Poznań University of Technology

Scheduling malleable tasks for mean flow time criterion

Massimiliano Caramia, Maciej Drozdowski

Research Report RA-008/05

2005

Institute of Computing Science, Piotrowo 3A, 60-965 Poznań, Poland

Scheduling malleable tasks for mean flow time criterion

M.Caramia[1], M.Drozdowski[2]

Abstract

In this paper we study scheduling malleable tasks with limited parallelism, for mean flow time criterion. Malleable tasks may use more than one processor at the same time, and the number of processors used may change over time. The maximum number of processors that can be used by some task is limited. We examine the computational complexity of this problem, and present polynomially solvable cases.

Keywords: Deterministic scheduling, malleable tasks, mean flow time.

1 Introduction

Malleable tasks can be executed by more than one processor at the same time. Furthermore, the number of used processors can be changed over the course of a task execution. Malleable task model may be applied to represent parallel applications executed in environments in which migration is possible. For example, on a parallel computer with shared memory a parallel application can create threads. These threads can be executed simultaneously. Operating system assigns the threads to the processors for time quanta in a round-robin fashion, and preempts the threads when the quanta expire. When the load of the computer system is low all the application threads may run in parallel in real time. When the load is increasing, operating system assigns the application threads to fewer processors. Thus, the number of processors used over time can be changed according to the decisions of the operating system. An upper limit on the number of usable processors may exist. This may be either the number of threads created by the application, or a limit imposed by the operating system protecting its resources

[1]Istituto per le Applicazioni del Calcolo, CNR, V.le del Policlinico, 137 - 00161 Rome, Italy.

[2]Institute of Computing Science, Poznan University of Technology, Piotrowo 3A, 60-965 Poznan, Poland. The research of this author has been partially supported by Polish Committee for Scientific Research. Corresponding Author. Email: Maciej.Drozdowski@cs.put.poznan.pl

from overuse. Another example of malleable tasks is in bandwidth allocation. Bandwidth of a communication link is a resource which can be divided among many simultaneously operating channels. The bandwidth assigned to a channel may vary over time. However, the channels have an upper limit on the usable bandwidth (e.g. Peak Cell Rate in ATM networks). A router must divide the bandwidth between the simultaneous communications such that the maximum for each channel is not exceed. Malleable task model for loom production scheduling has been presented in [9]. A single request for production of a certain fabric can be distributed over several looms. The number of looms used during the course of satisfying the request may vary.

The scheduling problem studied in this paper can be formulated in the following way. Set $\mathcal{T}$ of n tasks is to be executed on set $\mathcal{P}$ of m parallel identical processors. Each task $j \in \mathcal{T}$ is defined by the parameters: processing requirement p_j, maximum number of processors that can be used δ_j, ready time r_j, and deadline d_j which cannot be exceeded in any feasible schedule. Tasks can be suspended, and restarted later without any additional cost. Each task can migrate to a different processor, increase or decrease the number of used processors, also without cost. The only restriction is that no more than δ_j processors can be used simultaneously. To verify if task j has received the required processing and can be finished, one has to calculate the area occupied by task j in the time $\times$ processors space, and compare it with p_j. The completion time of task j will be denoted by c_j. The objective is the minimization of the mean flow time $\frac{1}{n}\sum_{j=1}^{n}(c_j - r_j)$. Since $\sum_{j=1}^{n} r_j$ is constant for any instance, the minimization of the mean flow time is equivalent to the minimization of $\sum_{j=1}^{n} c_j$. Therefore, in the following discussion we will refer to the minimization of $\sum_{j=1}^{n} c_j$ as to the mean flow time criterion.

Malleable task scheduling has been considered in earlier publications. The first works considering parallel tasks, i.e. tasks executed on several processors simultaneously, seem to be [8], and [1]. Unfortunately, the lack of generally accepted terminology may confuse. It is often the case that the name *malleable tasks* is applied to parallel tasks that can be executed on several processors, but the number of processors must be selected before the task starts, and cannot be changed during the execution of the task. We follow the naming conventions proposed in [4, 6] where such tasks are called *moldable*. We do not consider moldable tasks here. The concept of malleable, moldable, and more generally parallel tasks, and the problems of scheduling them have been presented in [3, 4, 6]. The first study of scheduling malleable tasks appeared in [10]. Tasks had due-dates, and the objective was the minimization of maximum lateness. This problem can be solved by means of binary search and maximum network flows. Scheduling chains of three malleable tasks for schedule length criterion has been studied in [5]. The first, and the last task in the chain had parallelism limited to one processor ($\delta_j = 1$). The second, central task had unlimited parallelism ($\delta_j \geq m$). This problem has been shown to be **NP**-hard, and special cases solvable in polynomial time have been identified [5]. However, to our best knowledge not much is known about the problem of scheduling malleable tasks for the mean flow time criterion.

