Scheduling Independent Multiprocessor Tasks

A. K. Amoura[1], E. Bampis[2], C. Kenyon[3], and Y. Manoussakis[1]

[1] Université Paris Sud, LRI, Bâtiment 490,
91405 Orsay Cedex, France
[2] Université d'Evry, LaMI, Bvd des Coquibus,
91025 Evry Cedex, France
[3] ENS Lyon, LIP, URA CNRS 1398, 46 Allée d'Italie,
69364 Lyon Cedex 07, France

Abstract. We study the problem of scheduling a set of n independent multiprocessor tasks with prespecified processor allocations on a fixed number of processors. We propose a linear time algorithm that finds a schedule of minimum makespan in the preemptive model, and a linear time approximation algorithm that finds a schedule of length within a factor of $(1+\epsilon)$ of optimal in the non-preemptive model.

1 Introduction

A scheduling problem is usually given by a set T of n tasks, with an associated partial order which captures data dependencies between tasks, and a set P_m of m target processors. The goal is to assign tasks to processors and time steps so as to minimize an optimality criterion, for instance the makespan, *i.e.* the maximum completion time $C_{\max}$ of any task. Depending on the model, tasks can be preempted or not. In the *non-preemptive* model, a task once started has to be processed (until completion) without interruption. In the *preemptive* model, each task can be at no cost interrupted at any time and completed later. If there are no precedence constraints, the tasks are said to be *independent*.

In the most classic model, each task is processed by only one processor at a time. However, as a consequence of the development of parallel computers, new theoretical scheduling problems have emerged to model scheduling on parallel architectures. The *multiprocessor task system* is one such model[1] [7]. In this model, a task may need to be processed simultaneously by several processors. In the *dedicated* variant of this model, each task requires the simultaneous use of a prespecified set of processors. Since each processor can process at most one task at a time, tasks that share at least one resource cannot be scheduled at the same time step and are said to be *incompatible*. Hence, tasks are subject to compatibility constraints.

The present paper is concerned with *scheduling independent multiprocessor tasks on dedicated processors*, in both the preemptive case and the non-preemptive case, and the goal is to minimize the makespan $C_{\max}$. Following the

[1] See [6] for some practical justifications for considering multiprocessor task systems, and a survey on the main results obtained in multiprocessor task scheduling.

notations of [11,20], we will denote the problem by $P_m|\mathrm{fix}_j, \mathrm{pmtn}|C_{\max}$ in the preemptive case, and $P_m|\mathrm{fix}_j|C_{\max}$ in the non-preemptive case.

The preemptive problem. In [18], Kubale proved that the preemptive problem with an unbounded number of processors, $P_\infty|\mathrm{fix}_j, \mathrm{pmtn}|C_{\max}$, is strongly NP-hard even when $|\mathrm{fix}_j| = 2$, (*i.e.* when each task requires the simultaneous use of at most 2 processors). He used a reduction from the edge multi-coloring problem. However, he also showed that the preemptive problem with a fixed number m of processors, $P_m|\mathrm{fix}_j, \mathrm{pmtn}|C_{\max}$, can be solved by linear programming under the restriction that $|\mathrm{fix}_j| = 2$. On the other hand, Bianco et al. showed that if m is at most 4, then the preemptive problem $P_m|\mathrm{fix}_j, \mathrm{pmtn}|C_{\max}$ can be solved in linear time [1]. Recently, Krämer [16] showed that $P_m|\mathrm{fix}_j, \mathrm{pmtn}, r_j|L_{\max}$, which is a generalization of $P_m|\mathrm{fix}_j, \mathrm{pmtn}|C_{\max}$, can be solved by linear programming. In fact, he solves $O(\log n)$ linear programs each one with $O(n)$ constraints and $O(n^{m+1})$ variables.

In this paper, we also use a linear programming formulation to solve the preemptive problem with a fixed number m processors $P_m|\mathrm{fix}_j, \mathrm{pmtn}|C_{\max}$, in *linear time*, thereby improving the results of [16,18,1]. Our linear program is inspired from Kenyon and Remila's approximation scheme for strip-packing [14].

