

where d_j is a due date associated with each task j. If all due dates are equal to zero the maximum lateness problem converts into the makespan problem. Using the popular three-field notation $\alpha|\beta|\gamma$, where α specifies the processor environment, β specifies the task characteristics, and γ denotes the optimality criterion, the above problem can be denoted by $P|\text{prec}, p_j = 1|L_{\max}$. Here, P specifies that there are several parallel identical processors, prec indicates the presence of precedence constraints, and the term $p_j = 1$ indicates that each task has a unit processing time.

It is convenient to represent a partially ordered set of tasks by an acyclic directed graph, where nodes correspond to the tasks and arcs reflect the precedence constraints. Hence, the scheduling problems with UET task systems and m parallel identical processors can be viewed as graph partition problems, where the set of nodes must be partitioned into several subsets containing no more than m nodes each. The Brucker–Garey–Johnson algorithm solves the maximum lateness problem if the corresponding graph is an in-tree. The $P|\text{prec}, p_j = 1|L_{\max}$ problem is NP-hard if the corresponding graph is an out-tree (Brucker et al., 1977).

Therefore, for general precedence constraints, the Brucker–Garey–Johnson algorithm is only an approximation algorithm. This algorithm represents the family of so-called priority algorithms. A priority algorithm can be viewed as a two-phase procedure. The first phase of this procedure associates with each task a number, which indicates the priority (urgency) of the task. The second phase allocates tasks for processing in accord to these priorities, i.e. each time, when a processor becomes available, among all tasks, which are ready for processing, the algorithm selects a task with the highest priority.

The idea of a priority algorithm can be modified to the case of arbitrary processing times and preemptions. If preemptions are allowed, the processing of any task can be interrupted at any time and resumed later on the same or another processor. The corresponding maximum lateness problem with preemptions is denoted by $P|\text{prmp}, \text{prec}|L_{\max}$. In general, the preemptive version of the Brucker–Garey–Johnson algorithm provides only an approximation solution,

since the makespan problem with an arbitrary number of processors, general precedence constraints, arbitrary processing times, and preemptions is also NP-hard (Ullman, 1975).

The performance guarantees are viewed as an essential component of the characterization of approximation algorithms. For the scheduling problems with parallel identical processors many results of this type were obtained for the makespan problem (see, for example, Parker, 1995; Lawler et al., 1993). Interest in such performance guarantees is inspired not only by the fact that they provide an important insight into the nature of the corresponding approximation algorithms, but also by the role which these results play in the development of various algorithms requiring calculation of lower bounds, in particular, in the development of branch-and-bound algorithms. In spite of the existence of performance guarantees for a number of algorithms for the makespan problem, much less is known about the worst-case performance of the approximation algorithms for the maximum lateness problem. Thus, it was shown by Zinder and Roper (1998), that

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

where s^Z is a schedule constructed by the approximation algorithm, s^* is an optimal schedule for the maximum lateness problem, and

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

The establishment of similar results for other known approximation algorithms can be viewed as a logical next step in the theoretical study in the field. The result in Zinder and Roper (1998), as well as many performance guarantees for the makespan problem presents an upper bound on the value of the criterion corresponding to the approximation algorithm expressed in terms of its optimum value. In contrast, in this paper we present two performance guarantees in the form of bounds on the deviation of the criterion value from its optimum expressed in terms of the parameters characterizing the partially ordered set of tasks. This form provides more information on the worst-case performance of the respective algorithms and is convenient for the use in various flexible self-tuned algorithms that select an appropriate strategy by analyzing the problem on hand. A similar form of performance guarantee for the classical Hu's algorithm is presented in Singh and Zinder (2000). In Section 2 we consider the $P|\text{prec}, p_j = 1|L_{\max}$ problem and present a tight upper bound on the differences $L_{\max}(s^{\text{BGJ}}) - L_{\max}(s^*)$, where s^{BGJ} is the schedule constructed by the Brucker–Garey–Johnson algorithm and s^* is an optimal schedule for $P|\text{prec}, p_j = 1|L_{\max}$ problem. In Section 3 we consider the $P|\text{prmp}, \text{prec}|L_{\max}$ problem and present a tight bound for the preemptive counterpart of the Brucker–Garey–Johnson algorithm.

2. The worst-case performance of the Brucker–Garey–Johnson algorithm for the $P|\text{prec}, p_j = 1|L_{\max}$ problem

The priority associated with each task by the Brucker–Garey–Johnson algorithm is called a

modified due date, and the tasks are arranged in a list in the non-decreasing order of their modified due dates. The time points $t = 0$, $t = 1$, $t = 2$, and so on, are considered sequentially. For each time point the list is scanned from left to right and the first encountered task available for processing is assigned to an idle processor and eliminated from the list. The algorithm continues to scan the list and to allocate tasks for processing until at least one of the following events occurs: the end of the list has been reached or all processors have tasks assigned to them. Then the allocation of the remaining tasks is made for subsequent time points until all tasks have been assigned.

We will denote the modified due date associated with task j by d'_j. Modified due date d'_j depends on the modified due dates of the immediate successors of task j. Task i is an immediate successor of task j if $j \rightarrow i$ and there is no task k such that $j \rightarrow k$ and $k \rightarrow i$. We will denote the set of all immediate successors of task j by $K(j)$. In Brucker et al. (1977), the algorithm computing the modified due dates was given for in-trees. The algorithm below is a straightforward extension of this algorithm to the case of an arbitrary partially ordered set.

1. For each task j that does not have successors, set $d'_j = d_j$.
2. Select a task j which has not been assigned its modified due date d'_j and whose all immediate successors have been assigned their modified due dates. If no such task exists, then stop.
3. Set $d'_j = \min_{i \in K(j)}\{d_j, d'_i - p_i\}$, where p_i is the processing time of task i, and return to step 2.

We will refer to this algorithm as the due date modification algorithm. Note that this algorithm without any changes will be also used in the preemptive case, where the processing time p_i can be any positive number.