The rest of this paper is organized as follows: In Section 2 we study the complexity of the proposed problem. Section 3 is dedicated to the case of fixed sequences of task completion times. A low-order polynomial time algorithm is proposed in Section 4 for *agreeable* processing times and parallelism maxima.

2 Complexity of the problem

In this section we demonstrate that the problem of scheduling malleable tasks with bounded parallelism is **NP**-hard in general.

Theorem 1 *The problem of scheduling malleable tasks with limited parallelism, ready times, and deadlines, for mean flow time criterion is* **NP***-hard.*

Proof. We start by proving that this problem is in **NP**. A solution of the problem can be represented as a set of intervals in which the number of processors assigned to the tasks does not change. The number of such intervals is $O(n^2)$ because ready times, completion times, and deadlines define $O(n)$ periods, in which the processor assignment to the tasks changes $O(n)$ times (we discuss it in more detail in the next section). For each interval one has to verify if no task uses more than the admissible number of processors. By summing the amounts of work performed on the tasks in the consecutive intervals one can verify that each task is fully completed. Finally, the mean flow time is calculated by checking the sum of completion times of the tasks.

We will show now a polynomial time transformation from the problem PARTITION INTO EQUAL CARDINALITY SUBSETS [7] to a decision version of our problem. PARTITION INTO EQUAL CARDINALITY SUBSETS is defined as follows:
Instance: a set of $2k$ integers $A = \{a_1, \ldots, a_{2k}\}$, such that $\sum_{j=1}^{2k} a_j = 2B$.

Question: is it possible to partition A into two disjoined subsets A_1, A_2 such that $|A_1| = |A_2| = k$, and $\sum_{j \in A_1} a_j = \sum_{j \in A_2} a_j = B$?

The decision version of our problem is defined as follows:

Instance: set $\mathcal{T}$ of n malleable tasks with processing requirements $p_j \in Z^+$, maximum number of usable processors $\delta_j \in Z^+$, ready times $r_j \in Z^+$, and deadlines $d_j \in Z^+$, for $j = 1, \dots, n$, integer m, a positive rational number y. Question: is it possible to execute tasks from set $\mathcal{T}$ such that $\sum_{j=1}^{n} c_j \le y$?

The polynomial time transformation is defined as follows:
$m = kMB^2 - B^2; n = 2k + 2;$
$\delta_j = MB^2 - a_j B$ for $j = 1, \dots, 2k;$
$p_j = \delta_j + a_j$ for $j = 1, \dots, 2k;$
$r_j = 0, d_j = \infty$ for $j = 1, \dots, 2k;$
$\delta_{2k+1} = m - 1;\ p_{2k+1} = (m-1)B;\ r_{2k+1} = 0; d_{2k+1} = B;$
$\delta_{2k+2} = m;\ p_{2k+2} = mL;\ r_{2k+2} = B + 1; d_{2k+2} = B + L + 1;$
$y = B + B + 1 + L + k(B + 1) + k(B + 1 + L) + 2k;$
where $L > (B + 3)k$, and $M > k$ are big constants. Tasks $1, \dots, 2k$ will be called partition tasks, task $2k + 1, 2k + 2$ will be called blocking tasks.

Let us assume that there is a partition into equal cardinality subsets. Then, a feasible schedule for our problem can be as the one presented in Fig.1. Task $2k + 1$ is finished at time B, task $2k + 2$ is finished at time $B + 1 + L$, and the k tasks corresponding to the elements of set A_1 are completed at time $B + 1$. The k tasks corresponding to the elements $j \in A_2$ complete at times $B + 1 + L + \frac{p_j}{\delta_j}$, where $\frac{p_j}{\delta_j} = \frac{3B^2 - a_j(B-1)}{3B^2 - a_j B} \le 2$. Together we obtain mean flow time $\sum_{j=1}^{n} c_j = B + B + 1 + L + k(B + 1) + k(B + 1 + L) + \sum_{j \in A_2} \frac{p_j}{\delta_j} \le y$

Suppose that a feasible schedule with mean flow time at most y exists. We will demonstrate that also a partition into equal cardinality subsets must exist. Note that by the selection of their ready times, deadlines, and the shortest execution times $\frac{p_k}{\delta_k}, \frac{p_{k+1}}{\delta_{k+1}}$, tasks $2k + 1$, and $2k + 2$ must be finished at times B and $B + 1 + L$, respectively. This leaves free intervals $[0, B]$ with one processor, $[B, B + 1], [B + 1 + L, \infty)$ with m processors, available for the partition tasks $1, \dots, 2k$.