The non-preemptive problem. The non-preemptive three-processor problem, i.e. $P_3|\mathrm{fix}_j|C_{\max}$, has been extensively studied. In [2,13], it is proved to be strongly NP-hard by a reduction from 3-partition. The best approximation algorithm is due to Goemans [9] who proposed a $\frac{7}{6}$-algorithm improving the previous best performance guarantee of $\frac{5}{4}$ [5]. For unit execution time tasks, *i.e.* $P_m|\mathrm{fix}_j, p_j = 1|C_{max}$, the problem is solvable in polynomial time through an integer programming formulation with a fixed number of variables [13]. However, if the number of processors is part of the problem, *i.e.* $P|\mathrm{fix}_j, p_j = 1|C_{\max}$, the problem becomes NP-hard. Furhermore, Hoogeveen et al. [13] showed that, for $P|\mathrm{fix}_j, p_j = 1|C_{\max}$, there exists no polynomial approximation algorithm with performance ratio smaller than 4/3, unless $P = NP$.

In this paper, we use the linear programming formulation of the preemptive problem to get a polynomial time approximation scheme for the non-preemptive problem $P_m|\mathrm{fix}_j|C_{\max}$ for fixed m. The rough idea is that it does not matter much whether short tasks are preemptable or not; hence the tasks are separated into long tasks and short tasks, and long tasks are dealt with separately by trying essentially every possible arrangement. This is similar in spirit to Hall and Shmoys' polynomial time approximation scheme for single-machine scheduling [12]. Note that the running time, although linear in terms of n, is super-exponential in terms of $1/\epsilon$.

2 The algorithm in the preemptive model

Given any schedule of the tasks, if we take a snapshot at instant t, we see a certain set of tasks, which are all being processed at t. Thus we are led to consider configurations, which correspond to collections of tasks which can be processed simultaneously. With a linear programming approach, our algorithm finds partial schedules associated to the configurations. A global schedule is then produced by simply merging the obtained partial schedules.

We first need a few definitions. The *type of a task* is its associated set of required processors, which is a non-empty subset of $\{1, 2, \ldots, m\}$. In general, a *type* τ is just a subset of $\{1, 2, \ldots, m\}$. We say that types τ and τ' are *compatible* if $\tau \cap \tau' = \emptyset$. We define a *configuration* to be a maximum set of compatible task types. *I.e.*, a configuration is a collection $\mathcal{C}$ of non-empty types $\tau_i \subseteq \{1, \ldots, m\}$ such that:

1. $\forall \tau_i, \tau_j \in \mathcal{C}, \quad \tau_i \cap \tau_j = \emptyset$, *i.e.* types of $\mathcal{C}$ are pairwise compatible.
2. $\bigcup_{\tau_i \in \mathcal{C}} \tau_i = \{1, \ldots, m\}$, *i.e.* $\mathcal{C}$ is maximum.

Let $\#\mathcal{C}$ denote the total number of configurations, determined by the number m of processors.

Given: A fixed number m of processors.
Input: A set T of n multiprocessor dedicated preemptive tasks. Each task i is defined by its type and its processing time d_i.
Output: An optimal schedule.

The algorithm is in three stages:

1. **[Set up the Linear Program].** We partition the tasks of T, according to their types, into subsets

$$T^\tau = \{i \in T \mid \text{task } i \text{ has type } \tau\},$$

one for each possible task type. (We have $T = \bigcup_{\tau \subseteq \{1,\ldots,m\}} T^\tau$.) We define D^τ as the total processing time of all the tasks of type τ, *i.e.*

$$D^\tau = \sum_{i \in T^\tau} d_i.$$

Let us assign a variable x_i to each configuration $\mathcal{C}_i$, whose interpretation will be the length of the time interval during which, if we take a snapshot of the schedule during that interval, the types that we see form configuration $\mathcal{C}_i$. The problem can now be formulated by the following linear program.

$$\text{minimize} \sum_{i=1}^{\#\mathcal{C}} x_i \tag{1}$$

$$\text{subject to: } \forall \tau \subseteq \{1, \ldots, m\} : \sum_{i \text{ s.t. } \tau \in \mathcal{C}_i} x_i \geq D^\tau \tag{2}$$

The optimization criterion (1) means that the makespan must have minimum length subject to constraints (2) which guarantee that all the tasks are completed.