In order to analyze the worst-case performance of the Brucker–Garey–Johnson algorithm, we observe that the replacement of the original due dates by the corresponding modified due dates does not affect the maximum lateness. Indeed, since $d'_q \leq d_q$ for any task q, we have

$$\max_{q \in N}\left(C_q(s) - d'_q\right) \geq \max_{q \in N}\left(C_q(s) - d_q\right) = L_{\max}(s).$$

To show that

$$\max_{q \in N}\left(C_q(s) - d'_q\right) \leq L_{\max}(s), \tag{2}$$

among all tasks j satisfying

$$\max_{q \in N}\left(C_q(s) - d'_q\right) = C_j(s) - d'_j$$

select a task with the largest completion time. Let it be task b. If $d_b = d'_b$, then Eq. (2) holds. On the other hand, if $d_b > d'_b$, then according to the due date modification algorithm there is an immediate successor of task b, say task v, such that $d'_b = d'_v - p_v$. Since $C_v(s) \geq C_b(s) + p_v$, we have

$$C_b(s) - d'_b = C_b(s) - (d'_v - p_v) = (C_b(s) + p_v) - d'_v \leq C_v(s) - d'_v,$$

which contradicts the selection of task b.

Let s^{BGJ} be a schedule constructed by the Brucker–Garey–Johnson algorithm. From the set of all tasks j such that

$$C_j(s^{\mathrm{BGJ}}) - d'_j = L_{\max}(s^{\mathrm{BGJ}})$$

choose a task p with the smallest completion time. If $C_p(s^{\mathrm{BGJ}}) = 1$, then s^{BGJ} is optimal. If $C_p(s^{\mathrm{BGJ}}) > 1$, then we construct a sequence of tasks $a_h, \ldots, a_1$ and a sequence of sets of tasks $M_h, \ldots, M_1$ using the following procedure. Select task p as a_1, and denote by $\tilde{M}_1$ the set comprised of task p and all tasks j such that $d'_j \leq d'_p$ and $C_j(s^{\mathrm{BGJ}}) < C_p(s^{\mathrm{BGJ}})$. If for any positive integer $t < C_{a_1}(s^{\mathrm{BGJ}})$ at least two tasks from $\tilde{M}_1$ are processed in s^{BGJ} on the time interval $[t-1, t]$, then $M_1 = \tilde{M}_1$ and the procedure terminates. Otherwise, we select the largest number among all positive integers $t < C_{a_1}(s^{\mathrm{BGJ}})$ such that only one task from $\tilde{M}_1$ is processed on the time interval $[t-1, t]$. Denote this task by a_2 and set M_1 equals the set of all tasks $j \in \tilde{M}_1$ with $C_j(s^{\mathrm{BGJ}}) > C_{a_2}(s^{\mathrm{BGJ}})$. Let $\tilde{M}_2$ be the set of all tasks j such that $d'_j \leq d'_{a_2}$ and $C_j(s^{\mathrm{BGJ}}) \leq C_{a_2}(s^{\mathrm{BGJ}})$. Suppose that $C_{a_2}(s^{\mathrm{BGJ}}) = 1$, or $C_{a_2}(s^{\mathrm{BGJ}}) > 1$ and for any positive integer $t < C_{a_2}(s^{\mathrm{BGJ}})$ at least two tasks from $\tilde{M}_2$ are processed in s^{BGJ} on the time interval $[t-1, t]$. Then $M_2 = \tilde{M}_2$ and the procedure terminates. Otherwise, we select the largest number among all positive integers $t < C_{a_2}(s^{\mathrm{BGJ}})$ such that only one task from $\tilde{M}_2$ is processed on the time interval $[t-1, t]$. Denote this task by a_3 and set M_2 equals the set of all tasks $j \in \tilde{M}_2$ with $C_{a_2}(s^{\mathrm{BGJ}}) \geq C_j(s^{\mathrm{BGJ}}) > C_{a_3}(s^{\mathrm{BGJ}})$. We continue in the same manner until for some h the procedure terminates because either $C_{a_h}(s^{\mathrm{BGJ}}) = 1$, or $C_{a_h}(s^{\mathrm{BGJ}}) > 1$ and for any positive integer $t < C_{a_h}(s^{\mathrm{BGJ}})$ at least two tasks from $\tilde{M}_h$ are processed in s^{BGJ} on the time interval $[t-1, t]$.

Lemma 2.1. *Let j be an arbitrary task from M_i, for some $1 \leq i \leq h-1$. Then $a_{i+1} \rightarrow j$.*

Proof. Note that any task q, satisfying the conditions $d'_q \leq d'_{a_i}$ and $C_q(s^{\mathrm{BGJ}}) < C_{a_i}(s^{\mathrm{BGJ}})$, belongs to $\tilde{M}_i$. Therefore, on the time interval $[C_{a_{i+1}}(s^{\mathrm{BGJ}}) - 1,\ C_{a_{i+1}}(s^{\mathrm{BGJ}})]$, according to the schedule s^{BGJ}, only one processor processes a task with the modified due date less than or equal to d'_{a_i}. Let j be an arbitrary task from M_i. Since $d'_j \leq d'_{a_i}$, according to the Brucker–Garey–Johnson algorithm, task j must have a predecessor, say task j', with $C_{j'}(s^{\mathrm{BGJ}}) = C_{a_{i+1}}(s^{\mathrm{BGJ}})$. Since $j' \rightarrow j$, we have $d'_{j'} < d'_j \leq d'_{a_i}$, but on the time interval $[C_{a_{i+1}}(s^{\mathrm{BGJ}}) - 1,\ C_{a_{i+1}}(s^{\mathrm{BGJ}})]$ only task a_{i+1} has the modified due date less than or equal to d'_{a_i}. Therefore, $a_{i+1} \rightarrow j$.

Let $M = \bigcup_{i=1}^{h} M_i$. For any positive integer $t < C_p(s^{\mathrm{BGJ}})$, we say that the time slot t is complete, if exactly m tasks from M are processed on the time interval $[t-1, t]$. Otherwise, the time slot t is said to be incomplete. Let w be the number of all incomplete time slots. If $w > 0$, we will denote incomplete time slots by $t_1, \ldots, t_w$, where $t_1 < \ldots < t_w < C_p(s^{\mathrm{BGJ}})$.

