Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors*

L. Bianco[a], J. Błażewicz[b], P. Dell'Olmo[a] and M. Drozdowski[b]
[a]*Istituto di Analisi dei Sistemi ed Informatica del CNR, Rome, Italy*
[b]*Instytut Informatyki Politechniki Poznańskiej, Poznań, Poland*

In the classical scheduling theory, it is widely assumed that a task can be processed by only one processor at a time. With the rapid development of technology, this assumption is no longer valid. In this work we present a problem of scheduling tasks, each of which requires for its processing a set of processors simultaneously and which can be executed on several alternative sets of processors. Scheduling algorithms based on dynamic and linear programming are presented that construct minimum length non-preemptive and preemptive schedules, respectively. Results of computational experiments are also reported.

1. Introduction

One of the commonly imposed assumptions in the classical scheduling theory is that any task is processed by one processor at a time, see e.g. [2,9,12]. With the development of technology (parallel systems), this assumption is not so obvious. As examples, a fault tolerant system in which several processors test each other [17] or a testing system in which one processor simulates the tested object and the other processor is analyzing its output can be considered. Another range of applications appears in the field of new parallel algorithms and corresponding task systems [1,15,22]. Thus, it seems reasonable to reconsider the assumptions and propose new models.

In recent years, several papers dealt with the problem in which a task requires more than one processor simultaneously. Two groups of models have been distinguished [21]. In the first group of models, it is assumed that any task can be executed on any set of processors under the condition that a fixed *number* of processors is assigned to the task [7,8,11,13,20]. There are three models in this group, namely, model "$size_j$" where a task requires a fixed number of processor simultaneously [7,8], model "$cube_j$" where a task requires a number of processors which is a power of 2 (e.g. either 1 or 2 or 4, etc. processors) [11,12], and model "*any*" for which each task can be executed on any subset of processors but the execution speed depends on the number of processors processing the task [13,23].

*This research was partially supported by a KBN grant and by project CRIT.

In the second group of models, it is assumed that the *set* of processors processing a task is important [4,6,18]. This problem is similar to classical scheduling with additional resources [10] and can be expressed in terms of weighted graph coloring [18]. Two models in this group have been distinguished. These are: model "fix_j" where a task can be executed by a fixed set of processors [4,6,18], and model "set_j" in which each task has a set of alternative sets of processors by which it can be processed.

In this paper we will concentrate on model "set_j", which is a generalization of model "fix_j". Before presenting results, we will set up the problem more formally.

Let us denote by $\mathcal{T}$ a set of n tasks and by $\mathcal{P}$ a set of m dedicated processors. Each task requires for its processing some set D of processors simultaneously. Moreover, for each task there can be more than one such set, i.e. the task can be executed on alternative sets of processors $D_1,\ldots,D_{s_j}$. Let us denote by S_j the family of sets of processors which can process task T_j (i.e. $S_j = \bigcup_{l=1}^{|s_j|}\{D_l\}$). We will name each of such subsets D_l a *processing mode* or a *processing configuration* of task T_j.

The processing time of a task depends on the set of processors processing it. We assume that processing times of tasks are given in the matrix:

$$X = \{t_j^{D_i} : t_j^{D_i} \in R^+ \text{ is a processing time of task } T_j \text{ in processing mode } i \text{ requiring a set of processors } D_i\text{; if } T_j \text{ cannot be scheduled in this mode, then } t_j^{D_i} = +\infty\}.$$

Tasks are independent. We will analyze both *preemptable* and *nonpreemptable* task cases. In the case of preemptable tasks, any task can be interrupted at no cost and restarted later, probably in a different processing mode. In this case, we also assume that processing percentages of tasks processed in various processing modes are "additive" or, in other words, can be accumulated. For example, if some task has been processed 1 second in processing mode A while the total processing time for this task in this mode is 10 seconds, then the task is processed in 10%. If next this task has been processed in additional 20% in some other processing mode, then it is processed in 30%. After restarting in the processing mode A, this task will occupy processors appropriate in this mode in 7 additional seconds. This approach is similar to the case of scheduling on unrelated machines or scheduling under resource requirements [5] and differs in considering alternative processing modes for tasks.

The optimality criterion is schedule length ($C_{\max}$).

In order to denote analyzed problems, we will use an extended version of the scheme proposed by Graham et al. [14] with later extensions [10,21]. In this notation, a scheduling problem is described by three fields (separated by the symbol "|"). The first field describes the processor system. In this work it will be the letter P, optionally followed by a positive integer which denotes the number of processors. If there is no constant after P, then the number of processors is not fixed and is given in the current instance of the problem. The second field describes the task system. The word "*pmtn*" is used to denote that tasks are preemptable; if this word is absent, tasks are

nonpreemptable. The word "set_j" denotes simultaneous requirement of multiple processors by tasks. Moreover, in general, any task can be processed by more than one such set of processors. The last field denotes the optimality criterion; here it is $C_{\max}$.

In this paper, we will present a dynamic programming based procedure to solve optimally simple cases of the nonpreemptive version of the problem. This will result in pseudo-polynomial algorithms. For a general case of the nonpreemptive scheduling, a heuristic will be proposed and its worst case behavior will be analyzed. The preemptive version of the problem will be solved via linear programming. The organization of the paper is as follows. In section 2, the case of nonpreemptive scheduling is considered. In section 3, the preemptive version of the problem is analyzed. Section 4 summarizes results of computational experiments.

2. Nonpreemptive scheduling

The problem of nonpreemptive scheduling arises in many practical applications. It can be the case in manufacturing systems where the change-over costs are usually too high to allow for swapping partially processed tasks on the machines. In computer systems, two users cannot access a printer simultaneously or modify at the same moment of time the same record in a database. Hence, nonpreemptive schedules seem quite natural.

Here, problem $P|set_j|C_{max}$ is NP-hard in general. This can be easily shown by a reduction from the set partition problem to problem $P2|set_j|C_{\max}$. For three processors and tasks requiring processors from only one set (i.e. when $|S_j| = 1$ for $j = 1,\ldots,n$), the problem is NP-hard in the strong sense [6]. Thus, it is unlikely to propose an algorithm solving these problems in polynomial time. Moreover, for more than two processors, it is difficult to expect a pseudo-polynomial time algorithm in a general case.

In this section, we will present pseudo-polynomial time algorithms for problems $P2|set_j|C_{\max}$ and a restricted version of the problem $P3|set_j|C_{\max}$. Then, a simple heuristic for the problem $P|set_j|C_{\max}$ with the worst-case behavior bound equal to m will be presented.

2.1. $P2|set_j|C_{\max}$

Now, we will given an optimization pseudo-polynomial time algorithm based on a dynamic programming procedure. We assume that processing times of tasks are positive integers. This assumption does not cause significant loss of generality because any real number can be approximated by a rational number; what is more, computer representation of numbers has only limited precision.

In order to find the optimal solution, let us calculate the function checking the existence of a feasible schedule of length i for the set of tasks $T_1,\ldots,T_j$. This function can be defined more formally as follows:

$f(i, j, x, y) = \{1$ if and only if the set of tasks $T_1, \ldots, T_j$ can be feasibly executed starting at moment 0 and finishing exactly at moment i while processors P_1, P_2 are utilized x and y units of time, respectively; otherwise, $f(i, j, x, y) = 0\}$,

$$j = 1, \ldots n,\ i = 1, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}),\ x, y = 0, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}).$$

Assuming that all the tasks are executed in their shortest processing time mode, sequentially and in any order, we would have a feasible schedule of length $\sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}(t_k^{D_l})$. Thus, this value is an upper bound of the schedule length in the above definition. Note that the smallest i for which the function $f(i, n, x, y) = 1$ is also $C^*_{\max}$, the optimal length of the schedule.

Before giving formulae for calculating $f(i, j, x, y)$, let us first analyze the cases when it takes value 1. For $j = 1$, i.e. for the first task only, $f(i, 1, x, y)$ takes value 1 only for such i that execution of task T_1 can be feasibly finished in moment i (starting at moment 0). This means that i must be equal to the processing time of T_1 in one of its modes, namely, $i = t_1^1$ or $i = t_1^2$ or $i = t_1^{12}$.

Now, consider the case when $j > 1$. Suppose we have already calculated the function $f(i, j', x, y)$ for entries such that $j' = 1, \ldots, j-1$. For $j > 1$, the value of $f(i, j, x, y)$ is 1 when one of the following cases occurs:

(1) task T_j is scheduled in the duo-processor mode and the schedule finishes at time i,

(2) task T_j is scheduled on one of the processors in one of the two uni-processor modes while the schedule finishes at time i.