2. **[Solve the Linear Program].** Let $(x_1^*, x_2^*, \ldots, x_{\#\mathcal{C}}^*)$ denote the solution to the linear program.
3. **[Deducing a preemptive schedule].** A feasible optimal schedule can be constructed in a greedy way as follows. For each type τ, at any stage of the construction, some tasks of T^τ have already been completely scheduled, one task of T^τ may have been partially scheduled and preempted before completion, and the remaining tasks of T^τ have not yet been scheduled at all.
   Take the configurations $\mathcal{C}_i$ such that $x_i^* \neq 0$, in some arbitrary order. For each such configuration $\mathcal{C}_i$, we construct a partial schedule of length x_i^* as follows. Consider all the types $\tau \in \mathcal{C}_i$. For every such type τ, select first the partially scheduled task of T^τ (if there is one), then the remaining tasks of T^τ in arbitrary order, until the total processing time of the selected tasks becomes greater than or equal to x_i^* (or until all tasks of T^τ have been scheduled). If the total processing time is strictly greater than x_i^*, then the last selected task is preempted so that the total processing time is exactly x_i^*.

3 Analysis of the preemptive algorithm

Lemma 1. *The number of configurations is equal to $\mathcal{B}(m)$, the m^{th} Bell number.*

Proof. The number of configurations $\mathcal{C}$ having k elements, $1 \leq k \leq m$, is equal to the number of ways to partition a set of m elements into k nonempty disjoint subsets. But this is exactly the Stirling number of the second kind $S(m,k) = \left\{ {k \atop m} \right\}$ [15]. Therefore, the number of all possible configurations is given by $\mathcal{B}(m) = \sum_{k=1}^{m} S(m,k)$ (which corresponds to the m^{th} Bell (or exponential) number [10]). □

Remark 2. Using the fact that $S(m,k) = kS(m-1,k) + S(m-1,k-1)$ (see [15] p.73) we get the crude upper bound $\mathcal{B}(m) \leq m!$, hence the number of configurations is a constant (as the number of processors m is fixed) independent of the number n of tasks.

Theorem 3. *$P_m|fix_j, pmtn|C_{\max}$ can be solved in time $O(n)$.*

Proof. Correctness.

From the definition of a configuration, we see that within the same configuration tasks with different types are compatible and thus can be scheduled in parallel while tasks with the same type have to be scheduled consecutively. Hence, the obtained schedule is feasible. Furthermore, since tasks can be preempted, it is obvious that the above linear program gives the length $\sum_{i=1}^{\#\mathcal{C}} x_i^*$ of the optimal schedule.

Running time.

Step 1: The list of tasks is scanned once, and so this step takes time $O(n)$. Step 2: Since there are $(2^m - 1)$ types of tasks and $\mathcal{B}(m)$ configurations, the

linear program has $(2^m - 1)$ constraints and $\mathcal{B}(m)$ variables. Both values are constants and so the linear program can be solved in time $O(1)$.
Step 3: Each list T^τ is scanned once, and so this step takes time $O(n)$.
Overall the algorithm runs in time $O(n)$. $\square$

4 The algorithm in the non-preemptive model

We say that a task is *preemptable* if it can be at no cost interrupted at any time and completed later. We will be led to consider mixed models where some tasks are preemptable and others not.

A simple approach to obtain a feasible non-preemptive schedule from a preemptive schedule S is to remove the preempted tasks from S and process them sequentially in a naïve way at the end of S. The produced schedule has a makespan equal to the makespan of S plus the total processing time of the delayed tasks. However, even if S is optimal preemptive, the schedule thus produced is almost certainly not close to optimal, since there can be a large number of possibly long preempted tasks.

Our algorithm does use a preemptive schedule to construct a non-preemptive schedule in the way just described. However, to ensure that the non preemptive solution is close to optimal, only "short" tasks are allowed to be preempted. The intuitive idea of the algorithm is the following: first, partition the set of tasks into "long" tasks and "short" tasks. For each possible schedule of the long tasks, complete it into a schedule of T, **assuming that the short tasks are preemptable**. From each such preemptive schedule construct a non-preemptive schedule. Finally, among all the schedules thus generated, choose the one that has the minimum makespan.

