Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 302 (2003) 319–335

Theoretical Computer Science

www.elsevier.com/locate/tcs

On maximizing the throughput of multiprocessor tasks☆

Aleksei V. Fishkin, Guochuan Zhang*,[1]

Institut für Informatik und Praktische Mathematik, Universität Kiel, Olshausenstrasse 40, 24098 Kiel, Germany

Received 31 May 2002; accepted 7 November 2002
Communicated by D.-Z. Du

Abstract

We consider the problem of scheduling n independent multiprocessor tasks with due dates and unit processing times, where the objective is to compute a schedule maximizing the throughput. We derive the complexity results and present several approximation algorithms. For the parallel variant of the problem, we introduce the first-fit increasing algorithm and the latest-fit increasing algorithm, and prove that their worst-case ratios are 2 and $2-1/m$, respectively ($m \geqslant 2$ is the number of processors). Then we propose a revised algorithm with a worst-case ratio bounded by $\frac{3}{2}-1/(2m)$ (m is odd) and $\frac{3}{2}-1/(2m-2)$ (m is even). For the dedicated variant, we present a simple greedy algorithm. We show that its worst-case ratio is bounded by $\sqrt{m}+1$. We straighten this result by showing that the problem cannot be approximated within a factor of $m^{1/2-\varepsilon}$ for any $\varepsilon > 0$, unless $NP = ZPP$.

Keywords: Multiprocessor task; Complexity; Approximation algorithm

1. Introduction

In the traditional theory of scheduling, each task is processed by only one processor at a time. However, due to the rapid development of parallel computer systems, new

☆ Supported by Alexander von Humboldt Foundation. A preliminary version of this paper appeared in Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS'02, pp. 269–279).

* Corresponding author.

E-mail addresses: avf@informatik.uni-kiel.de (A.V. Fishkin), gzh@informatik.uni-kiel.de (G. Zhang).

[1] On leave from Zhejiang University, Hangzhou, China. Research partially supported by National 973 Fundamental Research Project of China and NSFC.

0304-3975/03/$ - see front matter
doi:10.1016/S0304-3975(02)00850-2

theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor tasks, see e.g. [4,7].

In this paper we address the following multiprocessor scheduling problem. A set $T=\{T_1,T_2,\dots,T_n\}$ of n tasks has to be executed by a set of m processors $M=\{1,2,\dots,m\}$. Each processor can work on at most one task at a time and a task can (or may need to be) processed simultaneously by several processors. Each task T_j has a unit processing time $p_j=1$ and integral due date $d(T_j)$. Here we assume that all tasks are available at time zero and the objective is to maximize the *throughput* $\sum \bar{U}_j$, where $\bar{U}_j=1$ if task T_j is completed before or at time $d(T_j)$, and $\bar{U}_j=0$ otherwise.

We deal with two variants of this problem. In the *parallel* variant, the multiprocessor architecture is disregarded and for each task T_j there is given a prespecified number $size_j \in M$ which indicates that the task can be processed by any subset of processors of the cardinality equal to this number. In the *dedicated* variant, each task T_j requires the simultaneous use of a prespecified set of processors $fix_j \subseteq M$.

We call a task T_j *early* if it meets its due dates $d(T_j)$ ($\bar{U}_j=1$), and *lost* ($\bar{U}_j=0$) otherwise. An early task is also called *accepted*. A lost task will not be scheduled. We use D to denote the largest due date $\max_j d(T_j)$. We say that tasks have a *common* due date if $d(T_j)=D$ for all tasks T_j.

To refer to the two variants of the above scheduling problem, we use the standard notation scheme by Graham et al. [7]. Let $P|size_j, p_j=1|\sum \bar{U}_j$ denote the parallel variant and $P|fix_j, p_j=1|\sum \bar{U}_j$ denote the dedicated variant. If all tasks have a common due date D, we denote the two variants as $P|size_j, p_j=1, d(T_j)=D|\sum \bar{U}_j$ and $P|fix_j, p_j=1, d(T_j)=D|\sum \bar{U}_j$, respectively.

For simplicity, throughout this paper we use s_j instead of $size_j$ when we refer to the size of task T_j in the parallel variant and τ_j instead of fix_j when we refer to the subset of processors task T_j requires in the dedicated variant.

1.1. Known results

In classical scheduling theory, there are a lot of results known for the objective of minimizing the (weighted) number of *late* tasks $(w_j)U_j$, where $U_j=1$ if task T_j is completed after $d(T_j)$, and $U_j=0$ otherwise, e.g. [12,14,1]. In the multiprocessor setting, the previous research has mainly focused on the objectives of minimizing *the makespan* $C_{\max}$ and *the sum of completion times* $\sum C_j$. As a rule, scheduling multiprocessor tasks with unit processing times is a strongly *NP*-hard problem [13,10]. However, recently there have been proposed a number of different approximation algorithms, e.g. [2,5,13,15]. Up to our knowledge, no results are known for the multiprocessor tasks scheduling problem concerning the throughput objective. Note that it is more conventional in scheduling to minimize the number of tardy tasks, rather than maximizing the number of early tasks. In fact, the optimal value of the two objectives is the same. When we investigate optimal algorithms or prove complexity results, the criterion of minimizing the number of tardy tasks is used. However, when we study approximation results, the number of tardy tasks in an optimal schedule may be zero. In this case, a finite worst-case ratio cannot be derived. The same situation occurs in on-line scheduling [9]. The problems considered in this paper are hard problems (will be proved in successive section) and we focus on finding good approximation algorithms. It is thus reasonable to choose the objective of maximizing the number of early tasks.

1.2. Our results

In this paper, focusing on the throughput objective, we give the complexity results and present several approximation algorithms, for both parallel and dedicated variants of the problem. The quality of an approximation algorithm A is measured by its *worst-case ratio* defined as

$$R_A = \sup_T \{N_O(T)/N_A(T)\},$$

where $N_A(T)$ denotes the number of early tasks in the schedule produced by the approximation algorithm A, and $N_O(T)$ denotes the number of early tasks in an optimal schedule for a task set T. In this paper we sometimes simply use N_O and N_A instead of $N_O(T)$ and $N_A(T)$ if no confusion is caused.

In the first part of the paper we consider the parallel variant of the problem. We prove that it is strongly *NP*-hard and present a number of approximation algorithms. We start with two simple greedy algorithms, namely, FFI_s and LFI_s. We prove that the worst-case ratio of FFI_s is 2, and the worst-case ratio of LFI_s is $2-1/m$, respectively. Then, by refining the algorithm LFI_s we get an improved algorithm HA with the worst-case ratio at most $\frac{3}{2}-1/(2m)$ (m is odd) and $\frac{3}{2}-1/(2m-2)$ (m is even).