Lemma 2.2. *The time slot $C_p(s^{\mathrm{BGJ}}) - 1$ is complete and any task that is processed in this time slot does not precede task p.*

Proof. Suppose that there exists a task j such that $C_j(s^{\mathrm{BGJ}}) = C_p(s^{\mathrm{BGJ}}) - 1$ and $j \rightarrow p$. Then

$d'_j \le d'_p - 1$, and

$$C_j(s^{\mathrm{BGJ}}) - d'_j \ge C_p(s^{\mathrm{BGJ}}) - 1 - \left(d'_p - 1\right) = C_p(s^{\mathrm{BGJ}}) - d'_p,$$

which contradicts the selection of task p.

Suppose that the time slot $C_p(s^{\mathrm{BGJ}}) - 1$ is incomplete. Then in this time slot at least one processor is either idle or processes a task with the modified due date greater than d'_p. Therefore, according to the Brucker–Garey–Johnson algorithm, task p must have a predecessor that is processed in s^{BGJ} in the time slot $C_p(s^{\mathrm{BGJ}}) - 1$. But as has been shown above, this contradicts the selection of task p.

Lemma 2.3. *Let U be the set comprised of task p and all tasks processed in the time slot $C_p(s^{\mathrm{BGJ}}) - 1$. Then for any task $b \in U$, there exists a sequence of tasks $b_1, \ldots, b_w$ from M such that $b_1 \to \cdots \to b_w \to b$, and $C_{b_i}(s^{\mathrm{BGJ}}) = t_i$, for all $1 \le i \le w$.*

Proof. Let us denote task b by b_{w+1}. Suppose that for some $k \le w+1$ a subsequence $b_k, \ldots, b_{w+1}$ has been constructed, and $b_k \in M_i$. If $t_{k-1} = C_{a_{i+1}}(s^{\mathrm{BGJ}})$, then by Lemma 2.1 $a_{i+1} \to b_k$, and we choose task a_{i+1} as b_{k-1}. Suppose that $t_{k-1} > C_{a_{i+1}}(s^{\mathrm{BGJ}})$. Observe that any task j such that $d'_j \le d'_{a_i}$ and $C_j(s^{\mathrm{BGJ}}) < C_{a_i}(s^{\mathrm{BGJ}})$ belongs to $\tilde{M}_i$. Hence, in the schedule s^{BGJ} in the time slot t_{k-1} at least one processor is either idle or processes a task with the due date greater than d'_{b_k}. Therefore, task b_k must have a predecessor, which is processed in the time slot t_{k-1}. We denote this predecessor by b_{k-1}. Since $b_{k-1} \to b_k$, according to the due date modification algorithm $d'_{b_{k-1}} \le d'_{b_k} - 1$. Hence, $b_{k-1} \in M_i$. Continuing in the same manner, we obtain the desired chain.

Consider a longest path $\wp$ in the subgraph corresponding to the set M. Let $l(t)$ be the number of tasks in this path with the completion time greater than t. Then the length of $\wp$ equals $l(0)$.

Lemma 2.4. *If each incomplete time slot contains exactly one task from the set M, then*

$$L_{\max}(s^*) \ge l(0) + 1 - d'_p, \tag{3}$$

where s^* is an optimal schedule for the maximum lateness problem.

Proof. Lemma 2.2 implies that there exists a positive integer $t \le C_p(s^{\mathrm{BGJ}})$ such that no task from $\wp$ is processed on the time interval $[t-1, t]$. Let t' be the smallest of these numbers, and let r be the largest value of i satisfying inequality $t' \le C_{a_i}(s^{\mathrm{BGJ}})$. Since the time slot t' does not contain a task from $\wp$,

$$l\left(C_{a_r}(s^{\mathrm{BGJ}})\right) + C_{a_r}(s^{\mathrm{BGJ}}) \ge l(0) + 1. \tag{4}$$

By the due date modification algorithm, in the path $\wp$ the $l(C_{a_r}(s^{\mathrm{BGJ}}))$th from the right task has a due date less than or equal to $d'_p - l(C_{a_r}(s^{\mathrm{BGJ}})) + 1$. Then Lemma 2.1 implies that $d'_{a_r} \le d'_p - l(C_{a_r}(s^{\mathrm{BGJ}}))$. Let j be an arbitrary task from M_r. Since $d'_j \le d'_{a_r}$, we have

$$d'_j \le d'_p - l\left(C_{a_r}(s^{\mathrm{BGJ}})\right). \tag{5}$$

Suppose that $r < h$. Then a_{r+1} belongs to the path $\wp$, and each time slot t, where $t \le C_{a_{r+1}}(s^{\mathrm{BGJ}})$, contains a task from $\wp$. Hence, task a_{r+1} cannot be completed earlier than in the schedule s^{BGJ}, and therefore, $C_{a_{r+1}}(s^*) \ge C_{a_{r+1}}(s^{\mathrm{BGJ}})$. Because all time slots t, where $C_{a_{r+1}}(s^{\mathrm{BGJ}}) < t' < C_{a_r}(s^{\mathrm{BGJ}})$, are complete, and because by Lemma 2.1 a_{r+1} precedes all tasks from M_r,

$$\max_{j\in M_r} C_j(s^*) \ge C_{a_r}(s^{\mathrm{BGJ}}). \tag{6}$$

Since all time slots t, where $t < C_{a_h}(s^{\mathrm{BGJ}})$, are complete, the inequality Eq. (6) also holds if $r = h$. Using Eqs. (4)–(6), we have

$$L_{\max}(s^*) \ge \max_{j\in M_r}\left\{C_j(s^*) - d'_j\right\} \ge \max_{j\in M_r}\left\{C_j(s^*) - d'_p + l\left(C_{a_r}(s^{\mathrm{BGJ}})\right)\right\}$$

$$\ge C_{a_r}(s^{\mathrm{BGJ}}) - d'_p + l\left(C_{a_r}(s^{\mathrm{BGJ}})\right) \ge l(0) + 1 - d'_p.$$

This completes the proof.