In what follows, for simplicity we give the algorithm for the three-processor case. Our approach can be directly generalized to deal with the m-processor case for any constant m.

We first need a few definitions. Given a set of tasks L, a *snapshot* of L is a set of compatible tasks of L. A *relative schedule* of L is a sequence of snapshots of L such that each task occurs in a subsequence of consecutive snapshots, and consecutive snapshots are different. Roughly speaking, each relative schedule corresponds to an order of processing of the long tasks. Note that to any given non-preemptive schedule of L, one can associate a relative schedule in a natural way by looking at the schedule at every instant where a task of L starts or ends, and writing the set of tasks of L being processed right after that transition. Also note that the last snapshot is then the empty set.

Recall that in section 2, we defined types and configurations. We now also need to define τ-configurations. Given a type τ, a τ-configuration is a collection of types $\tau_i \subseteq \tau$ such that:

1. $\forall \tau_i, \tau_j \in \mathcal{C}_\tau, \quad \tau_i \cap \tau_j = \emptyset$, *i.e.* types of $\mathcal{C}_\tau$ are pairwise compatible.
2. $\bigcup_{\tau_i \in \mathcal{C}} \tau_i = \tau$.

Let $\#\mathcal{C}_\tau$ denote the number of τ-configurations.
Let D denotes the total processing time of all the tasks, *i.e.* $D = \sum_{i=1}^{n} d_i$. The algorithm is in six stages.

Given: Three processors.

Input: A set T of n multiprocessor dedicated non-preemptive tasks, each task i being defined by its type and its processing time d_i; and a constant ϵ such that $0 < \epsilon < 1$.

Output: A schedule with makespan $C_{\max} \leq (1+\epsilon)C_{\max}^{opt}$, where $C_{\max}^{opt}$ is the optimal makespan.

1. **[Partition tasks according to lengths].** Sort the $7^{1/\epsilon}$ longest tasks, of lengths $d_1 \geq \ldots \geq d_{7^{1/\epsilon}}$. Find $i \leq 1/\epsilon$ such that $d_{7^i} + \ldots + d_{7^{i+1}-1} < \epsilon$. Let $k = 7^i$. Perform the partition $T = T' \cup L$ where L is the set of tasks with the k longest execution times.
2. **[Construct the relative schedules].** List all possible relative schedules $M = (M(0), M(2), \ldots, M(f-1))$ of L.
3. **[Set up linear programs.]** During this stage of the algorithm, we assume that the tasks of T' are preemptive. For each relative schedule M of L, we determine an optimal schedule for the tasks of T such that the associated relative schedule is M; this is done using the following linear program.

   Variables. There are three types of variables. First, there is one variable t_i for each snapshot of M; it represents the instant in the schedule corresponding to the snapshot $M(i)$. Thus t_{f-1} represents the time at which the last task of L terminates its execution. In addition, a variable t_f represents the time at which the last task of T terminates its execution. Second, there is one variable e_τ for each type τ, such that e_τ represents the total time during which the processors of $\{1,2,3\} \setminus \tau$ are processing tasks of L and the processors of τ are not processing tasks of L, *i.e.* processors of τ are either processing tasks of T' or idle. Third, for each type τ and each τ-configuration $\mathcal{C}_i(\tau)$, we define a variable $x_i(\tau)$. These variables play the same role as the variables x_i of section 2, *i.e.* they help determine the schedule during the e_τ amount of time during which the processors of τ are available to process tasks of T'.

   Constraints. The constraints are the following:

   (a)

   $$\begin{cases} \text{for every } i \in \{1,\ldots,f\}: \quad t_i > t_{i-1}, \\ \text{if } t_j = \text{finish time of task } r \text{ started at } t_i\text{, then } t_j - t_i = d_r \end{cases} \tag{3}$$

   These constraints mean that the t_i's, together with M, define a feasible schedule of L[2].

[2] Note that there are many more t_i's than equality constraints in general.