In the second part we consider the dedicated variant. Each dedicated task requires a subset of processors. Hence, two tasks that share a processor cannot be processed at the same time. If all tasks have a common due date, we can adopt the complexity result for Maximum Clique [8]. Accordingly, we prove that our problem cannot be approximated within $m^{1/2-\varepsilon}$ unless $NP=ZPP$, where $\varepsilon>0$ is any given small number. On the other hand, we are able to show that the worst-case ratio of a greedy algorithm does not exceed $\sqrt{m}+1$. To grip on the case of individual due dates, we generalize this algorithm and demonstrate that the bound $\sqrt{m}+1$ remains valid.

Interestingly, there are a number of different relations to some well-known combinatorial problems. Just beyond the relation to Maximum Clique, we can find that Bin Packing and Multiple Knapsack correspond to the parallel variant of our problem. We discuss this in successive section.

The paper is organized as follows: Section 2 presents the results on the parallel model, and Section 3 on the dedicated model. Conclusions are given in Section 4.

2. Scheduling parallel tasks

We are given a set $T=\{T_1,T_2,\ldots,T_n\}$ of n tasks and a set of m processors. Each task T_j has a unit processing time $p_j=1$, an integral due date $d(T_j)$, and requires s_j

processors for its processing. The goal is to maximize the *throughput*, i.e. the number of *early* tasks T_j that meet their due dates $d(T_j)$.

Theorem 1. *Problem* $P|size_j, p_j = 1|\sum \bar{U}_j$ *is strongly NP-hard.*

Proof. Recall 3-Partition [6]:

Instance: Set A of $3N$ elements, a bound $B \in \mathbf{Z}^+$, and a size $s(a) \in \mathbf{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a\in B} s(a) = NB$.

Question: Can A be partitioned into N disjoint sets $A_1, A_2, \ldots, A_N$ such that, for $1 \leqslant i \leqslant N$, $\sum_{a\in A_i} s(a) = B$?

We transform 3-Partition to our problem. First, we take a set of B processors. Next, we replace each $a \in A$ by a single task T_a with the unit processing time, size $s(a)$ and due date $D = N$. (There are $3N$ tasks and all of them have a common due date N.) Clearly, this instance can be constructed in polynomial time from the 3-Partition instance, and 3-Partition instance is *YES* if and only if all the tasks meet the common due date. Since 3-Partition is strongly *NP*-complete [6], our problem is strongly *NP*-hard. □

We then concentrate on some efficient approximation algorithms. The first simple approximation algorithm is as follows.

Algorithm. FFI_s (First Fit Increasing for $size_j$)
Reindex the tasks of T in non-decreasing order of sizes s_j. Select the tasks one by one and assign them as early as possible. If a task T_j cannot be assigned to meet its due date $d(T_j)$, it gets lost (will not be processed).

When all tasks have the same due date, the problem $P|size_j, p_j = 1, d_j = D|\sum \bar{U}_j$ is just the bin packing problem for maximizing the number of items packed, which was studied by Coffman et al. [3]. They presented an algorithm called *FFI* and proved that tight asymptotic worst-case ratio is $\frac{4}{3}$. Actually, their proof is also valid for the absolute worst-case ratio. *FFI* is the same as FFI_s. Therefore, for the common due dates problem, the worst-case ratio of FFI_s is not greater than $\frac{4}{3}$. Furthermore, the following instance shows that the bound $\frac{4}{3}$ for FFI_s is tight for any specified $m \geqslant 3$: assume that the common due date is 2, and there are two small tasks, each of which requires only one processor, and two large tasks, each of which requires $m-1$ processors. FFI_s can only schedule three of them, while the optimal value is 4.

Note that the problem with common due date is also a special case of the multiple knapsack problem where all items have the same weight. Recently, Kellerer [11] proved that the multiple knapsack problem admits a *PTAS*. Now we turn to the general case where each task has an individual due date.

We need some definitions. A task T_j is *large* if its size s_j is greater than $m/2$, and *small* otherwise. Let $0 < d_1 < \cdots < d_g = D$ be all distinct due dates, where $D = \max_j d_j$.

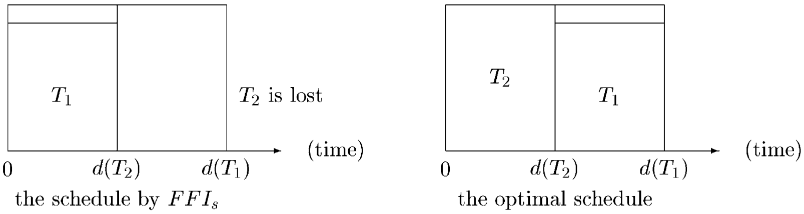

Fig. 1.

Then, we define time slots $I_t = (t-1, t]$, $t = 1, \ldots, D$. Consider any algorithm A scheduling tasks on time slots. We write $m(I_t)$ and N_t to denote the number of processors occupied and the number of tasks scheduled in time slot I_t, $t = 1, \ldots, D$. We say that I_t is *closed* if A meets the first task for which there is no room in I_t, and *open* if it is not closed yet.

Theorem 2. *For problem* $P|size_j, p_j = 1|\sum \bar{U}_j$, *the worst-case ratio* $R_{FFI_s} = 2$.

Proof. We first prove that $R_{FFI_s} \leqslant 2$. Consider an optimal schedule with N_O early tasks and the FFI_s schedule with N_{FFI_s} early tasks. Remove from the optimal schedule all the tasks involved in the FFI_s schedule and let ℓ_t be the number of the left tasks in time slot I_t. We prove that $\sum_{t=1}^{D} \ell_t$ is at most $N_{FFI_s} = \sum_{t=1}^{D} N_t$. In this case we get $N_{FFI_s} \geqslant N_O/2$.

Recall that all the ℓ_t left tasks in the optimal schedule are lost in the FFI_s schedule. Hence, in each time slot I_t the left ℓ_t tasks of the optimal schedule are not smaller in size than those of the FFI_s schedule. Since FFI_s schedules the tasks by non-decreasing order of sizes, the number of scheduled tasks N_t cannot be less than ℓ_t. Thus, we have $N_t \geqslant \ell_t$ for all $t = 1, \ldots, D$.