Lemma 2.5. *If $w = 0$, then s^{BGJ} is optimal.*

Proof. If there are no incomplete time slots, then the tasks from M cannot be processed more quickly by any other schedule. Therefore, for any schedule s^* that is optimal for the maximum lateness problem,

$$C_p(s^{\mathrm{BGJ}}) \le \max_{j\in M} C_j(s^*),$$

and since $d'_j \le d'_p$, for all $j \in M$,

$$L_{\max}(s^{\mathrm{BGJ}}) = C_p(s^{\mathrm{BGJ}}) - d'_p \le \max_{j\in M} C_j(s^*) - d'_p = \max_{j\in M}\left\{C_j(s^*) - d'_p\right\}$$

$$\le \max_{j\in M}\left\{C_j(s^*) - d'_j\right\} \le \max_{j\in N}\left\{C_j(s^*) - d'_j\right\} = L_{\max}(s^*).$$

Hence, s^{BGJ} is optimal.

Theorem 2.1. *If s^* is an optimal schedule for the maximum lateness problem, then*

$$L_{\max}(s^{\mathrm{BGJ}}) - L_{\max}(s^*) \le \begin{cases} \min\left\{\frac{n-l}{m} - 1, \frac{m-1}{m}l\right\}, & \text{if } n - l \ge m \\ 0, & \text{otherwise} \end{cases}, \tag{7}$$

where n is the number of tasks and l is the length of the longest path in the corresponding graph. For any positive integer $\hat{n}$ there is an instance of the maximum lateness problem with $n \ge \hat{n}$ such

that Eq. (7) *is an equality.*

Proof. It is easy to see that schedule s^{BGJ} is optimal, and therefore Eq. (7) holds, if either $n-l<m$, or $n-l\geq m$ and $C_p(s^{\mathrm{BGJ}})=1$. By Lemma 2.5, s^{BGJ} is also optimal when $n-l\geq m$, $C_p(s^{\mathrm{BGJ}})>1$, and $w=0$. Hence, we only need to prove that Eq. (7) holds when $n-l\geq m$, $C_p(s^{\mathrm{BGJ}})>1$, and $w>0$. In this case we can construct the set M, and let c be the number of complete time slots t satisfying the inequality $t<C_p(s^{\mathrm{BGJ}})$. Hence, $C_p(s^{\mathrm{BGJ}})=c+w+1$. Since $l(0)$ is the length of a longest path in the subgraph corresponding to the set M, we have $L_{\max}(s^*)\geq l(0)-d'_p$. Suppose that $L_{\max}(s^*)\geq l(0)+1-d'_p$. By Lemma 2.3 $l(0)\geq w+1$, and

$$\begin{aligned}L_{\max}(s^{\mathrm{BGJ}}) &= C_p(s^{\mathrm{BGJ}})-d'_p\\ &= \frac{(mc+w+1)+(w+1)(m-1)}{m}-d'_p\leq\frac{|M|-(w+1)}{m}+w+1\\ &\quad -d'_p\leq\frac{|M|-l(0)}{m}+l(0)-d'_p.\end{aligned}$$

Hence,

$$L_{\max}(s^{\mathrm{BGJ}})-L_{\max}(s^*)\leq\frac{|M|-l(0)}{m}-1. \tag{8}$$

Now suppose that $L_{\max}(s^*)=l(0)-d'_p$. Then from Lemma 2.4 we conclude that there is an incomplete time slot containing at least two tasks from M, and therefore, $|M|\geq cm+w+2$. Consider the set U specified in the statement of the Lemma 2.3. Since $|U|=m+1$, this lemma implies that $\max_{j\in M}C_j(s^*)\geq w+2$. On the other hand, since $L_{\max}(s^*)=l(0)-d'_p$, we have $\max_{j\in M}C_j(s^*)=l(0)$, and hence, $l(0)\geq w+2$. We have

$$\begin{aligned}L_{\max}(s^{\mathrm{BGJ}}) &= C_p(s^{\mathrm{BGJ}})-d'_p\\ &= \frac{(mc+w+2)-(w+2)}{m}+(w+2)-1-d'_p\leq\frac{|M|-l(0)}{m}+l(0)-1-d'_p,\end{aligned}$$

and, subtracting $L_{\max}(s^*)=l(0)-d'_p$, we again obtain Eq. (8). Since $n-|M|\geq l-l(0)$, Eq. (8) gives

$$L_{\max}(s^{\mathrm{BGJ}})-L_{\max}(s^*)\leq\frac{n-l}{m}-1. \tag{9}$$

Using Lemma 2.3, we conclude that $l\geq l(0)\geq w+1$. Hence,

$$L_{\max}(s^{\mathrm{BGJ}})=C_p(s^{\mathrm{BGJ}})-d'_p=\frac{mc+w+1+(w+1)(m-1)}{m}-d'_p\leq\frac{|M|}{m}+\frac{m-1}{m}l-d'_p,$$

and subtracting the obvious inequality $L_{\max}(s^*)\geq\frac{|M|}{m}-d'_p$, we obtain

$$L_{\max}(s^{\mathrm{BGJ}}) - L_{\max}(s^*) \leq \frac{m-1}{m} l,$$

which together with Eq. (9) gives Eq. (7).

In order to show that Eq. (7) is tight, consider a partially ordered set of tasks presented by the graph in Fig. 1. This graph is comprised of k identical sections and one terminal section.