(b) The variables e_τ, $\tau \subseteq \{1,2,3\}$, are determined from the t_i's by linear equations[3].

$$\forall \tau \quad e_\tau = \sum_{i \text{ s.t. } \{\text{processors used by } M(i)\}=\{1,2,3\}\setminus\tau} t_{i+1} - t_i. \tag{4}$$

(c) The total lengths of the τ-configurations must fit in the corresponding amount of available time e_τ. For every variable e_τ, we must have one constraint.

$$\forall \tau \quad \sum_{\mathcal{C}_i(\tau)} x_i(\tau) \le e_\tau \tag{5}$$

(d) We must guarantee that we have enough time for the execution of all the tasks of T', *i.e.*, for each $\tau' \subseteq \{1,2,3\}$, we put in the constraint

$$\forall \tau' \quad \sum_{\tau \supseteq \tau'} \sum_{i \text{ s.t. } \tau' \in \mathcal{C}_i(\tau)} x_i(\tau) \ge D^{\tau'}, \tag{6}$$

where $D^{\tau'}$ is the total processing time of all the tasks of T' of type τ'.

The linear program's objective is to minimize t_f.

4. **[Solve the linear programs].** For each relative schedule M, we solve the associated linear program, hence a schedule Schedule(M).
5. **[Deducing a non-preemptive schedule].** We now no longer assume that the tasks of T' are preemptive. To construct a non-preemptive schedule, we proceed as follows: for each relative schedule M, we consider the set of tasks of T' which are preempted in Schedule(M). We modify Schedule(M) by executing these tasks at the end of the schedule in a sequential manner.
6. **[Compare all schedules].** Among all the schedules produced in stage 5, output the one that has the smallest makespan.

5 Example for the non-preemptive case

Consider the task system T given in Table 1. We have $D = 90$. Assume in addition that k (used in step 1 of the algorithm) is equal to 3. Hence, $L = \{1,2,3\}$ and $T' = \{4,5,6,7,8,9\}$. A possible relative schedule of the tasks of L is given by the sequence of snapshots $M = (\{3,1,2\},\{1,2\},\{1\},\emptyset)$ (see Fig. 1 (a)).

We shall now construct the linear program corresponding to M. There are five variables t_0, t_1, t_2, t_3 and t_4. Let $\tau = 1, \tau' = 13$ and $\tau'' = 123$ denote the various sets of processors available at the various times to execute tasks of T'.

Variable e_1 corresponds to the time steps where processors 2 and 3 are processing some tasks of L while processor 1 remains idle or processes some tasks of T'. We define, using similar arguments to those of e_1, variables e_{13} and e_{123}

[3] Thus the variables e_τ are not really necessary - they can be viewed as a convenient notation.

Table 1. The task system

Task name	1	2	3	4	5	6	7	8	9
Processing time	25	24	20	6	5	4	3	2	1
Required processors	2	3	1	2,3	1,3	1,2	3	2	1

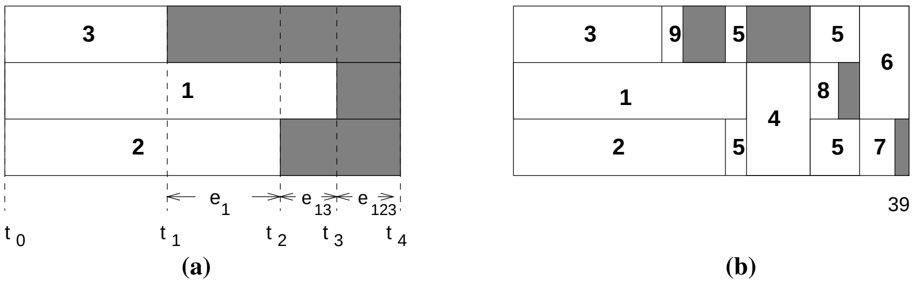

Fig. 1. *(a) A relative schedule of the long tasks (b) A preemptive schedule respecting the relative schedule of (a).*

(the other variables e_{τ_i} are all 0). Considering all possible partitions of τ, τ' and τ'', we have the variables $x_1(\tau)$, $x_{1,3}(\tau')$, $x_{13}(\tau')$, $x_{1,2,3}(\tau'')$, $x_{12,3}(\tau'')$, $x_{1,23}(\tau'')$, $x_{2,13}(\tau'')$, $x_{123}(\tau'')$.