The bound is tight. Consider two tasks: task T_1 with size $s_1 = 1$ and due date $d(T_1) = 2$, and task T_2 with size $s_2 = m$ and due date $d(T_2) = 1$. In an optimal schedule both of the tasks meet their due dates, but FFI_s loses task T_2 (Fig. 1). □

The bad example tells us that the a task with larger due date may wait a moment to save space for some other task with smaller due date. The following simple algorithm takes into account this point.

Algorithm. LFI_s (Latest Fit Increasing for $size_j$)
Reindex the tasks in nondecreasing order of sizes. Select the ordered tasks one by one. If a task can be completed before or at its due date, assign it as late as possible provided that its due date can be met. If a task cannot be assigned to meet its due date, it gets lost (will not be processed).

In the schedule produced by algorithm LFI_s, we partition $(0, d_g] = \bigcup_{t=1}^{D} I_t$ into *blocks* $B(1), \ldots, B(l)$:

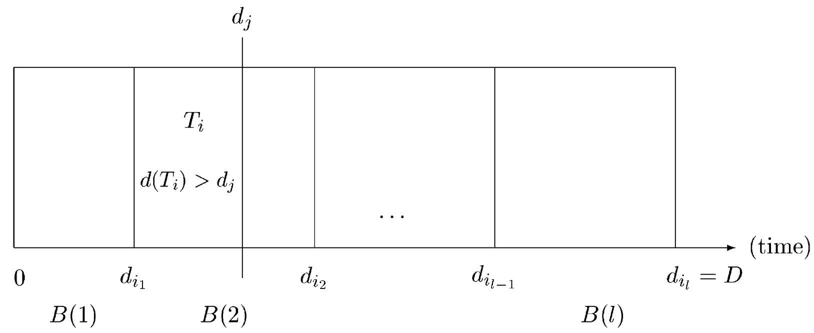

Fig. 2.

- the first block $B(1) = (0, d_{i_1}]$ with the smallest d_{i_1} such that all tasks T_j in $B(1)$ have due dates $d(T_j) \leqslant d_{i_1}$,
- each further block $B(s)$ is the smallest interval $(d_{i_{s-1}}, d_{i_s}]$ in which all tasks T_j have due dates $d_{i_{s-1}} < d(T_j) \leqslant d_{i_s}$.

Notice that it can happen that there is only one block or there are g blocks. Accordingly, we use $lost(s)$ and $sch(s)$ to denote the set of the lost tasks and the set of early tasks with due dates in block $B(s)$.

Fig. 2 shows that in a block, say $B(2)((d_{i_1}, d_{i_2}])$, for any due date d_j where $d_{i_1} < d_j < d_{i_2}$, there must exist some task T_i in this block, which starts before d_j and has due date larger than d_j (but no more than d_{i_2}).

Lemma 3. *For problem $P|size_j, p_j = 1|\sum \bar{U}_j$, the worst-case ratio R_{LFI_s} is at least $2 - 1/m$, where $m \geqslant 2$ is the number of processors.*

Proof. An example is constructed below. We are given m small tasks (denoted by s in Fig. 3), where task T_j has a size $s_j = 1$ and a due date $d(T_j) = j$, for $j = 1, \ldots, m$, and $m - 1$ large tasks (denoted by L in Fig. 3), where $s_j = m$ and $d(T_j) = m$, for $j = m + 1, \ldots, 2m - 1$. An optimal algorithm can complete all small tasks at time 1 and schedule the large tasks each in a time slot afterwards. Then all tasks are early. However, with algorithm LFI_s, only small tasks are scheduled and all large tasks are lost. Hence the worst-case ratio is at least $2 - 1/m$. □

Theorem 4. *For problem $P|size_j, p_j = 1|\sum \bar{U}_j$, the worst-case ratio $R_{LFI_s} = 2 - 1/m$.*

Proof. We show that $R_{LFI_s} \leqslant 2 - 1/m$, and then we complete by Lemma 3. We prove by a contradiction. Assume that $R_{LFI_s} > 2 - 1/m$. Accordingly, let $T_{\min}$ be the minimum task set, in terms of the number of tasks, such that $N_{\rm O}(T_{\min})/N_{LFI_s}(T_{\min}) > 2 - 1/m$. For all task sets T with $|T| < |T_{\min}|$, it follows $N_{\rm O}(T)/N_{LFI_s}(T) \leqslant 2 - 1/m$.

Assume that LFI_s runs on $T_{\min}$. Then, we can claim the following.

Lemma 5. *There are no open time slots.*

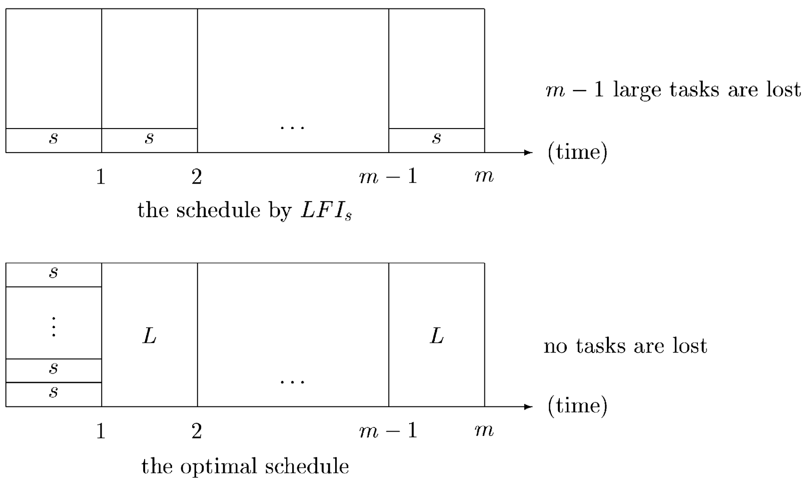

Fig. 3.