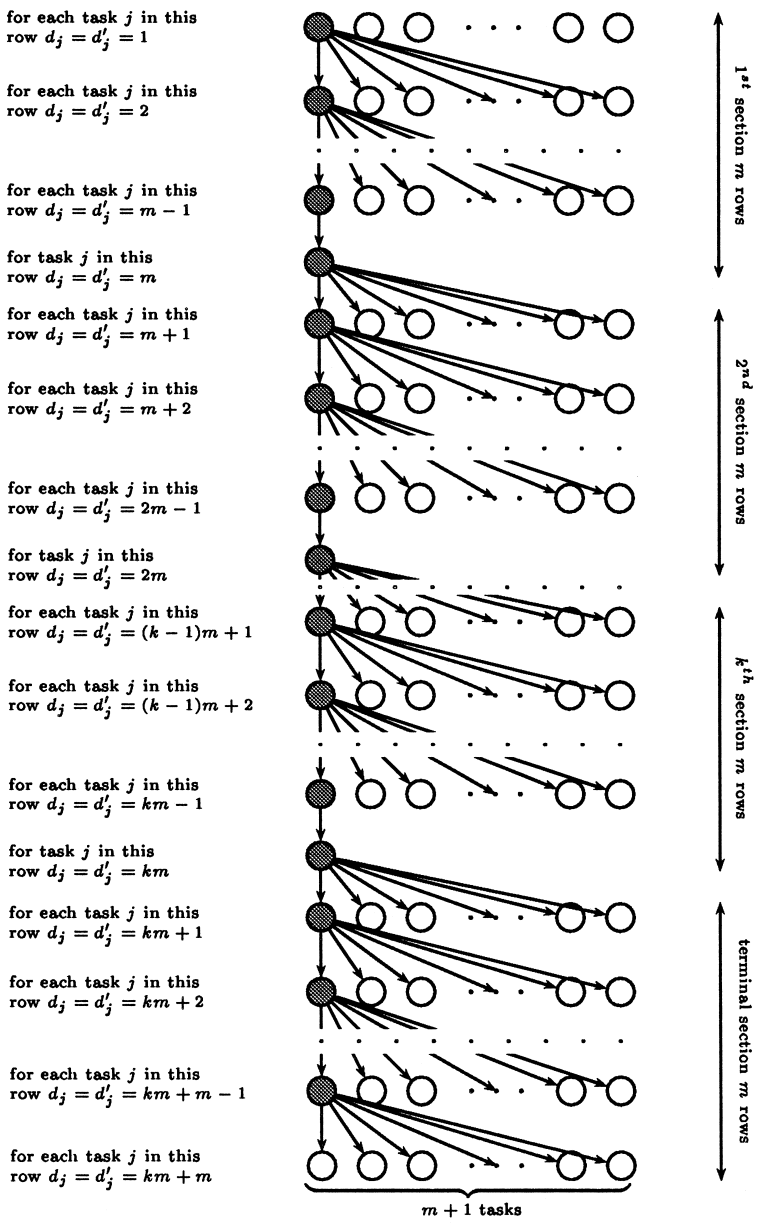

Fig. 1. The partially ordered set of tasks considered in Theorem 2.1.

Each task corresponding to a node in the first (top) row has a due date of one unit of time. Each task from any other row has a due date of the previous row plus one. The corresponding optimal schedule and a schedule constructed by the Brucker–Garey–Johnson algorithm are presented in Figs. 2 and 3, respectively. It is easy to see that $\max_{j\in N} d_j = m(k+1)$, $L_{\max}(s^{\mathrm{BGJ}}) = mk - k + m$, $L_{\max}(s^*)=1$ and Eq. (7) is an equality for any k and m.

3. The worst-case performance of the preemptive version of the Brucker–Garey–Johnson algorithm for the $P|\mathrm{prmp}, \mathrm{prec}|L_{\max}$ problem

The statement of the maximum lateness problem with preemptions is the same as for the $P|\mathrm{prec}, p_j = 1|L_{\max}$ problem, but now we assume that each task j can have an arbitrary processing time p_j and its processing can be interrupted at any time and resumed later on the same or another processor. Let $p_j(t)$ be the remaining processing time for task j at time t. Since preemptions are allowed and the same task j can be allocated for processing at different points of time, its priority depends on $p_j(t)$. As a straightforward generalization of the Brucker–Garey–Johnson algorithm, one may compute the priority of task j at time t as $d'_j - p_j(t)$, where d'_j is the modified due date calculated by the due date modification algorithm presented in Section 2. In this case, a smaller value of $d'_j - p_j(t)$ gives a higher priority.

The construction of the schedule is based on a sequence of decisions regarding what tasks should be allocated for processing, what amount of processing time should receive each of these tasks, and how these tasks should be scheduled. Points in time that correspond to these decisions will be referred to as points of allocation. The first point of allocation is $t = 0$. At each point of allocation tasks are assigned for processing as follows. All tasks, which are ready for processing, are split into several subsets. Each subset is comprised of all tasks with the same priority. If the number of tasks of the highest priority is greater than or equal to the number of processors, then the corresponding subset will occupy all processors and each task will receive the same amount of processing time on the interval between the current and the next points of allocation. If the number of tasks of the highest priority is less than the number of processors, then these tasks are allocated one task per processor, and the similar allocation procedure is conducted for the remaining subsets of tasks and the remaining processors. The

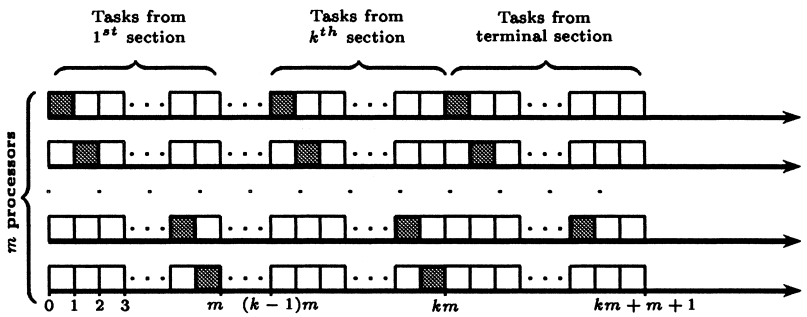

Fig. 2. An optimal schedule.

allocation terminates if there are either no remaining subsets, or no remaining processors. All processors are released at the next point of allocation, and the allocation procedure repeats. Suppose that Δ is the length of the time interval between two successive points of allocation. If a subset comprising u tasks is allocated to k processors, where $u > k$, then each task in this subset will receive $\frac{\Delta}{u}k$ units of processing time. If $u = k$, then each task from this subset will receive Δ units of processing time. The next point of allocation is selected in such a way that either some task is completed or the priority of tasks in one subset becomes equal to the priority of tasks in another. In other words, suppose that tasks from a subset N_1 are allocated one task per processor at a point of allocation t, then Δ — the length of the time interval between the current and the next points of allocation — must satisfy the inequality

$$\Delta \leq \min_{j \in N_1} p_j(t).$$

Suppose that tasks from another subset N_2 are allocated to remaining k processors, where $k < |N_2|$. Then

$$\Delta \leq \min_{j \in N_2} \frac{p_j(t)|N_2|}{k}$$

and for any $j_1 \in N_1$ and $j_2 \in N_2$

$$d'_{j_1} - \left(p_{j_1}(t) - \Delta\right) \leq d'_{j_2} - \left(p_{j_2}(t) - \frac{\Delta k}{|N_2|}\right).$$