The outline of the linear program is as follows.

minimize t_4 s.t.

$$t_0 = 0, \qquad t_1 - t_0 = 20, \qquad t_2 - t_0 = 24, \qquad t_3 - t_0 = 25$$

$$e_1 = t_2 - t_1 = 4$$
$$e_{13} = t_3 - t_2 = 1$$
$$e_{123} = t_4 - t_3$$

$$x_1(\tau) \le e_1$$
$$x_{1,3}(\tau') + x_{13}(\tau') \le e_{13}$$
$$x_{1,2,3}(\tau'') + x_{12,3}(\tau'') + x_{1,23}(\tau'') + x_{2,13}(\tau'') + x_{123}(\tau'') \le e_{123}$$

$$x_1(\tau) + x_{1,3}(\tau') + x_{1,2,3}(\tau'') + x_{1,23}(\tau'') \ge D^1 = d_9 = 1$$
$$x_{1,3}(\tau') + x_{1,2,3}(\tau'') + x_{12,3}(\tau'') \ge D^3 = d_7 = 3$$
$$x_{1,2,3}(\tau'') + x_{2,13}(\tau'') \ge D^2 = d_8 = 2$$
$$x_{13}(\tau') + x_{2,13}(\tau'') \ge D^{13} = d_5 = 5$$
$$x_{1,23}(\tau'') \ge D^{23} = d_4 = 6$$
$$x_{12,3}(\tau'') \ge D^{12} = d_6 = 4$$

Note that in this particular case the values of the variables t_1, t_2 and t_3 are fixed since tasks 1, 2 and 3 start at the same time as implied by the first snapshot $\{3, 1, 2\}$ of M (this is not true in general).

Figure 1 (b) represents the preemptive schedule given by the linear program in the stage 4 of the algorithm, and Fig.2 (a) gives the corresponding non preemptive schedule produced in stage 5. The optimal schedule is given in Fig.2 (b).

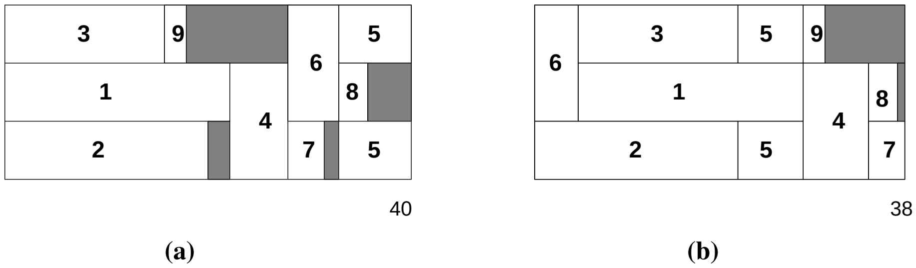

Fig. 2. *(a) A non-preemptive schedule given by the algorithm respecting the relative schedule of Fig.1.a (b) An optimal schedule.*

6 Analysis of the non-preemptive algorithm

6.1 Running time

Lemma 4. *The total number of relative schedules is at most* $(k^3)^{2k}$.

Proof. We create a new snapshot each time there is a new configuration with respect to L, *i.e.* when a task of L starts or ends its execution. Given that $|L| = k$, there are at most $2k$ such events and thus, the sequence contains at most $2k$ snapshots. Each snapshot consists of at most three tasks chosen among k possibilities, so there are at most k^3 possible snapshots. Consequently, there is a constant number, *i.e.* at most $(k^3)^{2k}$, of relative schedules for the tasks of L. □

Lemma 5. *The number of variables of the linear program in stage 3 is* $O(k)$.

Proof. There are as many variables t_i as the number of snapshots of the considered relative schedule M, plus one. Hence, at most $2k + 1$ variables. In addition, there is one variable e_τ for each subset $\tau \subseteq \{1, 2, 3\}$, *i.e.* $O(1)$ variables. Finally, there is one variable of the third type, $x_i(\tau)$, for each τ-configuration, where $\tau \subseteq \{1, 2, 3\}$, hence $O(1)$ variables. The claim follows. □

Lemma 6. *The number of constraints in the linear program of stage 3 is at most* $O(k)$.