Proof. Assume that a time slot I_t is open, where $d_{i_s} < t \leqslant d_{i_{s+1}}$. Then, tasks j with due dates $d_j > d_{i_s}$ are early, and removing all these tasks from $T_{\min}$, we get a smaller set. It causes a contradiction. □

Lemma 6. *For each block $B(s)$, the size of any task in $lost(s)$ is not smaller than that of any task in $sch(s)$, $s = 1, \ldots, l$. Moreover, any task in $lost(s)$ cannot fit in any of the time slots in Block $B(s)$.*

Proof. We only need to consider the first lost task T_j in *lost*(s) (this task T_j has the smallest size among the tasks of *lost*(s)). Then its due date $d_{i_{s-1}} + 1 \leqslant d(T_j) \leqslant d_{i_s}$, and its size s_j is not less than the size of any tasks in *sch*(s) which have due dates at most $d(T_j)$. Moreover, $s_j + m(t) > m$ holds for each time slot I_t, $t = d_{i_{s-1}} + 1, \ldots, d(T_j)$. If $d(T_j) = d_{i_s}$, we have proved the lemma. Now we assume that $d(T_j) < d_{i_s}$. According to the definition of a block, there must be some task, which has a due date larger than $d(T_j)$, starts before time $d(T_j)$. Otherwise, Block $B(s)$ becomes $(d_{i_{s-1}} + 1, d(T_j)]$, which conflicts with the assumption. Denote by T_{j_1} the one with a largest due date among the tasks completed in $(d_{i_{s-1}} + 1, d(T_j)]$. Its due date and size are $d(T_{j_1})$ and s_{j_1}, respectively. Clearly $d(T_{j_1}) > d(T_j)$. Then $s_j \geqslant s_{j_1}$ and $s_j + m(t) \geqslant s_{j_1} + m(t) > m$ for each time slot I_t, $t = d(T_j) + 1, \ldots, d(T_{j_1})$. We then find task T_{j_2} with the largest due date among those tasks completed in $(d(T_j), d(T_{j_1})]$. Continue this process until we find task T_{j_k} among those tasks completed in $(d(T_{j_{k-2}}), d(T_{j_{k-1}})]$ such that $d(T_{j_k}) = d_{i_s}$. In the end, we have $s_j \geqslant s_{j_1} \geqslant \cdots \geqslant s_{j_k}$. Note the following two facts. For $u = 2, \ldots, k$,

- s_{j_u} is not less than the size of those tasks completed in the time period $(d(T_{j_{u-1}}), d(T_{j_u})]$.
- $s_{j_u} + m_t \geqslant m + 1$ for each time slot I_t, $t = d(T_{j_{u-1}}) + 1, \ldots, d(T_{j_u})$.

Therefore, s_j is not less than the size of any task $\in sch(s)$. Moreover, $s_j + m(t) > m$ for any time slot I_t, $t = d_{i_{s-1}} + 1, \ldots, d_{i_s}$. It means that T_j cannot fit in any time slot in Block $B(s)$. □

Analogously we can prove the following lemma.

Lemma 7. *For any two blocks $B(s)$ and $B(s')$ (with $s' < s$) the tasks of $lost(s)$ are not smaller in size than the tasks of $sch(s')$, and any task in $lost(s)$ cannot fit in any of the time slots of $B(s')$.*

Recall that the number of time slots is D. In the LFI_s schedule on $T_{\min}$, each time slot contains at least one task (Lemma 5). Hence $N_{LFI_s}(T_{\min}) \geqslant D$. For an optimal schedule π^* on $T_{\min}$, let h be the extra number of tasks accepted, i.e., $N_{\mathrm{O}}(T_{\min}) = N_{LFI_s}(T_{\min}) + h$. Let T^* and S^* be the set of early tasks and the total size of early tasks (in the optimal schedule π^*), respectively. Note that $S^* \leqslant mD$. We want to find a bound on h. To do this, we construct a set T_o of tasks by changing some tasks from T^* as follows.

1. Let $T_o = T^*$.
2. If T_o contains more than $|d_{i_1}| - 1$ tasks from $lost(1)$, some task T_p in $sch(1)$ must be lost in π^*, i.e., $T_p \notin T_o$. Replace a task $\in lost(1) \cap T_o$ by T_p. Continue this process until that T_o contains at most $|d_{i_1}| - 1$ tasks from $lost(1)$.
3. For $s = 2, \ldots, l$, the same as the above, we do as follows. If T_o contains more than $|d_{i_s}| - 1$ tasks from $\bigcup_{j=1}^{s} lost(j)$, some task T_q in $\bigcup_{j=1}^{s} sch(j)$ must be lost in π^*, i.e., $T_q \notin T_o$. Replace a task $\in \bigcup_{j=1}^{s} lost(j) \cap T_o$ by T_q. Continue this process until that T_o contains at most $|d_{i_s}| - 1$ tasks from $\bigcup_{j=1}^{s} lost(j)$.
4. For $s = l, \ldots, 1$, if $sch(s) \nsubseteq T_o$, i.e., a task $T_u \in sch(s) - T_o$, there must be some task $\in (\bigcup_{j=s}^{l} lost(j)) \cap T_o$. Replace such a task by T_u. Continue this exchange until that all tasks from $\bigcup_{s=1}^{l} sch(s)$ are involved in T_o.

In the above process, Steps 2 and 3 guarantee that the number of tasks in $(\bigcup_{j=1}^{s} lost(j)) \cap T_o$ is at most $d_{i_s} - 1$, for $s = 1, \ldots, l$. Step 4 ensures that T_o contains all tasks in $\bigcup_{s=1}^{l} sch(s)$. Furthermore,

- the number of tasks in T_o is $N_{\mathrm{O}}(T_{\min}) = N_{LFI_s}(T_{\min}) + h$;
- by Lemmas 6 and 7, the total size S_o of tasks in T_o does not increase when a replacement of tasks is made, which implies $S_o \leqslant S^* \leqslant mD$.

Note that T_o consists of all tasks involved in LFI_s schedule and h extra tasks (lost in the LFI_s schedule). By Lemmas 6 and 7, the total size of the h extra tasks and the tasks of the first h time slots in the LFI_s schedule is at least $h(m+1)$. The total size of the tasks in the remaining $D - h$ time slots in the LFI_s schedule is at least $D - h$, since each of these time slots contains at least one task (see an illustration in Fig. 4). Then $S_o \geqslant h(m+1) + (D-h)$. We have $mD \geqslant h(m+1) + (D-h)$ and then $h \leqslant D(m-1)/m$. However, since $N_{\mathrm{O}}(T_{\min})/N_{LFI_s}(T_{\min}) > 2 - 1/m$, i.e., $(N_{LFI_s}(T_{\min}) + h)/$

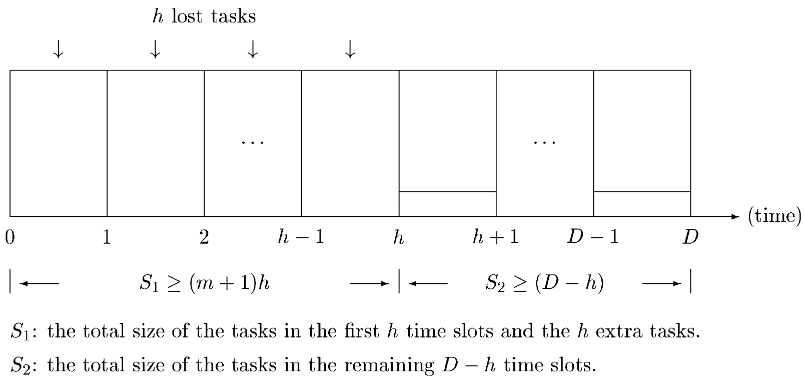

Fig. 4.