Let us observe that in order to have mean flow time not greater than y, at least k tasks from the set $1, \dots, 2k$ must be completed before task $2k + 2$, i.e. before time $B + 1$. Suppose it is otherwise and $x > k$ partition tasks are completed after task $2k + 2$. Then, the sum of the completion times for the blocking tasks, and x partition tasks completed after task $2k + 2$ is at least

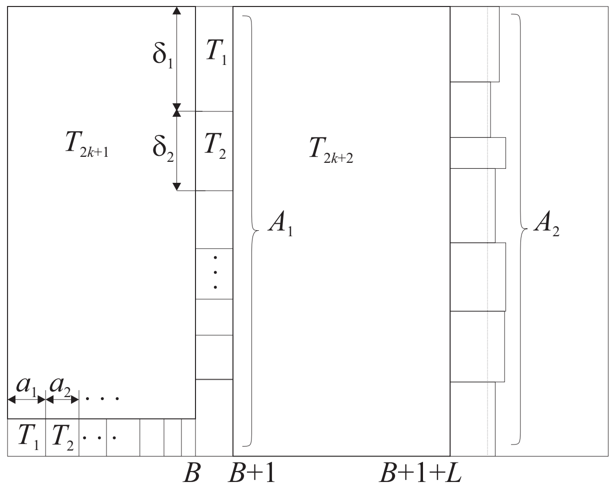

Figure 1: Illustration to the proof of Theorem 1.

$B + B + 1 + L + x(B + 1 + L) \geq B + B + 1 + L + (k + 1)(B + 1 + L) > y = B + B + 1 + L + k(B + 1) + k(B + 1 + L) + 2k$ because $L > k(B + 3)$. On the other hand at most k partition tasks can be completed before time $B+1$. Suppose it is otherwise and $x > k$ partition tasks are completed before $B + 1$. Since for each partition task $p_j = MB^2 - a_jB \geq MB^2 - B^2$, these tasks require at least $x(MB^2 - B^2) \geq (k + 1)(MB^2 - B^2)$ processing, while the available area is $B + m = B + kMB^2 - B^2$, which is smaller because $M > k$. Hence, no more than k partition tasks can be completed before $B + 1$. Together we have that exactly k partition tasks must be completed before $B + 1$. If we denote the set of tasks completed before $B + 1$ by A_1, and the rest as A_2, then we have $|A_1| = |A_2|$.

Note that a partition task j executed in the interval $[B, B+1]$ can receive at most δ_j processing. The remaining part $p_j - \delta_j = a_j$ must be processed in the interval $[0, B]$. Now we will prove that also $\sum_{j\in A_1} a_j = \sum_{j\in A_2} a_j = B$. Suppose that it is otherwise, and $\sum_{j\in A_1} a_j > B$. Then, $\sum_{j\in A_1}(p_j - \delta_j) = \sum_{j\in A_1} a_j \geq B + 1$, at least one unit of work must be processed after $B + 1$, and the criterion value y is not met. Note that there is free space in the interval $[B, B + 1]$ in the amount of $m - \sum_{j\in A_1} \delta_j = kMB^2 - B^2 - kMB^2 + B\sum_{j\in A_1} a_j > B$, but it cannot be exploited by any partition task in A_1 because the maximum number of processors is already used. Suppose that $\sum_{j\in A_1} a_j < B$. Then, the total processing requirement of the tasks in A_1 is $\sum_{j\in A_1} p_j = \sum_{j\in A_1}(\delta_j + a_j) = \sum_{j\in A_1}(MB^2 - a_j(B-1)) \geq kMB^2 - (B-1)(B-$

$1) = kMB^2 - B^2 + 2B - 1$ which is greater than the space $B + kMB^2 - B^2$ available in $[0, B+1]$. Hence, tasks in A_1 cannot be feasibly completed before $B+1$. Thus, we conclude that a feasible schedule not exceeding mean flow time y exists if $\sum_{j\in A_1} a_j = \sum_{j\in A_2} a_j = B$, and the answer to partition with equal cardinality subsets is also positive. □

3 Fixed sequences

In this section we present a linear programming solution for the case when the sequence of task completions, ready times and deadlines are known. We start the presentation with a simpler case.