Proof. There are at most as many equality constraints of type (3) as long tasks, *i.e.* k. Inequality constraints of type (3) are as many as variables t_i minus one, *i.e.* at most $2k$. There are only $O(1)$ other constraints. □

Proposition 7. *The algorithm presented in Section 4 has running time $O(n)$.*

Proof. A preprocessing phase is needed in order to determine the values of the total processing times of short tasks for each type τ, used in constraints (6). This phase must scan the list of tasks and thus takes $O(n)$ time. Then, it is sufficient to consider for each possible relative schedule the solution of the above linear program, and to choose the relative schedule that gives the schedule of minimum length. Given that the number of possible relative schedules is constant and the solution of the above linear program can be found in $O(1)$ time by the result of Lenstra [19], the result follows. □

6.2 Correctness

First of all notice that given a relative schedule M of the long tasks, the linear program of stage 3 of the algorithm gives the optimal schedule that respects M and where only short tasks are preemptable. Since at stage 3 of the algorithm all the possible relative schedules of the long tasks are considered, the following result is proved.

Lemma 8. *Under the assumption that only short tasks are preemptable, the schedule with the minimum length obtained in stage 4 of the algorithm is optimal.* □

Thus the makespan of the best schedule obtained in stage 4 is a lower bound to the optimal solution of the non-preemptive problem. Now in order to show that the solution provided by the algorithm is close to the optimal schedule, let us assume that the sum D of the execution times of tasks equals 1. This assumption has no effect on the result since the problem is invariant by time scaling. In fact, given any problem instance T, we can multiply all the execution times by a factor of $1/\sum_{i \in T} d_i$ to create a new instance T_s, solve the problem on T_s, and then scale the schedule by $\sum_{i \in T} d_i$ to get a schedule of T.

The main delay is due to stage 5 of the algorithm. We assume that the execution times are sorted: $d_0 \geq d_2 \geq \ldots \geq d_{n-1}$. Recall that L is the set of the k longest tasks. The relative schedule (number of snapshots) has length at most $2k$, hence at most $m(2k) = 6k$ tasks of T' are preempted in the schedule of stage 4 and thus delayed to the end in stage 5. The delay thus incurred is at most $d_k + \ldots + d_{7k-1}$.

Choice of parameter k. we choose k so that $d_k + \ldots + d_{7k-1} < \epsilon$. This is possible while keeping $k = O(c^{1/\epsilon}) = O(1)$ as the following lemma shows.

Lemma 9. *Let $1 \geq d_0 \geq d_1 \geq \ldots \geq d_{n-1} \geq 0$ be a sequence of numbers such that $\sum_i d_i = 1$, and let $\epsilon > 0$. Then there exists $k \leq 7^{1/\epsilon}$ such that $d_k + d_{k+1} + \ldots + d_{7k-1} < \epsilon$.*

Proof. Decompose $d_0+\ldots+d_{n-1}$ into blocks: $B_0 = d_0+\ldots+d_6$, $B_1 = d_7+\ldots+d_{7^2-1}$, ..., $B_i = d_{7^i}+\ldots+d_{7^{i+1}-1}$. Since the sum of all blocks equals 1, at most $1/\epsilon$ of them are greater than ϵ. Take the first block B_i which is less than ϵ. We have: $i \leq 1/\epsilon$ and for $k = 7^i$, $B_i = d_k + \ldots + d_{7k-1} < \epsilon$, hence the lemma. □

Remark 10. If $n < 7^{1/\epsilon}$ then all the tasks may go into L and the algorithm finds the optimal solution in constant time.

All we have to do now is apply the algorithm setting $|L| = k \leq 7^{1/\epsilon}$, and we obtain a schedule of length at most $(1+\epsilon)$ times the optimal length.