$N_{LFI_s}(T_{\min}) > 2 - 1/m$. Thus $h > D(m-1)/m$. It is a contradiction. The proof of Theorem 4 is complete. □

Theorem 8. *If all early tasks are small, the worst-case ratio of algorithm LFI_s is at most $\frac{3}{2} - 1/(2m-2)$ for problem $P|size_j, p_j = 1|\sum \bar{U}_j$.*

Proof. We can prove this theorem in the similar way as that of Theorem 4. Since all early tasks are small, by Lemma 5, each time slot contains at least two tasks. Hence $N_{LFI_s}(T_{\min}) \geqslant 2D$. Assume that an optimal schedule can accept h more tasks than algorithm LFI_s. Analogously, as the proof of Theorem 4, $h(m+1) + 2(D-h) \leqslant mD$. It implies that $h \leqslant D(m-2)/(m-1)$. Therefore,

$$\begin{aligned} N_O(T_{\min})/N_{LFI_s}(T_{\min}) &= (N_{LFI_s}(T_{\min}) + h)/N_{LFI_s}(T_{\min}) \\ &\leqslant 1 + (m-2)/(2m-2) \\ &= \tfrac{3}{2} + 1/(2m-2). \qquad \square \end{aligned}$$

Notice that both FFI_s and LFI_s attach importance to the task sizes. In some sense, FFI_s “groups” small tasks together, whereas LFI_s “spreads” them (see the above “bad” examples). Can we do something better?

It seems that *earliest due date* (*EDD*) rule—*schedule tasks in non-decreasing order of their due dates*—cannot help. For example, take k large tasks with $s_j = m$ and $d(T_j) = k$ ($j = 1, \ldots, k$), and $m(k+1)$ small tasks with $s_j = 1$ and $d(T_j) = k+1$ ($j = k + 1, \ldots, (m+1)(k+1)$). Then, $N_O = m(k+1)$, but *EDD* schedules only k large tasks and m small tasks. Thus, as $k \to \infty$, the ratio tends to m. However, we can combine all our ideas together. In the following a hybrid algorithm is presented.

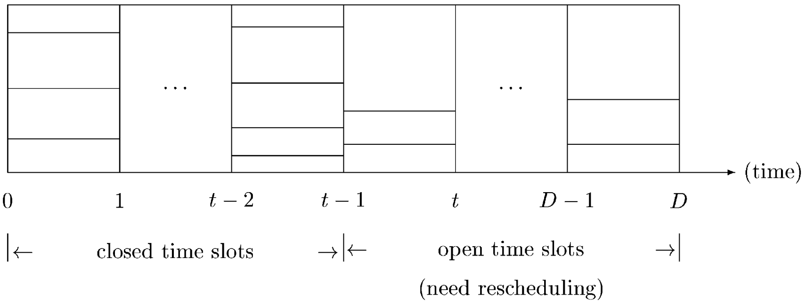

Fig. 5.

Algorithm. *HA* (Hybrid Algorithm)

1. Divide the tasks of T into small ones and large ones.
2. Schedule the set of small tasks by LFI_s. If there are no time slots open, go to Step 5.
3. Start from the first open time slot and go further taking the tasks in a slot and indexing them from the bottom of the slot. Then, reschedule the indexed tasks in a first-fit manner.
4. If there is a time slot which contains a single small task, say T_{j_s}, put this task T_{j_s} into the set of large tasks.
5. Schedule the set of large tasks by *EDD*(ties broken in favor of smaller size).

Fig. 5 gives an illustration for algorithm *HA*.

Lemma 9. *For problem* $P|size_j, p_j = 1|\sum \bar{U}_j$, *the worst-case ratio of algorithm HA*

$$R_{HA} \geqslant \begin{cases} \frac{3}{2} - 1/(2k-2) & \text{if } m = 3k, \\ \frac{3}{2} - 1/(2k-1) & \text{if } m = 3k+1, \\ \frac{3}{2} - 1/(2k) & \text{if } m = 3k+2. \end{cases}$$

Proof. Consider the following instance. There are $3n$ tasks (the value of n related with m will be specified later). For $i = 1, \ldots, n$, exact three tasks have the due date i. Their sizes are denoted by x_i, y_i and z_i, respectively, where $x_i, y_i \leqslant z_i$; $x_i \geqslant x_{i+1}$; $y_i \geqslant y_{i+1}$; $z_i < z_{i+1}$. Furthermore, $x_i + y_i + z_i = m + 1$ and $x_{i+1} + y_{i+1} + z_i = m$ (we will prove that there exists such an instance below). Clearly, algorithm *HA* starts the task of size x_i and the task of size y_i at time $i-1$, and all tasks with size z_i are lost. Then the total number of tasks accepted is $2n$. An optimal algorithm can start the three tasks of respective sizes x_i, y_i and z_{i+1} together at time $i-1$ for $i = 1, n-1$, and schedule the task of size z_n at time $n-1$. The number of tasks accepted in an optimal way is $3n-2$. Thus $R_{HA} \geqslant (3n-2)/(2n) = \frac{3}{2} - 1/n$. Now we specify the value of n by considering the following three cases:

1. $m = 3k$. In this case $n = 2k - 2$. Let $x_i + y_i = 2k - i$ and $z_i = k + i + 1$, for $i = 1, \dots, 2k - 2$.
2. $m = 3k + 1$. In this case $n = 2k - 1$. Let $x_i + y_i = 2k - i + 1$ and $z_i = k + i + 1$, for $i = 1, \dots, 2k - 1$.
3. $m = 3k + 2$. In this case $n = 2k$. Let $x_i + y_i = 2k - i + 2$ and $z_i = k + i + 1$, for $i = 1, \dots, 2k$.