3.1 Fixed sequence of completion times

In this paragraph we assume that all tasks are available at time 0, and have not bounding deadlines (e.g. $\forall j\ d_j = \infty$). Without loss of generality let us assume that the sequence of task completions is $c_1 \le c_2 \le \ldots \le c_n$. Let us denote by x_{ij} the amount of processing that task j receives in the interval $[c_{i-1}, c_i]$, for $i = 1, \ldots, n$. For completeness of arguments we assume $c_0 = 0$. The linear program is as follows:

minimize $\sum_{i=1}^{n} c_i$
subject to:

$$x_{ij} \le \delta_j(c_i - c_{i-1}) \qquad j = 1, \ldots, n;\ i = 1, \ldots, j \tag{1}$$

$$\sum_{j=i}^{n} x_{ij} \le m(c_i - c_{i-1}) \qquad i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{j} x_{ij} \ge p_j \qquad j = 1, \ldots, n \tag{3}$$

In the above linear program inequalities (1) guarantee that no task j uses more than δ_j processors in the interval $[c_{i-1}, c_i]$. By inequalities (2) tasks processed in the interval $[c_{i-1}, c_i]$ use no more processing than the capacity of the m processors. Inequalities (3) ensure that all tasks receive necessary processing.

Though the above linear program includes constraints necessary for feasibility of a schedule, it is not known yet if a feasible schedule can be constructed using the solution of (1)-(3). A feasible schedule can be built using an extension of McNaughton's algorithm proposed in [5] for schedule length criterion (C_{max}). We describe the extension for the sake of completeness of the presentation. Tasks with processing requirements p_j, and parallelism bound δ_j, can be scheduled in time

$$C_{max} = \max\left\{\max_j\left\{\frac{p_j}{\delta_j}\right\}, \frac{1}{m}\sum_{j=1}^{n} p_j\right\}. \tag{4}$$

This is necessarily a lower bound because no schedule can be shorter than the length of the longest task or the total processing requirement equally distributed on all processors. A schedule of this length is built by using McNaughton's wrap-around rule. However, here if a task is wrapped it may use more than one processor at the same time. By the selection of $C_{max} \geq \max_j\left\{\frac{p_j}{\delta_j}\right\}$ it is guaranteed that no task j uses more than δ_j processors simultaneously. Let us return now to scheduling the pieces x_{ij} of the tasks in the intervals $[c_{i-1}, c_i]$. By constraints (1)-(2), pieces x_{ij} fulfill condition (4) imposed by the extended McNaughton rule, and can be feasibly scheduled in the intervals $[c_{i-1}, c_i]$.

We conclude this section with an example in which we have $m = 4$ processors, and three tasks such that $c_1 \leq c_2 \leq c_3$, $p_1 = 2$, $p_2 = 5$, $p_3 = 4$, and $\delta_1 = 2$, $\delta_2 = 4$, $\delta_3 = 1$. The linear program is as follows:

minimize $c_1 + c_2 + c_3$
subject to:

$$\begin{aligned}
x_{11} &\leq 2c_1\\
x_{12} &\leq 4c_1\\
x_{22} &\leq 4(c_2 - c_1)\\
x_{13} &\leq c_1\\
x_{23} &\leq (c_2 - c_1)\\
x_{33} &\leq (c_3 - c_2)\\
x_{11} + x_{12} + x_{13} &\leq 4c_1\\
x_{22} + x_{23} &\leq 4(c_2 - c_1)\\
x_{33} &\leq 4(c_3 - c_2)\\
x_{11} &\geq 2\\
x_{12} + x_{22} &\geq 5
\end{aligned}$$

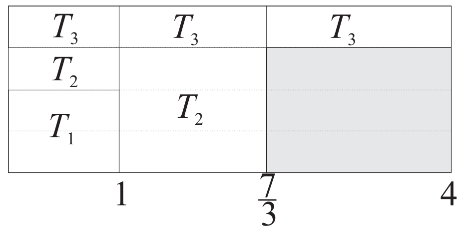

Figure 2: Illustration to the example in Section 3.1.

$$x_{13} + x_{23} + x_{33} \geq 4 \tag{5}$$

By solving the above linear program we obtain: $x_{11} = 2, x_{12} = 1, x_{13} = 1, x_{22} = 4, x_{23} = \frac{4}{3}, x_{33} = \frac{5}{3}, c_1 = 1, c_2 = \frac{7}{3}, c_3 = 4$. The optimal schedule is depicted in Fig.2.