Theorem 11. *There is an algorithm A which given a set T of n independent multiprocessor tasks, a fixed number m of dedicated processors and a constant $\epsilon > 0$, produces, in time at most $O(n)$, a schedule of T whose makespan is at most $(1+\epsilon)C_{\max}^{opt}$.*

7 Remarks

In this paper we proposed an optimal algorithm for scheduling n preemptable dedicated independent multiprocessor tasks on a fixed number of processors, and a polynomial time approximation scheme for the same problem when the tasks are non-preemptable. The approximation algorithm given is polynomial in the size of the problem but super-exponential in $1/\epsilon$. This result is the best performance guarantee one might hope for since no fully polynomial approximation scheme can be expected as the problem is strongly NP-hard [8]. Of course, it remains interesting to see whether there exists polynomial approximation schemes with lower time complexity in the number of processors m.

Note that we conjecture that our algorithm can be extended for the parallel variant of the model (non-dedicated case) as well (see [3] for related work).

Some natural questions arise. Do the linear programming ideas developed in this paper have applications to other variants of scheduling?

References

1. L. Bianco, J. Blazewicz, P. Dell'Olmo, and M. Drozdowski. *Scheduling Preemptive Multiprocessor Tasks on Dedicated Processors.* Performance Evaluation, **20**:361-371, (1994)
2. J. Blazewicz, P. Dell'Olmo, M. Drozdowski, and M.G. Speranza. *Scheduling Multiprocessor Tasks on Three Dedicated Processors.* Information Processing Letters, **41**:275-280, (1992)
3. J. Blazewicz, M. Drabowski, and J. Węglarz. *Scheduling multiprocessor tasks to minimize schedule length.* IEEE Transactions on Computing, **35**(5):389-393, (1986)
4. E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. Lapaugh. *Scheduling file transfers.* SIAM J. Compt., **14**(3):744-780, (1985)
5. P. Dell'Olmo, M. G. Speranza, and Z. Tuza. *Efficiency and Effectiveness of Normal Schedules on Three Dedicated Processors.* Submitted.

6. M. Drozdowski. *Scheduling Multiprocessor Tasks - An Overview.* Private communication, (1996)
7. J. Du, and J. Y-T. Leung. *Complexity of Scheduling Parallel Task Systems.* SIAM J. Discrete Math., **2**(4):473-487, (1989)
8. M. Garey, D. Johnson. *Computers and Intractability, A Guide to the theory of NP-completeness.* W. H. Freemann and Company, San Francisco, CA, (1979)
9. M. X. Goemans. *An Approximation Algorithm for Scheduling on Three Dedicated Processors.* Disc. App. Math., **61**:49-59, (1995)
10. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, MA, (1990)
11. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnoy Kan. *Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey.* Ann. Disc. Math., **5**:287-326, (1979)
12. L.A. Hall and D.B. Shmoys. *Jackson's rule for single-machine scheduling: making a good heuristic better.* Mathematics of Operations Research, **17**:22-35, (1992)
13. J. Hoogeveen, S. L. van de Velde, and B. Veltman. *Complexity of Scheduling Multiprocessor Tasks with Prespecified Processors Allocation.* Disc. App. Math., **55**:259-272, (1994)
14. C. Kenyon, E. Rémila. *Approximate Strip-Packing.* Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS), 31-36, (1996)
15. D. E. Knuth. *The Art of Computer Programming,* Vol.1: *Fundamental Algorithms.* Addison-Wesley, Reading, MA, (1968)
16. A. Krämer. *Scheduling Multiprocessor Tasks on Dedicated Processors.* PhD Thesis, Fachbereich Mathematik/Informatik, Universität Osnabrück, (1995)
17. H. Krawczyk, and M. Kubale. *An approximation algorithm for diagnostic test scheduling in multicomputer systems.* IEEE Trans. Comput., **C-34**(9):869-872, (1985)
18. M. Kubale. *Preemptive Scheduling of Two-Processor Tasks on Dedicated Processors (in polish).* Zeszyty Naukowe Politechniki Śląskiej, Seria: Automatyka z.100, **1082**:145-153, (1990)
19. H.W. Lenstra *Integer Programming with a Fixed Number of Variables.* Math. Oper. Res., **8**:538-548, (1983)
20. B. Veltman, B. J. Lageweg, and J. K. Lenstra. *Multiprocessor Scheduling with Communication Delays.* Parallel Computing, **16**:173-182, (1990)