In the latter case, we have to distinguish three further sub-cases:

(a) The schedule finishes at moment i, but task T_j does not change the length of the schedule imposed by tasks $T_1, \ldots, T_{j-1}$ (cf. figures 1(a)).

(b) The schedule finishes at moment i, and i is also the moment when the execution of task T_j stops. For tasks $T_1, \ldots, T_{j-1}$, however, the length of the schedule was imposed by a different processor than the one to which T_j is assigned (figure 1(b)).

(c) The schedule finishes at moment i, and this is also the moment when the execution of task T_j finishes, while for tasks $T_1, \ldots, T_{j-1}$ the length of the schedule was imposed by the processor to which T_j is assigned (figure 1(c)).

According to the above discussion, there are seven possible situations when $f(i, j, x, y) = 1$ (one for a duo-processor mode and six (3×2) for processing a task in the uni-processor mode on each of the two processors). The last thing to be

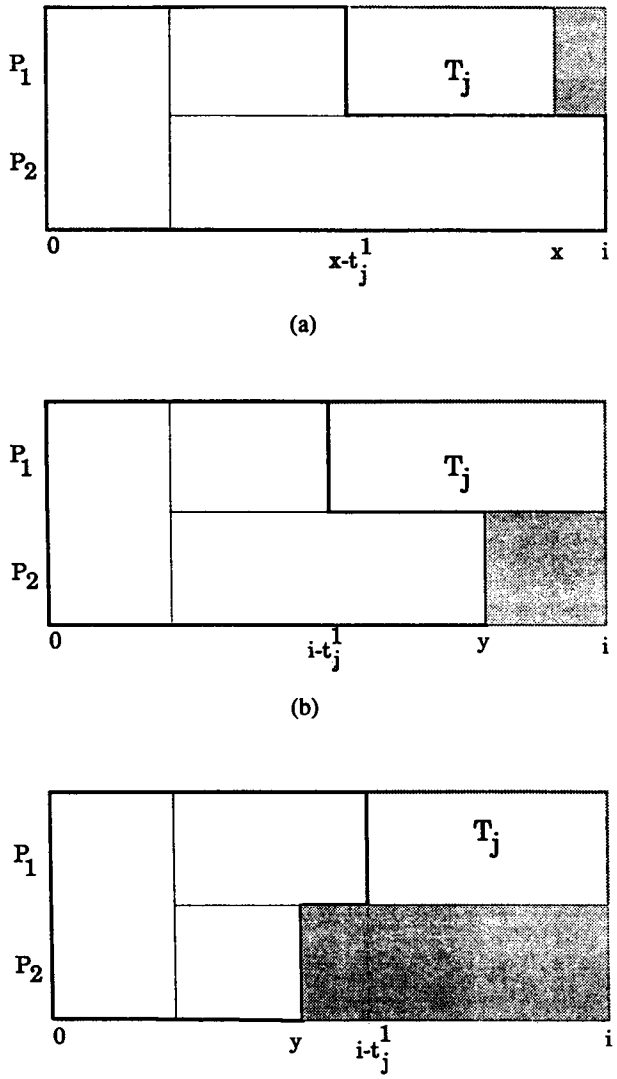


Figure 1. Three possible cases of scheduling uni-processor tasks on one of two processors.

considered before presenting formulae are mutual dependencies between i, x and y. We will discuss them in the order of the previously enumerated cases.

(1) Since it is impossible for a processor to work longer than i in the schedule of length i, thus $x \leq i$ and $y \leq i$.

(2a) Since the schedule finishes at moment i on one processor, say P_1 (T_j is scheduled on P_2), and T_j is finished before i, so $y \le i$. Similarly, $x \le i$ when T_j is scheduled on P_1 and the length of the schedule i is imposed on P_2.

(2b) Suppose T_j is scheduled on P_1. In this case, $i - t_j^1 \le y \le i$ because only for such values of y adding T_j on P_1 will increase the length of the schedule, which for tasks $T_1,\dots,T_{j-1}$ has been y. Analogously, $i - t_j^2 \le x \le i$, when T_j is scheduled on P_2.

(2c) Suppose T_j is scheduled on P_1. In this case, $y < i - t_j^1$ because only for y satisfying this inequality the length of the schedule for tasks $T_1,\dots,T_{j-1}$ was imposed on processor P_1. Similarly, $x < i - t_j^2$ when T_j was scheduled on P_2.

Now, we can give exact formulae for the calculation of $f(i, j, x, y)$. The function $f(i, j, x, y)$ can be calculated iteratively using the following equations:

For $j = 1$:

$f(i, 1, i, 0) = 1$ when $i = t_1^1$,

$f(i, 1, 0, i) = 1$ when $i = t_1^2$,

$f(i, 1, i, i) = 1$ when $i = t_1^{12}$,

in all other cases $f(i, 1, x, y) = 0$,

$$i = 1,\dots,\sum_{k=1}^{n} \min_{l=1,\dots,|S_k|}(t_k^{D_l}),\quad x, y = 0,\dots,\sum_{k=1}^{n} \min_{l=1,\dots,|S_k|}(t_k^{D_l}).$$

For $j > 1$, $f(i, j, x, y) = 1$, if one of the following cases occurs:

$f(i - t_j^{12}, j-1, x - t_j^{12}, y - t_j^{12}) = 1$	$x \le i, y \le i,$	case 1,
$f(i, j-1, x - t_j^1, i) = 1$	$x \le i,$	case 2a, T_j is scheduled on P_1,
$f(y, j-1, i - t_j^1, y) = 1$	$i - t_j^1 \le y \le t_j^1,$	case 2b, T_j is scheduled on P_1,
$f(i - t_j^1, j-1, i - t_j^1, y) = 1$	$y < i - t_j^1,$	case 2c, T_j is scheduled on P_1,
$f(i, j-1, i, y - t_j^2) = 1$	$y \le i,$	case 2a, T_j is scheduled on P_2,
$f(x, j-1, x, i - t_j^2) = 1$	$i - t_j^2 \le x \le t_j^2,$	case 2b, T_j is scheduled on P_2,
$f(i - t_j^2, j-1, x, i - t_j^2) = 1$	$x < i - t_j^2,$	case 2c, T_j is scheduled on P_2,

$$i = 1,\dots,\sum_{k=1}^{n} \min_{l=1,\dots,|S_k|}(t_k^{D_l}),\quad j = 2,\dots,n,$$

in all other case $f(i, j, x, y) = 0$,

$$i = 1,\dots,\sum_{k=1}^{n} \min_{l=1,\dots,|S_k|}(t_k^{D_l}),\quad j = 2,\dots,n,\quad x, y = 0,\dots,\sum_{k=1}^{n} \min_{l=1,\dots,|S_k|}(t_k^{D_l}).$$

The optimal length of the schedule, $C^*_{\max}$, is the minimal k such that $f(k, n, x, y) = 1$.

The optimal schedule can be found by backtracking from $f(C^*_{\max}, n, x, y)$ through the non-zero entries in the table of the function $f(i, j, x, y)$ until some non-zero entry for $j = 1$. One backtracking step could look as follows. For some entry $f(i, j, x, y) \neq 0$, one of the above seven cases occurred for one of the processing modes requiring set D_l of processors. When T_j has $f(i, j, x, y) = 1$ due to cases 2a, 2b, 2c, then we place T_j in uni-processor mode on an appropriate processor before the previously allocated uni-processor tasks. If there are no such tasks, we assume that T_j is the last task on the given processor. For case 1 (i.e. $D_l = \{P_1, P_2\}$), T_j can be processed in the beginning of the schedule. Next, we step back to the entry $f(i', j-1, x', y') = 1$ which caused $f(i, j, x, y)$ to be 1. In this way, we can find iteratively the order of tasks and their processing modes. Finally, we shift all the tasks to the left (in the direction of moment 0) as far as possible.

Now, let us assess the complexity of this approach. Note that the most time-consuming activity is the calculation of all the values of the function $f(i, j, x, y)$. Since i takes all values in the range $1,\ldots,\sum_{k=1}^{n}\min_{l=1,\ldots,|S_k|}(t_k^{D_l})$, j in the range $1,\ldots,n$, and x, y in the range $1,\ldots,\sum_{k=1}^{n}\min_{l=1,\ldots,|S_k|}(t_k^{D_l})$, it is easy to observe that $f(i, j, x, y)$ can be calculated in pseudo-polynomial time $O(n(\sum_{j=1}^{n}\min_{l=1,\ldots,|S_j|}\{t_j^{D_l}\})^3)$. We conclude this section with an example.