3.2 Fixed sequence of all events

When the sequence of r_j's, d_j's, and c_j's is fixed, our problem can be formulated as a linear program. Let us consider simultaneously all such events: ready times, due dates, completion times. We will denote the number of these events by l. Let τ_i and τ_{i+1} denote the endpoints of an interval determined by two consecutive events, for $i = 1, \ldots, l-1$. Note that τ_i is a constant if it represents a ready time, or a deadline. τ_i is a variable if event i is a completion time. Thus, we have the following formulation:

minimize $\sum_{i=1}^{l} \tau_i$
subject to:

$$\begin{aligned}
x_{ij} &\leq \delta_j(\tau_i - \tau_{i-1}) && i = 1, \ldots, l && (6)\\
\sum_{j=1}^{n} x_{ij} &\leq m(\tau_i - \tau_{i-1}) && i = 1, \ldots, l && (7)\\
\sum_{i=1}^{l} x_{ij} &\geq p_j && j = 1, \ldots, n && (8)\\
x_{ij} &= 0 \text{ if } \tau_{i-1} < r_j && i = 1, \ldots, l && (9)\\
x_{ij} &= 0 \text{ if } \tau_i > d_j && i = 1, \ldots, l && (10)
\end{aligned}$$

The main difference with respect to yhe linear program (1)-(3) is that in the above formulation we consider consecutive events which are not necessarily two completion times. Though, the objective function is a sum of time instants of all events, ready times and deadlines are fixed, and the sum of the τ_i's corresponding to them is constant. Therefore, minimizing $\sum_{i=1}^{l} \tau_i$ is equivalent to minimizing $\sum_{i=1}^{n} c_i$. Furthermore, we force to zero x_{ij} in inequalities (9) and (10), for those intervals i which are before the availability of task j, or after the deadline of task j.

Note that for a fixed number of tasks, the number of possible permutations of task completion times, ready times, and deadlines is also fixed. Hence, we have an observation.

Observation 1 *The problem of scheduling malleable tasks with ready times and deadlines is solvable in polynomial time for any fixed number of tasks.*

4 Agreeable processing requirements and parallelism maxima

In this section we study a special case of *agreeable* processing requirements and parallelism bounds. For this case a low-order polynomial time algorithm can be given.

By agreeable processing requirements, and parallelism bounds we mean the instances for which tasks can be ordered such that $\frac{p_1}{\delta_1} \leq \frac{p_2}{\delta_2} \leq \ldots \leq \frac{p_n}{\delta_n}$ and $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_n$. The agreeable feature of an instance can be checked in $O(n \log n)$ time by sorting the tasks. We also assume $r_j = 0, d_j = \infty$, for all tasks j. The algorithm can be formulated as follows:

Algorithm Agreeable
1: **for** j:=1 **to** n **do**
2: assign task j to the earliest possible time intervals using maximum possible number of processors, i.e. either δ_j or all the processors remaining available in a given time interval.

Let us illustrate this algorithm with an example. Processing requirements are given in a vector $\overline{p} = [2, 4, 4, 5, 7]$, parallelism bounds are given in a vector $\overline{\delta} = [1, 1, 2, 2, 4]$, $m = 5$. The schedule built by algorithm Agreeable is shown in Fig.3.

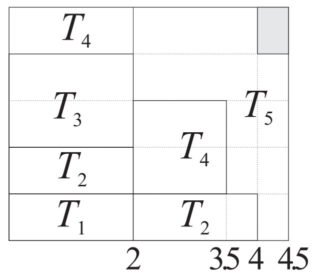

Figure 3: Illustration to the example in Section 4.

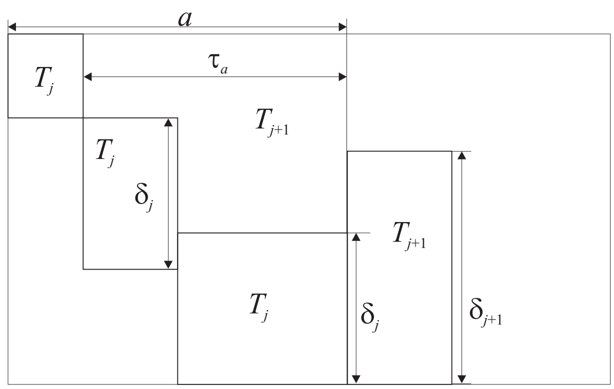

Figure 4: Illustration to the proof of Theorem 2.

Let us make some observations about the schedules built by algorithm Agreeable. Let α_j denote the number of processors used by task j at the end of its execution in a schedule constructed by algorithm Agreeable.

Theorem 2 $\alpha_j = \min\{\delta_j, m\}$, *for* $j = 1, \ldots, n$.