Suppose that a task j is ready for processing at point t, but is not assigned to be processed on the interval between this and the subsequent point of allocation. Then for any $j_1 \in N_1$ and $j_2 \in N_2$,

$$d'_j - p_j(t) \geq d'_{j_1} - \left(p_{j_1}(t) - \Delta\right) \quad \text{and} \quad d'_j - p_j(t) \geq d'_{j_2} - \left(p_{j_2}(t) - \frac{\Delta k}{|N_2|}\right).$$

The next point of allocation is defined by the maximum value of Δ, which satisfies these conditions.

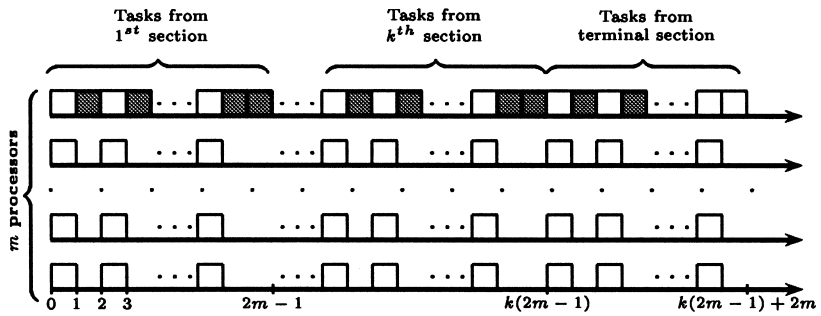

Fig. 3. A schedule constructed by the Brucker–Garey–Johnson algorithm.

The actual scheduling on the time interval between two successive points of allocation is obtained by applying McNaughton's algorithm (McNaughton, 1959). To describe this algorithm we assume that tasks from a subset N' are allocated at point t to k processors, where $k < |N'|$. As before, we denote by Δ the length of the time interval between the current point of allocation and the next one. Therefore, each task from N' will be processed on this interval for $\frac{\Delta k}{|N'|}$ time units. We select a processor and allocate to this processor from time t an arbitrary task $j_1 \in N'$. After that we select an arbitrary task $j_2 \in N'$ and allocate this task to the same processor from time $t + \frac{\Delta k}{|N'|}$. 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 $t + \Delta$. After that we select another processor and allocate this task to this new processor from time t in such a way that the total processing time for this task on both processors becomes $\frac{\Delta k}{|N'|}$. We continue to allocate tasks to this second processor until we encounter a situation that the next task cannot be allocated entirely to this processor. In this case we allocate this task only till the time $t + \Delta$, again select a new processor and allocate to this new processor the considered task from time t for the remaining processing time, and so on. To illustrate this algorithm consider the following example. Let $N' = \{j_1, \ldots, j_6\}$, $\Delta = 3$, and $k = 4$. Then each task from N' will be processed on the time interval $[t, t + \Delta]$ for 2 time units. Fig. 4 depicts the resulting schedule.

The algorithm described above produces an optimal schedule if the corresponding graph is an in-tree (Lawler, 1982). In what follows we will analyze the worst case performance of this algorithm for the general precedence constraints. We will denote the schedule constructed by this algorithm by s^L. In order to be able to indicate what schedule is considered, we also replace the notation $p_j(t)$ by $p_j(t, s)$, where $p_j(t, s)$ is the remaining processing time for task j at time t in schedule s.

As has been shown in Section 2, if preemptions are not allowed, then the replacement of the original due dates by the corresponding modified due dates does not effect the maximum lateness. It is easy to see that the same reasonings are valid for the preemptive case and in this case also, for any schedule s,

$$L_{\max}(s) = \max_{j \in N} \left\{ C_j(s) - d'_j \right\}.$$

Suppose that schedule s^L 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 $S_h = \varnothing$ and the procedure terminates

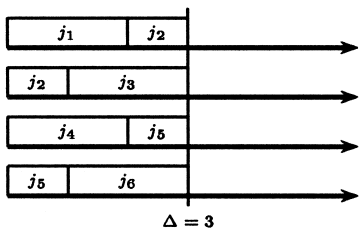

Fig. 4. A preemptive schedule.

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 on the time interval $[t_{i-1}, t_i]$ at point t_i. Since, for any $j \in F_i$, $C_j(s^L) \geq t_i + p_j(t_i, s^L)$,

$$\max_{j \in F_i} \left(t_i + p_j(t_i, s^L) - d'_j \right) \leq L_{\max}(s^L).$$

Consider task q satisfying the equality

$$C_q(s^L) - d'_q = L_{\max}(s^L).$$

Suppose that $t_{i-1} < C_q(s^L) < t_i$, 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, there is a task $r \in N'$ that completes its processing on the time interval $[t_{i-1}, t_i]$ at point t_i. Hence, $C_r(s^L) \geq t_i + p_r(t_i, s^L)$. Because q and r belongs at point t_{i-1} to the same subset, they must have the same priority at point t_i, that is $d'_q = d'_r - p_r(t_i, s^L)$. We have,

$$C_q(s^L) - d'_q < t_i - d'_q = t_i - d'_r + p_r(t_i, s^L) \leq C_r(s^L) - d'_r,$$

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

$$\max_{j \in F_i} \left(t_i + p_j(t_i, s^L) - d'_j \right) = L_{\max}(s^L). \tag{10}$$

Among all t_i satisfying Eq. (10) select the smallest, say t_{i^*}, and select $x \in F_{i^*}$ such that

$$t_{i^*} + p_x(t_{i^*}, s^L) - d'_x = \max_{j \in F_{i^*}} \left[t_{i^*} + p_j(t_{i^*}, s^L) - d'_j \right].$$

For each $1 < i \leq i^*$, we denote by M_i the subset of all $j \in S_{i-1}$ such that

$$d'_j - p_j(t_i, s^L) \leq d'_x - p_x(t_{i^*}, s^L).$$

We will say that an interval $[t_{i-1}, t_i]$, where $1 < i \leq i^*$, is complete if $|M_i| \geq m$. Otherwise the interval is said to be incomplete.