EXAMPLE 1

We are given $\mathcal{T} = \{T_1, T_2, T_3\}$. $S_1 = \{\{P_1\}, \{P_2\}\}$, $S_2 = \{\{P_1, P_2\}\}$, $S_3 = \{\{P_1\}, \{P_2\}\}$, $t_1^1 = 5$, $t_1^2 = 6$, $t_2^{12} = 7$, $t_3^1 = 5$, $t_3^2 = 9$.

The values of $f(i, j, x, y)$ and the process of backtracking are depicted in table 1. Let us take a closer look at the backtracking process. We start with the entry $i = 13$, $j = 3$, $x = 12$, $y = 13$. It is non-zero because $f(13, 2, 7, 13) = 1$ and T_3 can be scheduled according to case 2a. T_3 will be executed as the last on P_1. We step back to the entry $i = 13$, $j = 2$, $x = 7$, $y = 13$; it is non-zero due to $f(6, 1, 0, 6)$ and T_2 processed as the first duo-processor task on P_1 and P_2. Finally, T_1 is processed on P_2 as the last uni-processor task. The optimal schedule is presented in figure 2.

2.2. $P3|set_j|C_{\max}$

The problem of nonpreemptive scheduling of tasks requiring a set of processors simultaneously in the case of three dedicated processors is NP-hard in the strong sense [6]. However, if we relax the problem in the way that for all the tasks only two types of duo-processor processing modes are allowed, then a dynamic programming algorithm can be proposed. Assuming the above constraint, we will analyze the problem along the same lines as in the previous subsection.

Without loss of generality, let us assume that the processing mode requiring processors $\{P_1, P_3\}$ is not present (or is forbidden). Of course, this can be any other

Table 1

Example 1. Table of the function $f(i, j, x, y)$.

j \ i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1					x=5 y=0	x=0 y=6											
2												x=12 y=7	x=7 y=13				
3													x=12 y=13			x=12 y=16	x=17 y=7

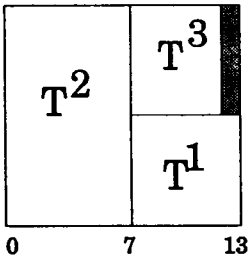


Figure 2. Example 1. The optimal schedule.

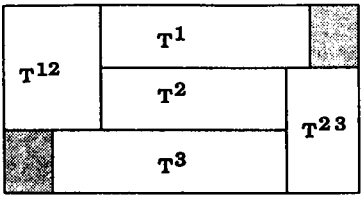


Figure 3. Typical form of the schedule for a restricted version of the problem P3$|set_j|C_{max}$.

pair of processors if we take into account appropriate renumbering of processors. Without any additional loss of generality, we assume that all processing times are positive integers. In the problem limited in this way, any feasible schedule always has the same structure (cf. figure 3), or can be transformed to a schedule with such a structure without changing C_{max}. Uni-processor tasks are scheduled in parallel, tasks requiring $\{P_1, P_2\}$ are scheduled before the uni-processor tasks (or after), tasks requiring $\{P_2, P_3\}$ are scheduled after (or before) the uni-processor tasks, and finally, tasks requiring $\{P_1, P_2, P_3\}$ are scheduled at the end (or at the beginning) of the schedule. The length of the schedule is equal to the maximal time a single processor is used.

In order to find the optimal solution, let us calculate the function checking the existence of a feasible schedule for tasks $T_1,\ldots,T_j$ in i units of time (starting at 0). In other words, we are going to calculate the function:

$f(i, j, x, y, z) = \{1$ if and only if the set of tasks $T_1,\ldots,T_j$ can be feasibly executed finishing exactly at moment i while processors P_1, P_2, P_3 are used x, y, and z units of time, respectively; otherwise the value of the function is $0\}$;

$$i = 1,\ldots,\sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}(t_k^{D_l});\quad j = 1,\ldots,n,\quad x,y,z = 0,\ldots,\sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}(t_k^{D_l}).$$

As has been observed in the previous subsection, $\sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}(t_k^{D_l})$ is an upper bound of the schedule length.

Before going into detailed description of how $f(i, j, x, y, z)$ can be calculated, let us first analyze when it takes value 1. Each of the tasks can be executed in one of the following ways.

(1) A task is scheduled in the triple-processor mode (i.e. requires processors P_1, P_2, P_3 simultaneously).

(2) A task is scheduled in one of the two allowed duo-processor modes (i.e. requires simultaneously either processors P_1, P_2 or P_2, P_3).

(3) A task is scheduled in the uni-processor mode (i.e. this task requires only one of the three processors).

The last two cases require further subdivision into three sub-cases. Suppose the task T_j is considered.

(a) T_j is scheduled in such a way that the length of the schedule for the tasks $T_1,\ldots,T_j$ is equal to the length of the schedule for the tasks $T_1,\ldots,T_{j-1}$ (cf. figure 4(a)).

(b) T_j is scheduled in such a way that the length of the schedule for the tasks $T_1,\ldots,T_j$ is greater than the length of the schedule for the tasks $T_j,\ldots,T_{j-1}$ due to assigning T_j in a particular mode to some processor(s). Here, however, processor(s) which have been assigned to T_j were not the most loaded for tasks $T_1,\ldots,T_{j-1}$ (cf. figure 4(b)).

(c) The length of the schedule is imposed by the processor(s) processing T_j both for $T_1,\ldots,T_j$ and for $T_1,\ldots,T_{j-1}$ (cf. figure 4(c)).

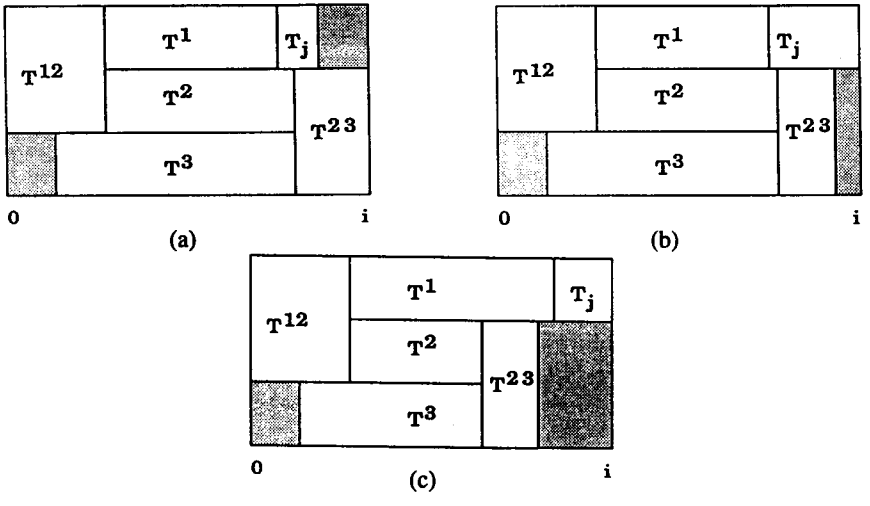


Figure 4. Three possible cases of scheduling uni-processor tasks on one of three processors.

Note that the above three cases apply to uni- and duo-processor modes. Next, we are going to discuss the dependencies between i, x, y and z in each of the cases.

(1) T_j is scheduled in the triple-processor mode. Since no processor can work longer than i in a schedule of length i, it is enough to check only x, y, z satisfying $x \le i, y \le i, z \le i$.

(2a) Suppose T_j is just about to be scheduled on processors P_1, P_2. The length of the schedule is imposed by P_3 (the processor which is not used by T_j). Hence, $x \le i$ and $y \le i$. When T_j is scheduled on P_2, P_3, then $y \le i$ and $z \le i$, analogously.

(2b) Suppose T_j is to be scheduled on processors P_1, P_2 so that the schedule finishes at i. Then $i - t_j^{12} \le z < i$ because only for such z may this case take place. Then, the length of the schedule for tasks $T_1,\ldots,T_j$ can be imposed by P_1 or by P_2. In the former case, $i = x, y \le i$, in the latter case $i = y, x \le i$. By analogy, when T_j is to be scheduled on P_2, P_3, then $i - t_j^{23} \le x < i$ and either $y = i, z \le i$ or $z = i, y \le i$.

(2c) Consider T_j in the processing mode requiring P_1, P_2. Then $z < i - t_j^{12}$ (it is a case of type c). As in case 2b, P_1 or P_2 is the most loaded processor. Then $x = i, y \le i$ or $y = i, x \le i$, respectively. When T_j is to be scheduled in the processing mode requiring P_2, P_3, then $x < i - t_j^{23}$, and by analogy, either $y = i, z \le i$ or $z = i, y \le i$.