The lemma is proved. □

Theorem 10. *For problem $P|size_j, p_j = 1|\sum \bar{U}_j$, the worst-case ratio of algorithm HA*

$$R_{HA} \leqslant \begin{cases} \frac{3}{2} - 1/(2m) & \text{if } m \text{ is odd}, \\ \frac{3}{2} - 1/(2m-2) & \text{if } m \text{ is even}. \end{cases}$$

Proof. We first consider the case that there are no time slots open immediately after Step 2 of algorithm *HA*. Let N_1 be the number of small tasks accepted and N_2 be the number of large tasks accepted. Then $N_{HA} = N_1 + N_2$. Let I_k be the last time slot (in time) containing small tasks. Obviously no small tasks have due dates greater than k. Thus any optimal algorithm can only accept $D - k$ tasks (large tasks) after time k. Assume that there are h tasks more in an optimal schedule. Following the same line of the ideas as in the proof of Theorem 8, we can bound the number h as follows: $h(m+1) + 2(k-h) \leqslant mk$. Hence $h \leqslant k(m-2)/(m-1)$ and

$$\begin{aligned} N_O/N_{HA} &= (N_1 + N_2 + h)/(N_1 + N_2) \leqslant 1 + h/(2k) \\ &\leqslant 1 + (m-2)/(2m-2) \\ &= \tfrac{3}{2} - 1/(2m-2). \end{aligned}$$

Now we consider the case that there is an open time slot after implementing Step 2 of algorithm *HA*. Let I_t be this time slot. We divide the tasks into three groups: (S1) small tasks completed before I_t, i.e. in the closed time slots; (S2) small tasks rescheduled at or after I_t except the small task T_{j_s} (if any) from Step 4; (L) large tasks scheduled and the small task T_{j_s} (if any) from Step 4. The tasks of (S1) have due dates smaller than t, and we can use Theorem 8. The tasks of (S2) have due dates at least t, and all of the small tasks with due dates at least t are accepted. Let k_{S2} be the number of time slots occupied by the tasks of (S2). Each of these time slots contains at least two tasks. Let N_{S1}, N_{S2} and N_L be the number of tasks in (S1), (S2) and (L), respectively. Then, $N_{HA} = N_{S1} + N_{S2} + N_L$.

We consider the following three cases: (a) task T_{j_s} shares a time slot with a large task; (b) task T_{j_s} stays alone; and (c) there is no task T_{j_s}.

We start with the last case. Take the optimal schedule. Assume that h_1 more tasks are accepted in the first $t-1$ time slots, and h_2 more tasks are accepted after time $t-1$. Then $N_O \leqslant N_{HA} + h_1 + h_2$. Analogous to the above analysis, we get $h_1 \leqslant (t-1)(m-2)/(m-1)$.

From $N_{S1} \geqslant 2(t-1)$ (there are $t-1$ time slots), we have

$$(N_{S1}+h_1)/N_{S1} \leqslant \tfrac{3}{2} - 1/(2m-2).$$

In the following we prove $(N_{S2}+N_L+h_2)/(N_{S2}+N_L) \leqslant \frac{3}{2} - 1/(2m-2)$ when m is even. Suppose that it does not hold. Then

$$h_2 > (m-2)(N_{S2}+N_L)/(2m-2). \tag{1}$$

Tasks in (S2) and (L) occupy $k_{S2}+N_L$ time slots, at most one of which has free space no more than $m/2$ (free space is the number of idle processors at the time). Since time slot I_t is open, the lost tasks with due date at least t are large tasks. It implies that the h_2 extra tasks accepted in the optimal schedule are large tasks, each of which requires at least $m/2+1$ processors (m is even). Since at most $k_{S2}+N_L$ large tasks can be assigned to $k_{S2}+N_L$ time slots, $h_2 \leqslant k_{S2}$.

If we put any of the h_2 extra task into a time slot occupied by tasks of (S2) and (L), the total size of tasks is at least $m+1$. Therefore,

$$h_2(m+1)+(N_L+k_{S2}-h_2)(m/2+1) \leqslant m(N_L+k_{S2}),$$

or simplifying

$$h_2 \leqslant (m-2)(N_L+k_{S2})/m. \tag{2}$$

Note that $N_{S2} \geqslant 2k_{S2}$. Combining (1) and (2), we have $N_{S2} < (m-2)N_L$. From (1), $h_2 > N_{S2}/2 \geqslant k_{S2}$. Then $h_2 > k_{S2}$. It is a contradiction. Hence, when m is even,

$$(N_{S2}+N_L+h_2)/(N_{S2}+N_L) \leqslant \tfrac{3}{2} - 1/(2m-2)$$

and

$$\begin{aligned} N_O/N_{HA} &= (N_{S1}+h_1+N_{S2}+N_L+h_2)/(N_{S1}+N_{S2}+N_L) \\ &\leqslant \tfrac{3}{2} - 1/(2m-2). \end{aligned}$$

When m is odd, we can prove $(N_{S2}+N_L+h_2)/(N_{S2}+N_L) \leqslant \frac{3}{2} - 1/(2m)$. If it is not true, then

$$h_2 > (m-1)(N_{S2}+N_L)/(2m). \tag{3}$$

On the other hand, similar to the above we have

$$h_2(m+1)+(N_L+k_{S2}-h_2)(m+1)/2 \leqslant m(N_L+k_{S2})$$

or

$$h_2 \leqslant (m-1)(N_L+k_{S2})/(m+1). \tag{4}$$

Combining (3) and (4), and noting that $N_{S2} \geqslant 2k_{S2}$, we get $h_2 > k_{S2}$. It is a contradiction. Thus,

$$N_O/N_{HA} = (N_{S1}+h_1+N_{S2}+N_L+h_2)/(N_{S1}+N_{S2}+N_L) \leqslant \tfrac{3}{2} - 1/(2m).$$

Finally, in case (a) we regard the time slot with j_s as one of the slots constructed by the small tasks of (S2), and in case (b) we can regard the time slot with j_s as one of the slots constructed by the large tasks of (L). The above analysis remains valid in both cases. □

3. Scheduling dedicated tasks

In the dedicated problem, the different aspect from the parallel problem is that each task T_j requires for its processing a set τ_j of processors. Consider the special case that all tasks have due date $D=1$, denoted by $P(1)$ ($P|fix_j, p_j=1, d(T_j)=1|\sum \bar{U}_j$). In the following we investigate the relationship between $P(1)$ and MC (Maximum Clique).

A $P(1)$ instance: n tasks with unit processing time are given, each of which requires a subset τ_j of processors. The common due date is 1. The goal is to schedule as many early tasks as possible.