Proof. The proof is inductive in nature. The theorem is satisfied for $j = 1$. Assume it is satisfied for tasks $1, \ldots, j$, where $j \geq 1$. Let us consider the time interval a in which task j is executed (cf. Fig.4). Interval a is the earliest possible time where task $j + 1$ can be executed because there are no idle intervals to the left of a. Otherwise task j would have been shifted to such earlier intervals.

a) Suppose there are some free processors in interval a, and task $j+1$ fits completely in interval a. Let τ_a denote the length of the sub-interval with free processors within a. We have $\tau_a \leq \frac{p_j}{\delta_j}$ because task j may be executed also before the sub-interval with free processors. On the other hand for task $j+1$

we have $\frac{p_{j+1}}{\delta_{j+1}} \leq \tau_a$ because $j+1$ fits completely in the interval. Together we get $\frac{p_{j+1}}{\delta_{j+1}} \leq \tau_a \leq \frac{p_j}{\delta_j}$. But due to the agreeable condition $\frac{p_{j+1}}{\delta_{j+1}} \geq \frac{p_j}{\delta_j}$. Consequently, $\frac{p_{j+1}}{\delta_{j+1}} = \tau_a = \frac{p_j}{\delta_j}$, and $\alpha_{j+1} = \delta_{j+1}$. Furthermore, if one task is completely processed in parallel with some other task then they are finished simultaneously.

b) Suppose $j+1$ does not fit completely in the interval a. Thus, $c_{j+1} > c_j$. It follows from the previous case that after completion of task j all processors are free because all tasks executed in parallel with j finish no later than by c_j. Hence, $\alpha_{j+1} = \min\{m, \delta_{j+1}\}$. □

Theorem 3 *Algorithm Agreeable constructs the optimum schedule in $O(n^2)$ if $\frac{p_1}{\delta_1} \leq \frac{p_2}{\delta_2} \leq \ldots \leq \frac{p_n}{\delta_n}$ and $\delta_1 \leq \delta_2 \leq \ldots \leq \delta_n$.*

Proof. This proof has inductive nature.
1) Schedule task 1 using $\alpha_1 = \min\{m, \delta_1\}$ processors. Mean flow time c_1 is minimum.
2) Suppose an optimum schedule for tasks $1, \ldots, j$ is constructed by algorithm Agreeable. We schedule task $j+1$ using algorithm Agreeable. $\sum_{i=1}^{j+1} c_i$ cannot be reduced by:
a) reducing $\sum_{i=1}^{j} c_i$ because the schedule for tasks $1, \ldots, j$ is optimal,
b) reducing only c_{j+1} because it is infeasible,

Thus, reducing c_{j+1} and increasing $\sum_{i=1}^{j} c_i$ is the only way of reducing $\sum_{i=1}^{j+1} c_i$. Suppose we reduce c_{j+1} by ε_{j+1}. This reduces the area available for task $j+1$ by $\varepsilon_{j+1}\alpha_{j+1}$ which must be compensated for by delaying the completion times of some tasks among $1, \ldots, j$. Without loss of generality, let them be tasks $1, \ldots, k$ and their completions are delayed by $\varepsilon_1, \ldots, \varepsilon_k$, respectively. This creates available area of at most $\sum_{i=1}^{k} \varepsilon_i \alpha_i$. This new area can be consumed by task $j+1$, in exchange for area $\varepsilon_{j+1}\alpha_{j+1}$. Thus, we reduce the completion time of task $j+1$ by no more than $\frac{\sum_{i=1}^{k} \varepsilon_i \alpha_i}{\alpha_{j+1}} \geq \varepsilon_{j+1}$. By Theorem 2 and agreeable condition $\alpha_i \leq \alpha_{j+1}$, for $i = 1, \ldots, k$. Hence, we have:

$$\varepsilon_1 + \ldots + \varepsilon_k \geq \frac{\sum_{i=1}^{k} \varepsilon_i \alpha_i}{\alpha_{j+1}} \geq \varepsilon_{j+1} \tag{11}$$

which means that the increase of the mean flow time by $\varepsilon_1 + \ldots + \varepsilon_k$ exceeds the reduction of ε_{j+1}. This conclusion can be invalidated only if some task(s) $i \in \{1, \ldots, k\}$ use $\alpha'_i > \alpha_{j+1}$ processors. Due to the agreeable condition, and Theorem 2, we have $\alpha'_i \leq \delta_i = \alpha_i \leq \alpha_{j+1}$ and (11) holds.