(3a) Consider T_j in the mode requiring P_1. Since the length of the schedule is not imposed by P_1, thus $x < i$. Moreover, the length of the schedule is imposed on P_2 or on P_3. In the former case $i = y, z \le i$, in the latter $z = i, y \le i$. By analogy, $y < i$ and either $i = x, z \le i$ or $i = z, x \le i$, when T_j is assigned to P_2, and $z < i$ and either $i = x, y \le i$ or $i = y, x \le i$, when T_j is assigned to P_3.

(3b) Suppose T_j requires P_1. For the tasks $T_1,\ldots,T_{j-1}$ the length of the schedule was equal to the load of P_2 or P_3. In the first case $i - t_j^1 \le y < i, z \le y$, in the second case $i - t_j^1 \le z < i, y \le z$. For assigning T_j to P_2 and P_3 we have, respectively, (P_2) $i - t_j^2 \le x < i, z \le x$ or $i - t_j^2 \le z < i, x \le z$, and (P_3) $i - t_j^3 \le x < i, y \le x$ or $i - t_j^3 \le y < i, x \le y$.

(3c) Suppose T_j requires P_1. Since for $T_1,\ldots,T_{j-1}$ the length of the schedule was equal to the load of P_1, then $y \le i - t_j^1$, $z \le i - t_j^1$. Analogously, $x \le i - t_j^2$, $z \le i - t_j^2$ when T_j requires P_2 and $x \le i - t_j^3, y \le i - t_j^3$ when T_j requires P_3.

Now, we are ready to present the formulae to calculate $f(i, j, x, y, z)$.

The values of the function $f(i, j, x, y, z)$ can be easily calculated iteratively, using the following formulae.

For $j = 1$:

$$f(i, 1, i, 0, 0) = 1 \text{ when } i = t_1^1,$$

$$f(i, 1, 0, i, 0) = 1 \text{ when } i = t_1^2,$$

$f(i, 1, 0, 0, i) = 1$ when $i = t_1^3$,
$f(i, 1, i, i, 0) = 1$ when $i = t_1^{12}$,
$f(i, 1, 0, i, i) = 1$ when $i = t_1^{23}$,
$f(i, 1, i, i, i) = 1$ when $i = t_1^{123}$,
for all other cases $f(i, 1, x, y, z) = 0$;

$$i = 1, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}), \quad x, y, z = 0, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}).$$

For $j > 1$, $f(i, j, x, y, z) = 1$ when one of the following cases occurs:

T_j is scheduled on P_1, P_2, P_3

$f(i - t_j^{123}, j - 1, x - t_j^{123}, y - t_j^{123}, z - t_j^{123}) = 1 \quad x, y, z \le i,$ case 1,

T_j is scheduled on P_1, P_2

$f(i, j - 1, x - t_j^{12}, y - t_j^{12}, i) = 1 \quad x, y \le i,$ case 2a,
$f(z, j - 1, i - t_j^{12}, y - t_j^{12}, z) = 1 \quad i - t_j^{12} \le z < i, y \le i,$ case 2b,
$f(z, j - 1, x - t_j^{12}, i - t_j^{12}, z) = 1 \quad i - t_j^{12} \le z < i, x \le i,$ case 2b,
$f(i - t_j^{12}, j - 1, i - t_j^{12}, y - t_j^{12}, z) = 1 \quad z < i - t_j^{12}, y \le i,$ case 2c,
$f(i - t_j^{12}, j - 1, x - t_j^{12}, i - t_j^{12}, z) = 1 \quad z < i - t_j^{12}, x \le i,$ case 2c,

T_j is scheduled on P_2, P_3

$f(i, j - 1, i, y - t_j^{23}, z - t_j^{23}) = 1 \quad y, z \le i,$ case 2a,
$f(x, j - 1, x, i - t_j^{23}, z - t_j^{23}) = 1 \quad i - t_j^{23} \le x < i, z \le i,$ case 2b,
$f(x, j - 1, x, y - t_j^{23}, i - t_j^{23}) = 1 \quad i - t_j^{23} \le x < i, y \le i,$ case 2b,
$f(i - t_j^{23}, j - 1, x, i - t_j^{23}, z - t_j^{23}) = 1 \quad x < i - t_j^{23}, z \le i,$ case 2c,
$f(i - t_j^{23}, j - 1, x, y - t_j^{23}, i - t_j^{23}) \quad x < i - t_j^{23}, y \le i,$ case 2c,

T_j is scheduled on P_1

$f(i, j - 1, x - t_j^1, i, z) = 1 \quad x \le i, z \le i,$ case 3a,
$f(i, j - 1, x - t_j^1, y, i) = 1 \quad x \le i, y \le i,$ case 3a,
$f(y, j - 1, i - t_j^1, y, z) = 1 \quad i - t_j^1 \le y \le i, z \le y,$ case 3b,
$f(z, j - 1, i - t_j^1, y, z) = 1 \quad i - t_j^1 \le z \le i, y \le z,$ case 3b,
$f(i - t_j^1, j - 1, i - t_j^1, y, z) = 1 \quad z, y < i - t_j^1,$ case 3c,

T_j is scheduled on P_2

$f(i, j-1, i, y-t_j^2, z) = 1$	$y \le i, z \le i,$	case 3a,
$f(i, j-1, x, y-t_j^2, i) = 1$	$y \le i, x \le i,$	case 3a,
$f(x, j-1, x, i-t_j^2, z) = 1$	$i - t_j^2 \le x < i, z \le x,$	case 3b,
$f(z, j-1, x, i-t_j^2, z) = 1$	$i - t_j^2 \le z < i, x \le z,$	case 3b,
$f(i-t_j^2, j-1, x, i-t_j^2, z) = 1$	$x, z < i - t_j^2,$	case 3c,

T_j is scheduled on P_3

$f(i, j-1, i, y, z-t_j^3) = 1$	$z \le i, y \le i,$	case 3a,
$f(i, j-1, x, i, z-t_j^3) = 1$	$z \le i, x \le i,$	case 3a,
$f(x, j-1, x, y, i-t_j^3) = 1$	$i - t_j^3 \le x < i, y \le x,$	case 3b,
$f(y, j-1, x, y, i-t_j^3) = 1$	$i - t_j^3 \le y < i, x \le y,$	case 3b,
$f(i-t_j^3, j-1, x, y, i-t_j^3) = 1$	$x, y < i - t_j^3,$	case 3c,

$$i = 1, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}), \quad j = 1, \ldots, n.$$

If none of the above 26 cases occurs, then $f(i, j, x, y, z) = 0$ for

$$i = 1, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}), \quad j = 1, \ldots, n, \quad x, y, z = 0, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|} (t_k^{D_l}).$$