Lemma 3.1. *For any task* $j \in S_{i^*-1}$

$$d'_j - p_j(t_{i^*-1}, s^L) = d'_x - p_x(t_{i^*-1}, s^L),$$

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

Proof. Suppose that there is a task $j \in S_{i^*-1}$ such that

$$d'_j - p_j(t_{i^*-1}, s^L) < d'_x - p_x(t_{i^*-1}, s^L).$$

Then $j \in F_{i^*}$ and

$$p_j(t_{i^*-1}, s^L) = t_{i^*} - t_{i^*-1} + p_j(t_{i^*}, s^L). \tag{11}$$

If $|M_{i^*}| \le m$ or there is a task $q \in S_{i^*-1}$ such that

$$d'_q - p_q(t_{i^*-1}, s^L) > d'_x - p_x(t_{i^*-1}, s^L),$$

then Eq. (11) holds for j equals x. Since in both cases

$$d'_j - p_j(t_{i^*}, s^L) = d'_x - p_x(t_{i^*}, s^L),$$

we have

$$t_{i^*-1} + p_j(t_{i^*-1}, s^L) - d'_j = t_{i^*} + p_j(t_{i^*}, s^L) - d'_j = t_{i^*} + p_x(t_{i^*}, s^L) - d'_x = L_{\max}(s^L),$$

which contradicts the selection of t_{i^*}. Therefore, $|M_{i^*}| > m$ and $d'_j - p_j(t_{i^*} - 1, s^L) = d'_x - p_x(t_{i^*} - 1, s^L)$, for all $j \in S_{i^*-1}$.

We define the length of any path in the directed acyclic graph which represents the partially ordered set of tasks as a sum of processing times corresponding to the nodes in this path. Let l be the length of the longest path, w be the total length of all incomplete time intervals for the schedule s^L, and $p_{\min} = \min_{j \in N} p_j$.

Lemma 3.2. *Let $[t_{i-1}, t_i]$ be an incomplete interval and j be an arbitrary task from $M_{i'}$, where $i < i'$. Then either j is processed on $[t_{i-1}, t_i]$ during $t_i - t_{i-1}$ time units, or there is a task j', such that $j' \in M_i$, $j' \rightarrow j$, and j' is processed on this interval during $t_i - t_{i-1}$ time units.*

Proof. Since $|M_i| < m$, on the interval $[t_{i-1}, t_i]$ at least one processor is either idle or processes a task with priority lower than the priority of j. Hence, either j is processed on $[t_{i-1}, t_i]$, or there is a task j', which precedes j and is ready for processing at t_{i-1}. In the former case, since $|M_i| < m$ and all other tasks from S_{i-1} have a lower priority than tasks from M_i, task j is processed on $[t_{i-1}, t_i]$ during $t_i - t_{i-1}$ time units. In the latter case, according to the due date modification algorithm

$$d'_{j'} \le d'_j - p_j \le d'_x - p_x(t_{i^*}, s^L).$$

Hence, $j' \in M_i$ and this task is also processed on the interval $[t_{i-1}, t_i]$ during $t_i - t_{i-1}$ time units.

Theorem 3.1. *Let s^* be an optimal schedule for the maximum lateness problem, then*

$$L_{\max}(s^L) - L_{\max}(s^*) \le \frac{m-1}{m}(l - p_{\min}). \tag{12}$$

For any positive integer $\hat{n}$ there is an instance of the maximum lateness problem with $n \ge \hat{n}$ such that Eq. (12) *is an equality.*

Proof. Suppose that $w = 0$. Because on the interval $[t_1, t_{i^*}]$ all processors are busy, there exists a task j such that j is processed on this interval in schedule s^L and $C_j(s^*) \ge t_{i^*} + p_j(t_{i^*}, s^L)$. Since d'_j

$-p_j(t_{i^*}, s^L) \le d'_x - p_x(t_{i^*}, s^L)$,

$$L_{\max}(s^*) \ge C_j(s^*) - d'_j \ge t_{i^*} + p_j\left(t_{i^*}, s^L\right) - d'_j \ge t_{i^*} + p_x\left(t_{i^*}, s^L\right) - d'_x = L_{\max}(s^L).$$

Hence, s^L is optimal, and Eq. (12) holds.

Suppose that $w > 0$. For any $j \in \cup_{i=2}^{i^*} M_i$ we denote by i_j the largest index i such that $j \in M_i$. Then for any $j \in \cup_{i=2}^{i^*} M_i$ and any $t \le t_{i_j}$

$$d'_j - p_j\left(t, s^L\right) \le d'_x - p_x\left(t_{i^*}, s^L\right).$$

On the other hand, because $M_{i^*} \neq \varnothing$, by Lemma 3.2, for each incomplete time interval $[t_{i-1}, t_i]$, there is a task $j \in M_i$, which is processed on this interval during $t_i - t_{i-1}$ time units. Hence, denoting the total length of all complete time intervals by c, we obtain

$$\sum_{j \in \cup_{i=2}^{i^*} M_i} p_j \ge \sum_{j \in \cup_{i=2}^{i^*} M_i} \left[p_j - p_j\left(t_{i_j}, s^L\right)\right] \ge mc + w \ge \sum_{j \in \cup_{i=2}^{i^*} M_i} \left[p_j - p_j\left(\frac{mc+w}{m}, s^*\right)\right]. \tag{13}$$