An MC instance: A graph $G(V,E)$ is given, where $|V|=n$. The goal is to find a Maximum Clique, i.e., a subset $V'\subseteq V$ such that any two vertices in V' are joined by an edge in E and $|V|$ is maximum.

$P(1)\rightarrow MC$: Suppose we have a $P(1)$ instance. Now we construct an MC instance. Each task corresponds to a vertex. Two vertices are adjacent if and only if the two corresponding tasks are compatible (do not share any processors). Then scheduling maximum number of tasks in the time slot $[0,1]$ is equivalent to finding a maximum clique in the graph.

$MC\rightarrow P(1)$: Suppose we have an MC instance. Consider the complementary graph $\bar{G}(V,\bar{E})$, in which each vertex V_i corresponds to a task T_i and each edge e_j corresponds to a processor j. If V_i is a vertex of edge e_j, then the task T_i requires the processor j for its processing. Moreover, the degree of a vertex in the complementary graph is just the number of processors the corresponding task requires. Obviously the number of processors is no more than $n(n-1)/2$, where n is the number of vertices (tasks). Then finding a maximum clique from graph $G(V,E)$ is equivalent to scheduling as many tasks as possible in the time slot $[0,1]$. Furthermore, the optimal value of MC is the same as that of $P(1)$.

Theorem 11. *Unless $NP=ZPP$, $P(1)$ is not approximable within $m^{1/2-\varepsilon}$ for any given small positive number ε.*

Proof. Assume that there exists a polynomial time algorithm with a worst-case ratio $m^{1/2-\varepsilon_0}$ for some number $\varepsilon_0>0$ for problem $P(1)$. Then for any MC instance this algorithm can approximate MC with a worst-case ratio $|E|^{1/2-\varepsilon_0}$ where $|E|$ is the number of edges. Note that $|E|\leqslant n(n-1)/2$, where n is the number of vertices. Thus, the worst-case ratio of the algorithm is at most $n^{1-2\varepsilon_0}$. However, Hastad [8] showed that for Maximum Clique problem, there does not exist a polynomial time algorithm with a worst-case ratio $n^{1-\varepsilon}$ for any given small positive number ε, unless $NP=ZPP$. It is a contradiction. Therefore, the theorem holds. □

Clearly the lower bound holds for the general problem $P|fix_j, p_j=1|\sum \bar{U}_j$. We first consider the case that all tasks have a common due date $D\geqslant 1$.

Algorithm. *FFI_f* (First Fit Increasing for *fix_j*)
Sort the tasks in non-decreasing order of the number of processors they require such that if $i<j$, $|\tau_i|\leqslant|\tau_j|$. Arrange the tasks from the list with First Fit before time D. If a task can not be completed before or at time D, it is lost (will not be processed).

As defined before, denote by N_{O} and N_{FFI_f} the number of early tasks scheduled by an optimal algorithm and by algorithm FFI_f, respectively.

Theorem 12. *For problem $P|fix_j, p_j=1, d_j=D|\sum \bar{U}_j$, FFI_f has a worst-case ratio no more than $\sqrt{m}+1$ but at least $\sqrt{m}$.*

Proof. Let L_t and L_t^* be the set of tasks scheduled in time slot $I_t=[t-1,t]$ in the FFI_f schedule and an optimal schedule, respectively, where $t=1,\ldots,D$. Let $lost_t^*$ be the tasks in L_t^*, which are lost in the FFI_f schedule. Without loss of generality, assume that there are k tasks in L_t, denoted by $T_1,\ldots,T_k$, and $|\tau_1|\leqslant|\tau_2|\leqslant\cdots\leqslant|\tau_k|$. Clearly any task in $lost_t^*$ is blocked by some task in L_t. Let B_1^* be the tasks in $lost_t^*$ blocked by T_1. For $i=2,\ldots,k$, denote by B_i^* the tasks in $lost_t^*-(B_1^*\cup\cdots\cup B_{i-1}^*)$, which are blocked by T_i. If $|B_i^*|\leqslant\sqrt{m}$, $|lost_t^*|\leqslant\sqrt{m}|L_t|$. Then we have $N_{\mathrm{O}}\leqslant(\sqrt{m}+1)N_{FFI_f}$. The worst-case ratio of algorithm FFI_f is not greater than $\sqrt{m}+1$. In the following we prove that $|B_i^*|\leqslant\sqrt{m}$.

Note that $|B_i^*|\leqslant\min\{|\tau_i|, m/|\tau_i|\}$. It can be observed from the following facts:

- T_i occupies $|\tau_i|$ processors. By removing T_i, at most $|\tau_i|$ lost tasks can be scheduled. Thus $|B_i^*|\leqslant|\tau_i|$.
- For each lost task $T_j\in B_i^*$, $|\tau_j|\geqslant|\tau_i|$. Thus at most $m/|\tau_i|$ lost tasks can be scheduled by removing task T_i.

Then $|B_i^*|\leqslant\min\{|\tau_i|, m/|\tau_i|\}\leqslant\sqrt{m}$.

The following simple instance shows that the worst-case ratio $R_{FFI_f}\geqslant\sqrt{m}$. Given $\sqrt{m}+1$ tasks, each of which has $|\tau_j|=\sqrt{m}$ and the common due date is 1. The last $\sqrt{m}$ tasks are compatible with each other, but are incompatible with the first one. Then $N_{\mathrm{O}}=\sqrt{m}$ and $N_{FFI_f}=1$. □

The above bounds are valid for general m. However, for some specified m, the algorithm may have a better performance ratio. It is trivial that for $m=2$ the algorithm provides an optimal schedule. For $m=3$, the ratio is $\frac{4}{3}$, which can be proved below.

Lemma 13. *FFI_f has a worst-case ratio $\frac{4}{3}$ for $m=3$.*

Proof. The following instance shows that the worst-case ratio of FFI_f for $m=3$ is at least $\frac{4}{3}$. There are four tasks T_1,T_2,T_3,T_4, where $\tau_1=\{1\}$, $\tau_2=\{3\}$, $\tau_3=\{2,3\}$ and $\tau_4=\{1,2\}$. The common due date is two. In the optimal schedule all tasks can be executed to meet the due date, whereas in the FFI_f schedule, T_3 or T_4 gets lost.

In the following we prove that $\frac{4}{3}$ is an upper bound of the worst-case ratio of algorithm FFI_f. Consider the minimum counterexample in terms of the number of tasks. Let T be the task set in the counterexample, i.e., $N_O(T)/N_{FFI_f}(T) > \frac{4}{3}$ and $|T|$ is minimum. The tasks can be divided into three groups G_1, G_2 and G_3, where $G_i = \{T_j \| \tau_j| = i\}$ for $i = 1, 2, 3$.