The optimal length of the schedule is equal to

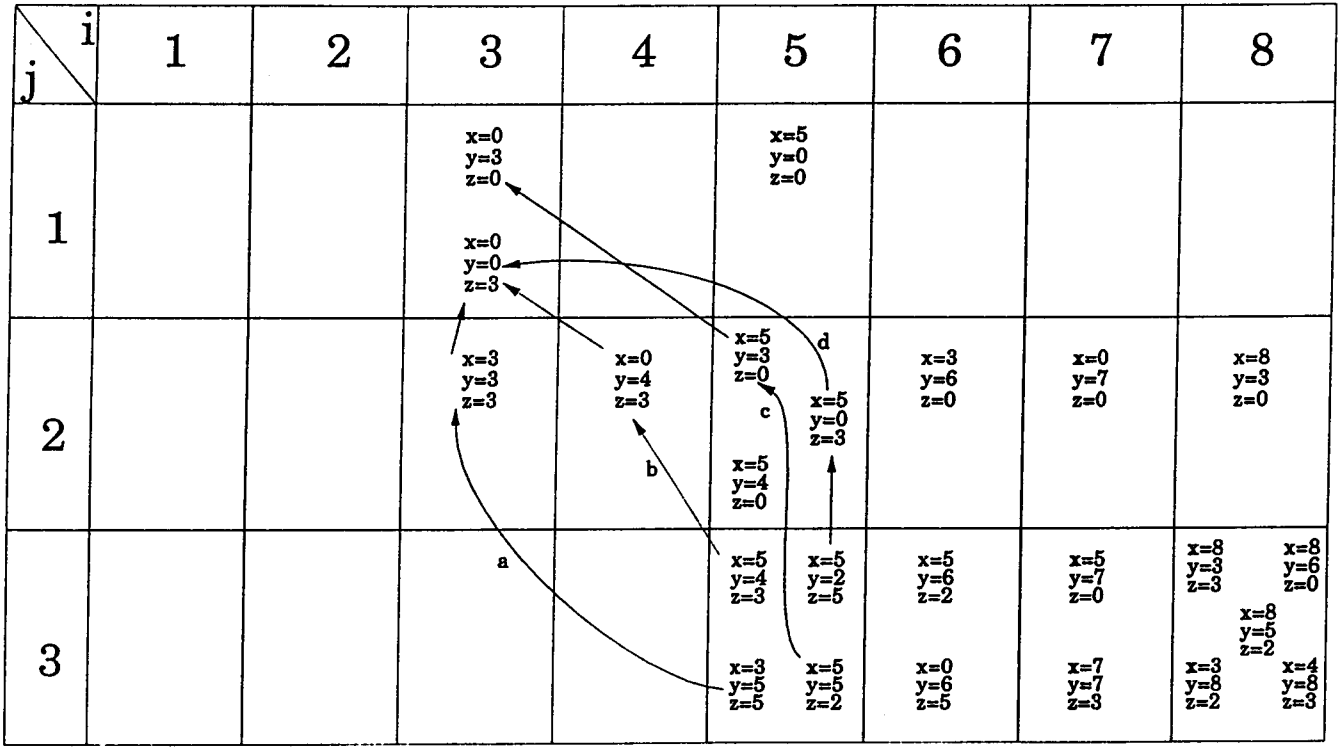
$$C^*_{\max} = \min_{x,y,z} \{i : f(i, n, x, y, z) = 1\}.$$

The optimal schedule can be found by a similar backtracking procedure as in the dynamic programming algorithm for the problem $P2|set_j|C_{\max}$. The only difference here is the fact that tasks processed on P_1, P_2 are executed first, then uni-processor tasks are executed, finally tasks assigned to P_2, P_3 follow. Tasks requiring P_1, P_2, P_3 can be executed at the beginning or at the end of the schedule.

Again, the complexity of the algorithm depends mainly on the complexity of calculation of all the values of the function $f(i, j, x, y, z)$. Since i takes all the values in the range $1, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}(t_k^{D_l})$, j in the range $1, \ldots, n$, and x, y, z in the range $0, \ldots, \sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}(t_k^{D_l})$, it is easy to observe that the complexity of this procedure is $O(n(\sum_{k=1}^{n} \min_{l=1,\ldots,|S_k|}\{t_k^{D_l}\})^4)$ time. We conclude this subsection with an example.

Table 2

Example 2. Table of the function $f(i, j, x, y, z)$.



EXAMPLE 2

Suppose we have $\mathcal{T} = \{T_1, T_2, T_3\}$. $S_1 = \{\{P_1\}, \{P_2\}, \{P_3\}\}$, $S_2 = \{\{P_1\}, \{P_2\}, \{P_1, P_2\}\}$, $S_3 = \{\{P_1\}, \{P_1, P_2\}, \{P_2, P_3\}\}$, $t_1^1 = 5$, $t_1^2 = 3$, $t_1^3 = 3$, $t_2^1 = 5$, $t_2^2 = 4$, $t_2^{12} = 3$, $t_3^1 = 5$, $t_3^{12} = 4$, $t_3^{23} = 2$.

The upper bound of the schedule length is equal to $t_1^2 + t_2^{12} + t_3^{23} = 8$. The values of $f(i, j, y, z)$ and the process of backtracking are depicted in table 2. As can be seen, there are more than only one possible optimal schedules. All the optimal schedules are presented in figure 5.

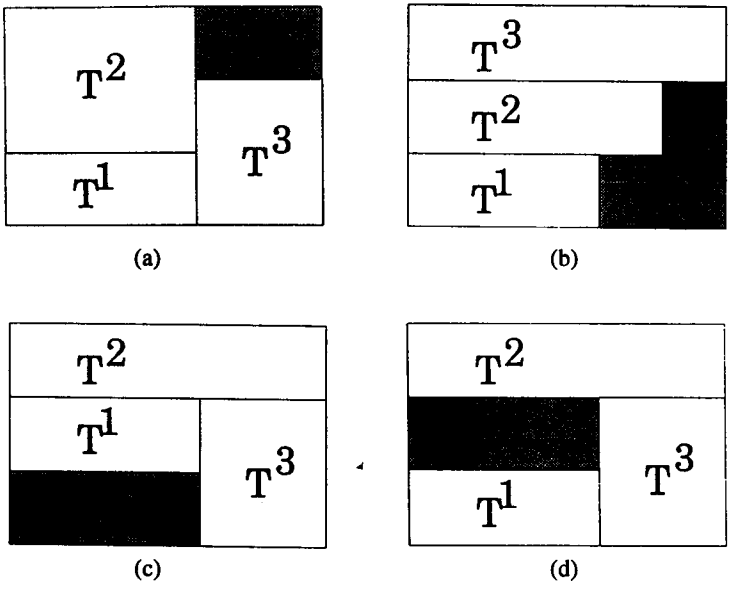


Figure 5. Example 2. The optimal schedules.

Let us comment on the two just presented dynamic programming algorithms. They are correct thanks to the fact that for $P2|set_j|C_{max}$ and the restricted version of $P3|set_j|C_{max}$ the structure of the optimal schedule is fixed (known in advance) and the order of execution of tasks processed in a certain mode is not important.

2.3. $P|set_j|C_{max}$

Here, the number of processors m is not fixed and is given as a parameter in the instance of the problem. The optimal solution in this case can be obtained by applying an exponential (in the worst case) branch and bound algorithm proposed in [4]. In this subsection, we will present a simple heuristic with the worst-case estimation.

ALGORITHM H:

Schedule tasks one by one in any order in their shortest processing time mode.

Let $C_{\max}^{H}$ denote the length of the schedule obtained from the heuristic and $C_{\max}^{*}$ the length of the optimal schedule.

THEOREM 1

For the assumptions stated above

$$\frac{C_{\max}^{H}}{C_{\max}^{*}} \leq m$$

and this bound is tight.

Proof

Obviously, for any algorithm (also an optimization one)

$$C_{\max} \geq \frac{1}{m} \sum_{j=1}^{n} \min_{l=1,\ldots,|S_j|} \{t_j^{D_l}\}.$$

This follows from the fact that in the above formula tasks are treated as preemptive ones, and multiprocessor tasks requiring a set D of processors simultaneously are treated as $|D|$ tasks requiring one processor. On the other hand,

$$C_{\max}^{H} \leq \sum_{j=1}^{n} \min_{l=1,\ldots,|S_j|} \{t_j^{D_l}\},$$

because, when the above relation holds with equality, all tasks have to be executed sequentially and no parallelism is achieved. Thus,

$$\frac{C_{\max}^{H}}{C_{\max}^{*}} \leq m.$$

Now, we will show that this bound is tight.

Consider the instance consisting of n tasks to be scheduled on m processors. Processing requirements of all tasks are identical and given by

$$\begin{aligned} t_j^{1} &= a && \text{for } j = 1,\ldots,n, \\ t_j^{k} &= a + \varepsilon && \text{for } k = \{2\},\ldots,\{m\} \text{ and } j = 1,\ldots,n, \\ t_j^{D} &= +\infty && \text{for } D \neq \{1\},\ldots,\{m\} \text{ and } j = 1,\ldots,n, \end{aligned}$$

where a, ε are some positive numbers and $\varepsilon < am/n$.

The length of the schedule according to the heuristic is equal to $C^H_{\max} = na$, while the optimal schedule (cf. figure 6) has length $C^*_{\max} = (n/m((a + \varepsilon)))$.

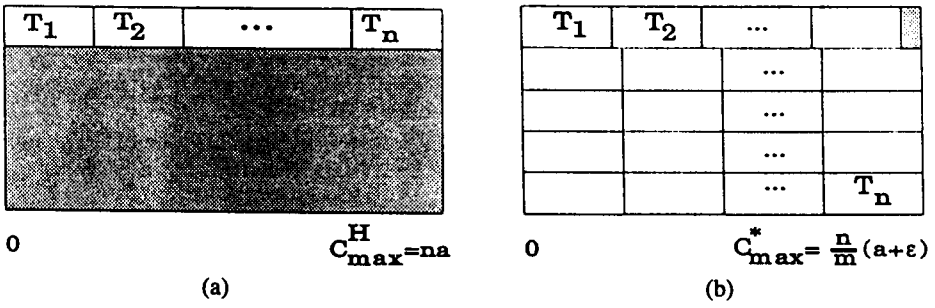


Figure 6. Heuristic H. The schedule built by heuristic H (a), and the optimal schedule (b).