Therefore, there exists a task $q \in \cup_{i=2}^{i^*} M_i$ such that

$$C_q(s^*) \ge \frac{mc+w}{m} + p_q\left(\frac{mc+w}{m}, s^*\right).$$

Without loss of generality we can assume that $p_q(\frac{mc+w}{m}, s^*) \ge p_q(t_{i_q}, s^L)$, because otherwise Eq. (13) implies that there exists a task $r \in \cup_{i=2}^{i^*} M_i$ such that $p_r(\frac{mc+w}{m}, s^*) > p_r(t_{i_r}, s^L)$, and since $p_r(\frac{mc+w}{m}, s^*) > 0$,

$$C_r(s^*) \ge \frac{mc+w}{m} + p_r\left(\frac{mc+w}{m}, s^*\right).$$

We have

$$L_{\max}(s^*) \ge C_q(s^*) - d'_q \ge \frac{mc+w}{m} + p_q\left(\frac{mc+w}{m}, s^*\right) - d'_q \ge \frac{mc+w}{m} + p_q\left(t_{i_q}, s^L\right)$$

$$- d'_q \ge \frac{mc+w}{m} + p_x\left(t_{i^*}, s^L\right) - d'_x.$$

Hence,

$$L_{\max}(s^L) = t_{i^*} + p_x\left(t_{i^*}, s^L\right) - d'_x = c + w + p_x\left(t_{i^*}, s^L\right) - d'_x$$

$$= \frac{mc+w}{m} + \frac{m-1}{m}w + p_x\left(t_{i^*}, s^L\right) - d'_x \le \frac{m-1}{m}w + L_{\max}(s^*).$$

Let $t_{i'}$ be the largest t_i for which interval $[t_{i-1}, t_i]$ is incomplete. By Lemma 3.1 $|M_{i^*}| > m$, and since $|M_{i'}| < m$, according to Lemma 3.2 there exist tasks $j' \in M_{i'}$ and $j \in M_{i^*}$ such that $j' \to j$. Therefore, Lemma 3.2 implies that $l \ge w + p_{\min}$, and

$$L_{\max}(s^L) - L_{\max}(s^*) \leq \frac{m-1}{m} w \leq \frac{m-1}{m}(l - p_{\min}).$$

In order to show that Eq. (12) is achievable, consider the graph depicted in Fig. 5. Nodes in the first (top) row do not precede any other nodes and the corresponding tasks have the same due date equals one. Tasks in each subsequent row have due date of the previous row plus one. All tasks have the same processing time equals one. It is easy to see that

$$L_{\max}(s^L) = \frac{(l-1)(m-1)}{m} + l - 2.$$

In the optimal schedule s^* the tasks from the first row are processed in parallel with the chain of $l-1$ tasks preceding all tasks from the last (bottom) row. Since $L_{\max}(s^*) = l - 2$, Eq. (12) is an equality for any value of l.

4. Conclusions

This paper presents the performance guarantees for the non-preemptive and preemptive versions of the Brucker–Garey–Johnson algorithm. The Brucker–Garey–Johnson algorithm is a straightforward generalization of the classical Hu's algorithm. The presented results compliment the results on the worst-case performance of Hu's algorithm (Singh and Zinder, 2000) and its preemptive counterpart known as the Muntz–Coffman algorithm (Lam and Sethi, 1977). Each performance guarantee presented in this paper is in the form of an upper bound on the deviation of the criterion value from its optimum. The bounds are expressed in terms of the number of processors and parameters characterizing the partially ordered set of tasks, and therefore, establish a relationship between these parameters and the performance of the algorithm. Another feature of the presented bounds is their achievability for arbitrary large instances of the respective problems. The presented results will be complimented by the

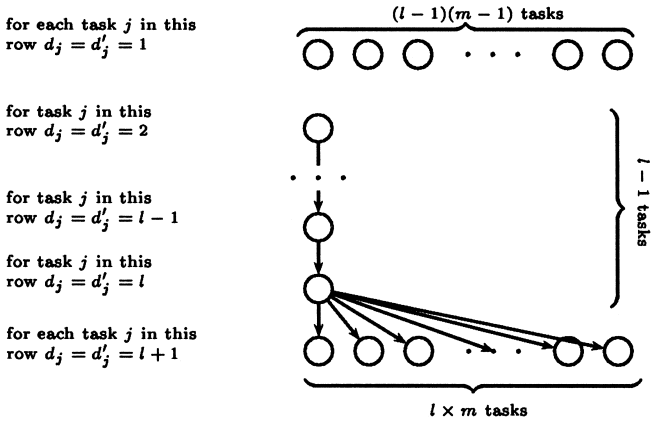

Fig. 5. The partially ordered set of tasks considered in Theorem 3.1.

computational experiments which constitute a part of the ongoing research. The design of approximation algorithms with better than known performance guarantees also remains an open problem for both makespan (Schuurman and Woeginger, 1999) and maximum lateness problems.

References

Brucker, P., Garey, M.R., Johnson, D.S., 1977. Scheduling equal-length tasks under tree-like precedence constraints to minimise maximum lateness. Math. Oper. Res. 2, 275–284.

Hu, T.C., 1961. Parallel sequencing and assembly line problems. Operations Research 9, 841–848.

Lam, S., Sethi, R., 1977. Worst case analysis of two scheduling algorithms. SIAM J. Comput. 6, 518–536.

Lawler, E.L., 1982. Preemptive scheduling of precedence-constrained jobs on parallel machines. In: Demster, M.A.H, Lenstra, J.K., Rinnooy Kan, A.H.G. (Eds.), Deterministic and Stochastic Scheduling, pp. 101–123.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., 1993. Sequencing and scheduling: algorithms and complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (Eds.), Logistics of Production and inventory. Elsevier, Amsterdam.

McNaughton, R., 1959. Scheduling with deadline and loss functions. Management Sci. 6, 1–12.

Parker, R.G., 1995. Deterministic Scheduling Theory. Chapman and Hall, New York.

Schuurman, P., Woeginger, G.J., 1999. Polynomial time approximation algorithms for machine scheduling: ten open problems. J. Sched. 2, 203–213.

Singh, G., Zinder, Y., 2000. Worst-case performance of two critical path type algorithms. Asia Pacific J. Oper. Res. 17(1) (to appear).

Ullman, J.D., 1975. NP-Complete scheduling problems. J. Comput. System Sci. 10, 384–393.

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