- We first prove that in the counterexample, $G_3 = \emptyset$. If it is not true, let $T_p \in G_3$. If T_p is an early task in an optimal schedule, we can construct an instance T' as follows: removing task T_p from T and decreasing the common due date by 1. Obviously, for instance T', the optimal value $N_O(T') = N_O(T) - 1$ and $N_{FFI_f}(T') \leqslant N_{FFI_f}(T) - 1$. $N_{FFI_f}(T')/N_O(T') > \frac{4}{3}$. The number of tasks in T' is smaller than that in T. It conflicts with the assumption that T is a smallest set . If T_p is lost in an optimal schedule, we construct an set T' by removing task T_p from T. There exists a counterexample with a smaller task set, which causes a contradiction too.
- Next, all tasks in G_1 are early tasks. In case that some task in G_1 is lost, removing such a task results in a smaller counterexample in terms of number of tasks.
- Furthermore, in the schedule by FFI_f, at any moment at most one processor is idle. If at some time (no more than D) all the three processors are idle, no tasks get lost and then the schedule is optimal. If at some time two processors are idle, without loss of generality, assume that the two idle processors are 1 and 2. Then processor 3 is always busy during the time $[0, D]$. Let T_q be a lost task in the FFI_f schedule. Clearly T_q requires processor 3. Note that in an optimal schedule there must be a lost task which requires processor 3, since the number of tasks requiring processor 3 is more than D. After some necessary exchanges, T_q is also a lost task in an optimal schedule. Removing T_q we get a smaller counterexample which induces a contradiction.

Let N_{O1} and N_{O2} be the number of early tasks in an optimal schedule, which belong to G_1 and G_2, respectively. Let N_{F1} and N_{F2} be the number of early tasks involved in the FFI_f schedule, which belong to G_1 and G_2, respectively. Note that $N_{O1} = N_{F1}$. Then $N_O(T) = N_{O1} + N_{O2}$ and $N_{FFI_f}(T) = N_{O1} + N_{F2}$. In the counterexample, $N_O(T)/N_{FFI_f}(T) > \frac{4}{3}$. It implies that $3N_{O2} > N_{O1} + 4N_{F2}$. Recall that in the FFI_f schedule at any time at most one processor is idle. In any unit time slot if there is at most one task of G_1, a task of G_2 must be scheduled at the moment. There are at least $D - N_{O1}/2$ such unit time slots, i.e., $N_{F2} \geqslant D - N_{O1}/2$. Then we have $3N_{O2} > 4D - N_{O1}$. Note that $N_{O1} + 2N_{O2} \leqslant 3D$. Thus $N_{O2} > D$. It is impossible. Therefore the lemma holds. □

Finally, we consider the general case that each task has individual due date.

Algorithm. LFI_f (Latest Fit Increasing)
Sort the tasks in non-decreasing order of the number of processors they require such that if $i < j$ then $|\tau_i| \leqslant |\tau_j|$. Arrange the tasks from the list with Latest Fit before their due dates $d(T_j)$. If a task can not be arranged to meet its due date, it is lost (will not be processed).

Theorem 14. *LFI_f has a worst-case ratio no more than $\sqrt{m}+1$ but at least $\sqrt{m}$.*

Proof. The proof is similar to the one of Theorem 12. □

4. Conclusions

In this paper we have considered the scheduling problem to maximize the number of early multiprocessor tasks on both dedicated processors and parallel processors. For the parallel model, several heuristics have been proposed and analyzed. For general m, the best algorithm we have obtained has a worst-case ratio no more than $\frac{3}{2}$, and this bound is asymptotically tight. For the dedicated model, no polynomial-time algorithms can have a worst-case ratio $m^{1/2-\varepsilon}$ for any $\varepsilon > 0$, while we have shown that a greedy algorithm has a worst-case ratio at most $\sqrt{m}+1$. Although multiprocessor task scheduling has been studied extensively, the objective of maximizing throughput is new. Our work raises the following questions: For the parallel variant, is there a *PTAS* or is it *APX*-Hard? How is the approximability for the general case that the processing times of tasks are non-identical? Another interesting question is designing on-line algorithms for the problem.

Acknowledgements

We would like to thank Klaus Jansen for valuable discussions.

References

[1] P. Brucker, Scheduling Algorithms, Springer, Berlin, 1998, pp. 217–218.
[2] X. Cai, C.-Y. Lee, C.-L. Li, Minimizing total completion time in two-processor task systems with prespecified processor allocation, Naval Res. Logist. 45 (1998) 231–242.
[3] E.G. Coffman, J.Y.-T. Leung, D.W. Ting, Bin packing: maximizing the number of pieces packed, Acta Inform. 9 (1978) 263–271.
[4] M. Drozdowski, Scheduling multiprocessor tasks—an overview, European J. Oper. Res. 94 (1996) 215–230.
[5] A. Feldmann, J. Sgall, S.-H. Teng, Dynamic scheduling on parallel machines, Theoret. Comput. Sci. 130 (1994) 49–72.
[6] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.
[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[8] J. Hastad, Clique is hard to approximate within $n^{1-\varepsilon}$, Acta Math. 182 (1999) 105–142.
[9] H. Hoogeveen, C.N. Potts, G.J. Woeginger, On-line scheduling on a single machine: maximizing the number of early jobs, Oper. Res. Lett. 27 (2000) 193–197.
[10] J.A. Hoogeveen, S.L. Van de Velde, B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processor allocations, Discrete Appl. Math. 55 (1994) 259–272.
[11] H. Kellerer, A polynomial time approximation scheme for the multiple knapsack problem, RANDOM-APPROX, 1999, pp. 51–62.
[12] E.L. Lawler, Sequencing to minimize the weighted number of tardy jobs, RAIRO Recherche Opéra. 10 (1976) 27–33.

[13] E.L. Lloyd, Concurrent task systems, Oper. Res. 29 (1981) 189–201.
[14] C.L. Monma, Linear-time algorithms for scheduling on parallel processors, Oper. Res. 37 (1982) 116–124.
[15] J. Turek, W. Ludwig, J. Wolf, P. Yu, Scheduling parallel tasks to minimize average response times, Proc. 5th ACM-SIAM Symp. on Discrete Algorithms, 1994, Arlington, Virginiia, ACM/SIAM, pp. 112–121.