Thus,

$$\lim_{\varepsilon \to 0} \frac{C^H_{\max}}{C^*_{\max}} = \lim_{\varepsilon \to 0} \frac{m}{1 + \frac{\varepsilon}{a}} = m. \qquad \square$$

This performance bound suggests that further work on the design of better heuristics seems to be necessary.

3. Preemptive scheduling

In this section, we will analyze the problem $P|set_j, pmtn|C_{\max}$, i.e. the problem when the tasks can be interrupted and then restarted without additional cost, probably on different processor(s). One has to deal with this kind of scheduling, for example, during scheduling file transfers on the set of buses [16]. In this case, transmission can be interrupted and then restarted using different bus(es). For a small number of processors, simple polynomial time algorithms are known [3]. In general (when the number of processors is unbounded), the problem in question is NP-hard in the strong sense [19]. For a limited number of processors, however, this problem can be solved in polynomial time using a linear programming procedure (i.e. for $Pm|set_j, pmtn|C_{\max}$).

Before presenting further details of the algorithm, let us introduce the notion of the *processor feasible set of tasks*. The processor feasible set of tasks is a set of tasks which can be feasibly executed in parallel on a given set of processors. Let there be a set $\mathcal{Q}$ of all feasible sets. Let $Q_j^{D_i}$ denote the set of indices of the processor feasible sets including task T_j processed in mode D_i. Let us assign variable x_i to each processor feasible set in $\mathcal{Q}$.

Now, our problem can be formulated in terms of linear programming:

$$\text{minimize} \quad \sum_{i \in Q} x_i$$

$$\text{subject to} \quad \sum_{D_i \in S_j} \sum_{l \in Q_j^{D_i}} \frac{x_l}{t_j^{D_i}} \geq 1 \quad \text{for } j = 1, \ldots, n.$$

We see that the number of variables in the above LP is $O(n^m)$ and the number of constraints is $O(n)$. Thus, for a fixed value of m, the number of variables is polynomially bounded. Since LP can be solved in polynomial time in the number of variables and constraints, we get that in the case of problem $Pm \mid set_j, pmtn \mid C_{\max}$, this approach can be implemented to run in time polynomial in the number of tasks.

4. Results of computational experiments

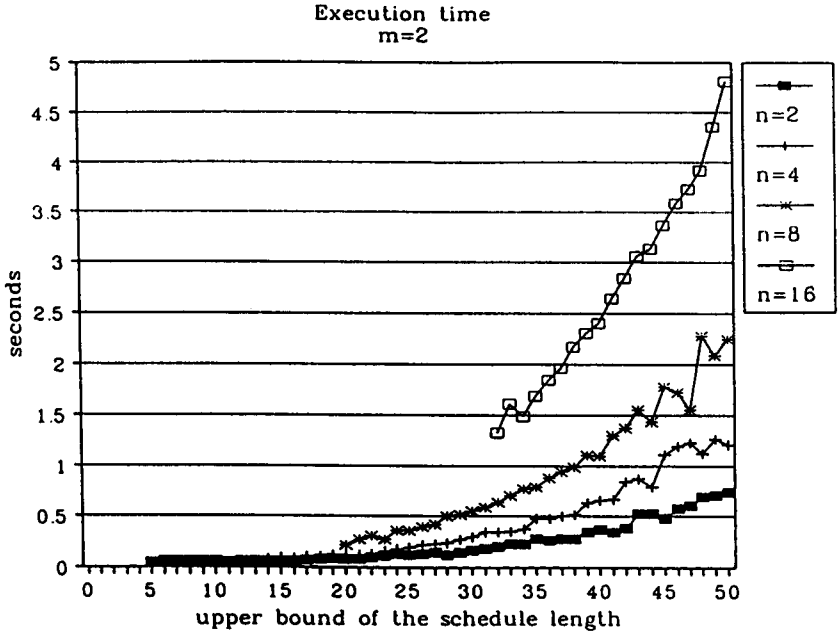
In this section, we will present the results of the computational experiments conducted over the algorithms presented in the preceding sections. First, for non-preemptive scheduling, we compare algorithms based on the dynamic programming against the heuristic algorithm using the shortest processing time mode of the task. Then, the results for preemptive scheduling and the algorithm based on linear programming are presented.

4.1. NONPREEMPTIVE SCHEDULING

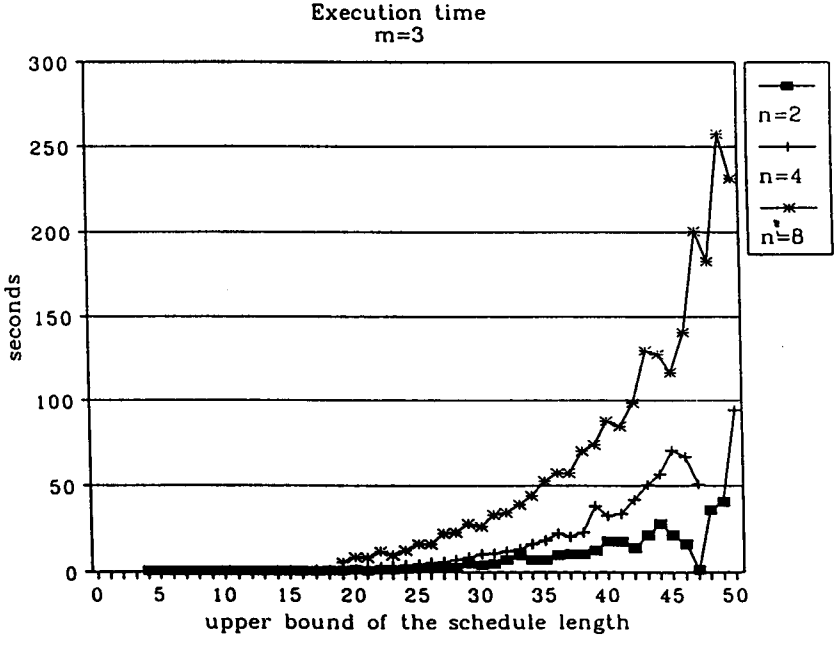
All the results have been collected in a series of experiments executed on PC-386. The simulating software has been written in BORLAND PASCAL version 7.0. Task parameters were generated in the following way. First, the number of processing modes in which a given task can be executed was generated with a uniform probability distribution. Then, particular processing modes for the task were chosen with uniform distribution of the probability. Finally, the execution time of the task for a given mode was generated, again with a uniform distribution of the probability.

The results are presented in figures 7(a) through 9(b). The execution time of the algorithm versus the number of tasks and the upper bound of the schedule are depicted in figure 7(a) for $P2 \mid set_j \mid C_{\max}$ and in figure 7(b) for $P3 \mid set_j \mid C_{\max}$. Memory consumption versus the number of tasks and the upper bound of the schedule length are presented in figure 8(a) for $P2 \mid set_j \mid C_{\max}$ and in figure 8(b) for $P3 \mid set_j \mid C_{\max}$. In figures 9(a) and 9(b), the average distance between the schedule length obtained by heuristic H and the optimum is presented. Each of the lines is a result of more than a thousand experiments. We have executed over 8300 experiments all together.