The complexity of the algorithm is a result of the fact that in step 2 of algorithm Agreeable the number of available processors for task j changes at most $n-1$ times, and at most this many times the remaining processing requirement of task j must be recalculated. $\Box$

5 Conclusions

In this paper we studied a problem of scheduling malleable tasks with bounded parallelism. The problem is **NP**-hard in the presence of ready times and deadlines. For fixed sequence of ready times, deadlines, and task completion times it can be solved in polynomial time by use of linear programming. When processing requirements and parallelism bounds of the tasks are agreeable, a low-order polynomial time algorithm was proposed. Yet, the complexity of a more fragile problem of scheduling malleable tasks with bounded parallelism without ready times and deadlines remains open.

References

[1] J.Błażewicz, M.Drabowski, J.Węglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Transactions on Computers* **35**, No.5, (1986) 389-393.

[2] J.Błażewicz, K.Ecker, E.Pesch, G.Schmidt, J.Węglarz, *Scheduling Computer and Manufacturing Processes* (Springer, Berlin, 2001).

[3] M.Drozdowski, Scheduling multiprocessor tasks - an overview, *European Journal of Operational Research* **94**, (1996) 215-230.

[4] M.Drozdowski, Scheduling parallel tasks - Algorithms and complexity, in: J.Y.-T.Leung, ed., *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (Chapman & Hall/CRC, Boca Raton, 2004), chapter 25.

[5] M.Drozdowski, W.Kubiak, Scheduling parallel tasks with sequential heads and tails, *Annals of Operations Research* **90**, (1999) 221-246.

[6] D.G.Feitelson, L.Rudolph, U.Schweigelshohn, K.Sevcik, P.Wong, Theory and practice of job scheduling, *Lecture Notes in Computer Science* **1291** (Springer, Berlin, 1997) 1-34.

[7] M.Garey, D.Johnson, *Computers and Intractability - A Guide to the Theory of NP-completeness* (Freeman, New York, 1979).

[8] E.L.Lloyd, Concurrent task systems, *Operations Research* **29**, No.1 (1981) 189-201.

[9] P.Serafini, Scheduling jobs on several machines with the job splitting property, *Operations research* **44** (1996) 617-628.

[10] V.G.Vizing, Minimization of the maximum delay in servicing systems with interruption, U.S.S.R. Computational Mathematics and Mathematical Physics **22**, No.3 (1982) 227-233.

See also the following pages.

Note. This addendum comprises results announced in PMS06 abstract [1] as Theorem 2, but in fact not present in the original Technical Report of 2008. Here it is present as Theorem 4. This whole bunch of results was submitted to DAM, IPL, NRL in 2005 and rejected.

6 $\delta_j = 1$, or $\delta_j = m$

Note that if $\forall_j \delta_j = 1$ our problem boils down to the classic preemptive scheduling on m parallel processors, while if $\forall_j \delta_j \geq m$ then our problem is equivalent to scheduling on a single processor. If $\forall_j r_j = 0, d_j = \infty$ then executing the tasks according the shortest processing time provides the optimum schedule.

7 Approximation results

In this section we present approximation algorithms for our problem. Algorithm Agreeable can be applied in the general, not agreeable case (but still $\forall_j r_j = 0, d_j = \infty$). If the tasks are scheduled according to the increasing processing requirement, i.e. $p_1 \leq p_2 \leq \ldots \leq p_n$, then algorithm Agreeable builds schedules with mean flow time not worse than twice the optimum.

Theorem 4 *Algorithm Agreeable has the worst case performance ratio at most 2.*

Proof. While scheduling task j by algorithm Agreeable one can distinguish two intervals: when task j is executed in parallel with tasks $1, \ldots, j-1$ using less than δ_j processors and no processor is free, and an interval when j is executed on $\min\{m, \delta_j\}$ processors. The former interval ends no later than by $\sum_{i=1}^{j} \frac{p_i}{m}$, the latter interval is not longer than $\frac{p_j}{\delta_j}$. Hence, $c_j \leq \sum_{i=1}^{j} \frac{p_i}{m} + \frac{p_j}{\delta_j}$. The mean flow time is $\sum_{j=1}^{n} c_j \leq \sum_{j=1}^{n} (n-j+1) \frac{p_j}{m} + \sum_{j=1}^{n} \frac{p_j}{\delta_j}$. Let OPT denote the optimum mean flow time. Since $OPT \geq \sum_{j=1}^{n} (n-j+1) \frac{p_j}{m}$, and $OPT \geq \sum_{j=1}^{n} \frac{p_j}{\delta_j}$ the observation holds. □

Below we present an approximation algorithm which can be applied to solve our problem on-line, i.e. in the presence of different ready times, but in the absence of the deadlines.