As can be seen in figures 7(a) and 7(b), the execution time grows faster than linearly (but slower than exponentially) and stays within 5 seconds for the two-

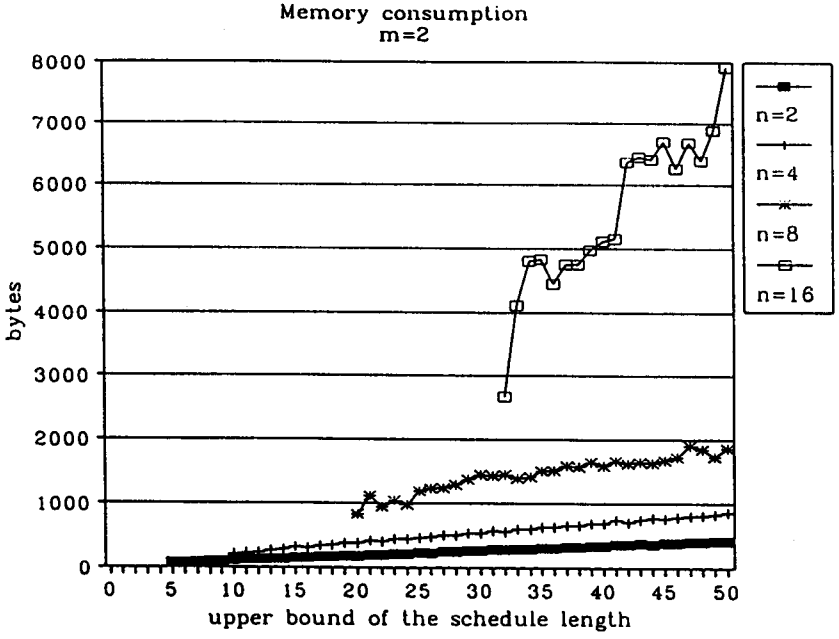


(a)

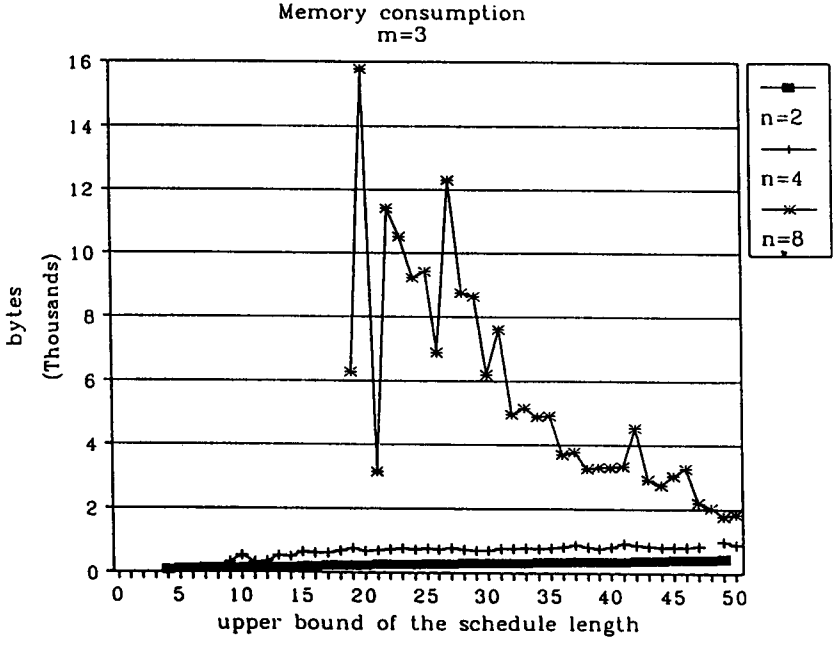


(b)

Figure 7. Dynamic programming approach. Execution time of the algorithm for (a) two and (b) three processors.

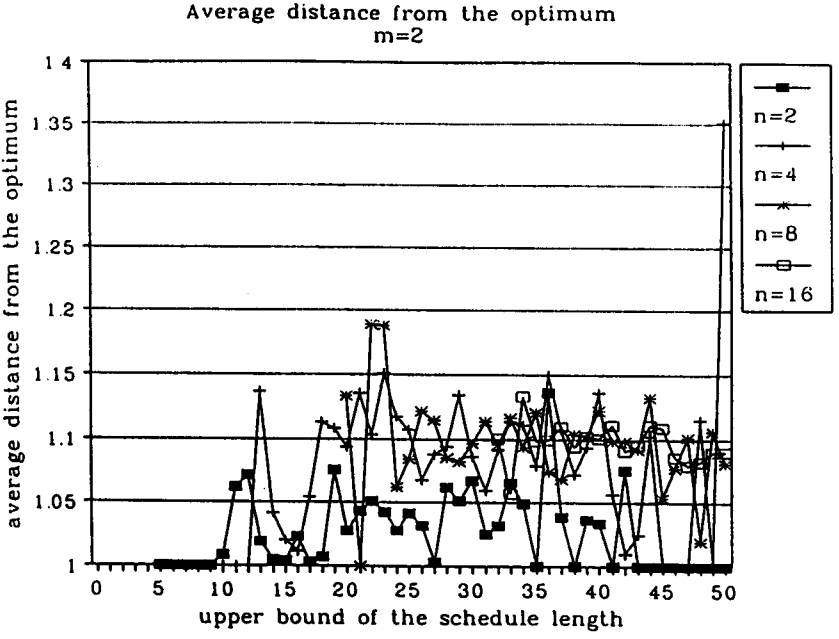


(a)

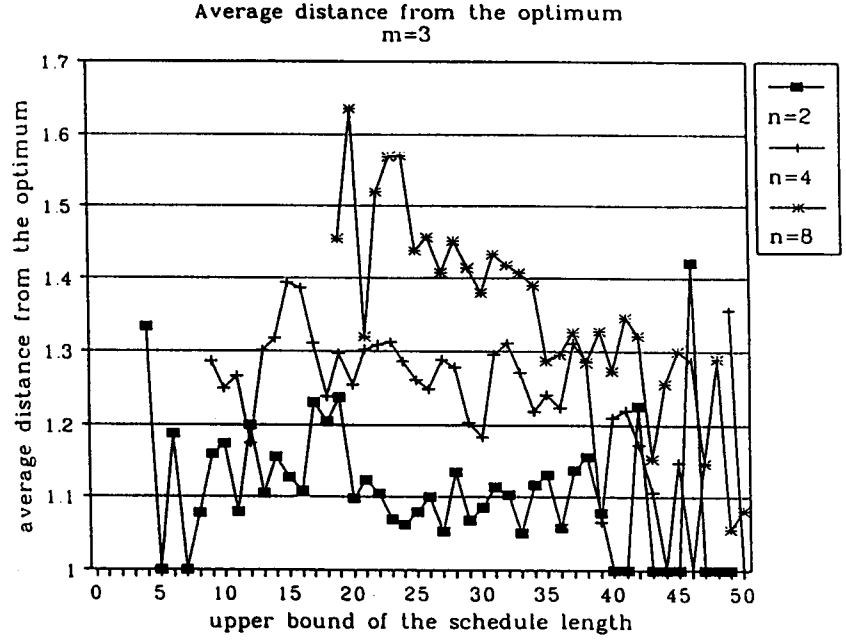


(b)

Figure 8. Dynamic programming approach. Memory consumption of the algorithm for (a) two and (b) three processors.



(a)



(b)

Figure 9. Heuristic H. Average distance from the optimum for (a) two and (b) three processors.

processor problem with 16 tasks and upper bound equal to 50, and within 4–5 minutes for the three-processor problem with 8 tasks and upper bound equal to 50.

Memory consumption is reasonably low, which is surprising when considering the fact that the table of the function $f(i, j, x, y)$ has four dimensions and five dimensions for the function $f(i, j, x, y, z)$. Since such a table is rather sparse, it would be inefficient to maintain a large matrix full of zeros. Thus, a different approach has been adopted. In each cell of a two-dimensional table of variables i and j, a list of non-zero entries of $f(i, j, x, y)$ or $f(i, j, x, y, z)$, respectively, has been stored. It can be seen in figure 8(b) that the memory consumption for 8 tasks decreases with the growth of the upper bound. This can be explained in the following way. Tasks with a larger upper bound of the schedule also have longer execution times. The result is that more feasible schedules have length greater than the upper bound and such schedules are not taken into consideration.

The last diagram in this subsection shows the average distance from the optimum of the schedule built with the heuristic H. As can be seen in figures 9(a) and 9(b), the distance from the optimum is about 10% for two processors and 30–40% for three processors. It turns out that heuristic H produces quite good solutions compared with the worst-case performance, equal to 2 for two processors and 3 for three processors. What is more, in all experiments the execution time of the heuristic H has been below 60 milliseconds. On the other hand, solutions close to the theoretic performance bound were also observed.

4.2. PREEMPTIVE SCHEDULING

The method of data generation was the same for testing the preemptive scheduling algorithm as for testing the nonpreemptive ones except for the fact that the number of processing modes for a task was generated with uniform probability distribution from the range [1, 6]. The execution time of the algorithm, memory consumption and number of variables in the linear program versus the number of tasks and processors are presented in figures 10 through 12. Each line in these figures presents results collected in more than a thousand experiments.

As can be seen in figures 10 and 11, execution time and memory consumption are growing very quickly with the number of processors, but much more slowly with the number of tasks (when m is fixed).

In figure 12, the number of variables in the linear program versus the number of tasks and processors is presented. It can be found that due to the mutual exclusion in the access to the processors, the number of feasible sets of tasks is not growing as drastically as could be expected according to the worst-case estimation ($O(n^m)$).