Theorem 5 *There is an on-line approximation algorithm with the worst case performance ratio $4+\varepsilon$ with complexity $O(\frac{n^2}{\varepsilon}(\log \max_j\{p_j\} + \log \max_j\{r_j\} + \log n))$.*

Proof. The on-line approximation algorithm is an adaptation of the technique proposed in [2]. It is shown in [2] that there is a 4ρ-approximation algorithm provided that there exists a dual ρ-approximation algorithm. We present a dual $\rho = (1+\varepsilon)$-approximation algorithm, and outline the arguments of [2] for the completeness of the presentation.

The dual approximation algorithm solves the following problem: Find a maximum cardinality set $S \subseteq \mathcal{T}$ of tasks which can be completed by deadline D. By equation (4) tasks in S can be feasibly scheduled by D if $D \geq \max_{j\in S}\{\frac{p_j}{\delta_j}\}$, and $D \geq \frac{1}{m}\sum_{j\in S} p_j$. The tasks violating the first condition can be excluded because they cannot be feasibly scheduled before D. We have to select a maximum number of tasks such that $\sum_{j\in S} p_j \leq mD$. This is a simplified version of a knapsack problem: the sizes of the objects are p_js, their values are equal 1, the size of the knapsack is mD. Knapsack problem can be solved by the fully polynomial time algorithm proposed in [?]. Given $\varepsilon > 0$, let $\gamma = \frac{mD\varepsilon}{n}$. We round down the processing times and size of the knapsack to the multiples of γ, i.e. $p'_j = \lfloor\frac{p_j}{\gamma}\rfloor, D' = \lfloor\frac{mD}{\gamma}\rfloor$. For the rounded instance the optimum selection S' can be found by applying dynamic programming in time $O(nD')$ which is $O(\frac{n^2}{\varepsilon})$. Let S^* be the optimum selection for the unmodified instance. Since $mD \geq \sum_{j\in S^*} p_j \geq \sum_{j\in S^*} p'_j\gamma$, p'_j are integers, $\sum_{j\in S^*} p'_j \leq D'$. Hence, S^* is a feasible solution for the modified instance. On the other hand, $|S'| \geq |S^*|$ because S' is the optimum solution for the rounded instance. A schedule for tasks in S' has length $\frac{1}{m}\sum_{j\in S'} p_j \leq \frac{\gamma}{m}\sum_{j\in S'}(p'_j+1) \leq \frac{\gamma}{m}(D'+n) \leq (1+\varepsilon)D$. Thus, the set S' has been constructed which can be scheduled in interval $(1+\varepsilon)D$, and its cardinality is at least equal the maximum possible for deadline D.

Let $\tau_0 = \frac{1}{m}$, and $\tau_l = \frac{2^{l-1}}{m}$. The on-line algorithm constructs the schedule iteratively. In iteration l wait until time τ_l, invoke the dual approximation algorithm for deadline $D = \tau_l$ and the ready tasks which have not been scheduled yet. Schedule the resulting tasks in the set S' in the interval $[(1+\varepsilon)\tau_l, (1+\varepsilon)\tau_{l+1}]$. This construction is feasible because $(1+\varepsilon)(\tau_{l+1}-\tau_l) = (1+\varepsilon)\tau_l$. Let c'_j denote completion time of T_j in the above schedule, and c^*_j in the optimal one. It has been shown in [2] that $\sum_{j=1}^n c'_j \leq 4\sum_{j=1}^n c^*_j$. Since no schedule can be longer than $\max_j\{r_j\} + n\max_j\{p_j\}$, the number of iterations cannot be greater than $O(\log \max_j\{p_j\} + \log \max_j\{r_j\} + \log n)$, and the total

complexity of the algorithm is $O(\frac{n^2}{\varepsilon}(\log \max_j\{p_j\} + \log \max_j\{r_j\} + \log n))$.
$\square$

Note that it would be purely technical modification to apply the above algorithm for tasks with weights w_j and $\sum w_j c_j$ criterion with the same performance guarantees.

References

[1] M.Caramia, M.Drozdowski, Scheduling malleable tasks for mean flow time criterion, X International Workshop on Project Management and Scheduling PMS2006, April 26-28, 2006, Poznań, 106-109.

[2] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein., Scheduling to minimize average completion time: Off-line and on-line approximation algorithms, *Mathematics of Operations Research* **22**, (1997) 513.