We may conclude that the algorithms presented earlier in this work can be efficiently implemented both from the point of view of execution time and memory limits.

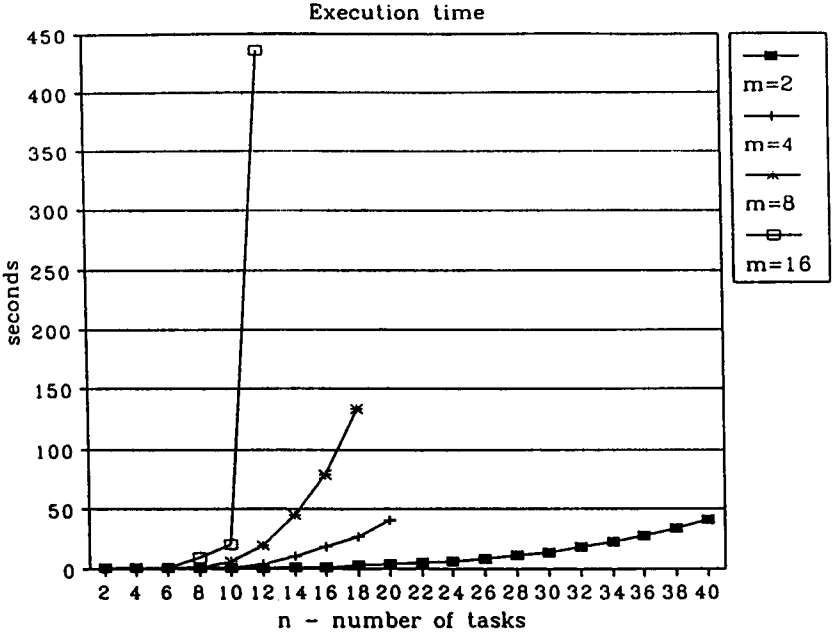


Figure 10. Preemptive scheduling algorithm based on linear programming. Execution time.

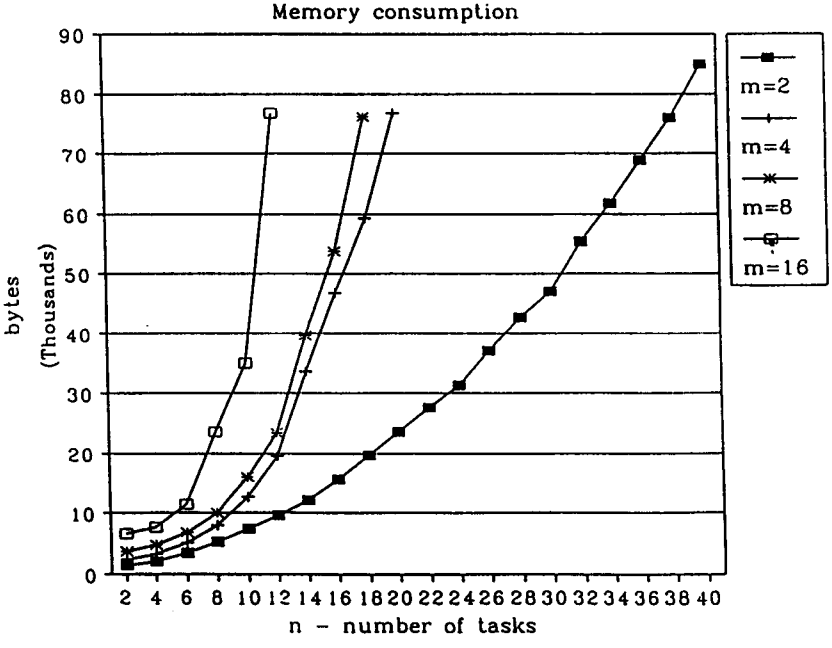


Figure 11. Preemptive scheduling algorithm based on linear programming. Memory consumption.

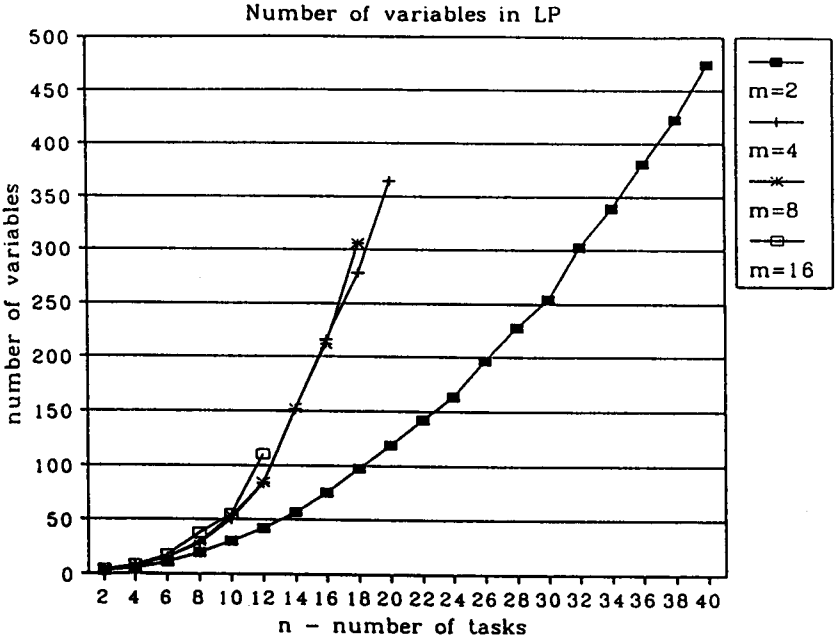


Figure 12. Preemptive scheduling algorithm based on linear programming. Number of variables in the linear program.

5. Conclusions

In this paper, we have presented a new model of scheduling tasks requiring more than one processor at a time. It was assumed that a task may be processed by some alternative sets of processors and that its processing time depends on the set of processors processing it. In this way, we have extended the model in which tasks can be processed by one set of processors simultaneously. For the case of nonpreemptive schedules, dynamic programming algorithms were presented for two processors and for a limited version of the three-processor problem. For the general case of nonpreemptive scheduling, a heuristic with a tight worst-case bound has been given. In the case of preemptive schedules and a limited number of processors, a linear programming approach has been proposed. All these algorithms have been experimentally tested and it appeared that the execution times and memory consumption remain within reasonable bounds.

Further research in this area may include other optimality criteria (e.g. maximum lateness L_{max} or mean flow time $\overline{F}$, which are very important from the practical point of view), or designing heuristics with better performance guarantees for the nonpreemptive case of the problem.

References

[1] M.J. Atallah, C. Lock Black and D.C. Marinescu, Models and algorithms for coscheduling computer-intensive tasks on a network of workstations, J. Parallel Distributed Comput. 16(1992)319–327.

[2] K.R. Baker, *Introduction to Sequencing and Scheduling*, (Wiley, New York, 1974).
[3] L. Bianco, J. Błażewicz, P. Dell'Olmo and M. Drozdowski, Scheduling preemptive multiprocessor tasks on dedicated processors, Perf. Eval. 20(1994)361–371.
[4] G. Bozoki and J.P. Richard, A branch-and-bound algorithm for continuous-process task shop scheduling problem, AIIE Trans. 2(1970)246–252.
[5] J. Błażewicz, W. Cellary, R. Słowiński and J. Węglarz, *Scheduling under Resource Constraints: Deterministic Models* (J.C. Baltzer, Basel, 1986).
[6] J. Błażewicz, P. Dell'Olmo, M. Drozdowski and M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors, IPL 41(1992)275–280.
[7] J. Błażewicz, M. Drabowski and J. Węglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Trans. Comput. C-35(1986)389–393.
[8] J. Błażewicz, M. Drozdowski, G. Schmidt and D. de Werra, Scheduling independent multiprocessor tasks on a uniform *k*-processor system, Parallel Comput. 20(1994)15–28.
[9] J. Błażewicz, K. Ecker, G. Schmidt and J. Węglarz, *Scheduling in Computer and Manufacturing Systems* (Springer, New York, 1993).
[10] J. Błażewicz, J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: Classification of complexity, Discr. Appl. Math. 5(1983)11–24.
[11] G.I. Chen and T.H. Lai, Preemptive scheduling of independent jobs on a hypercube, IPL 28(1988) 201–206.
[12] E.G. Coffman, Jr. (ed.), *Computer and Job-Shop Scheduling Theory* (Wiley, New York, 1976).
[13] J. Du and J.Y.-T. Leung, Complexity of scheduling parallel task systems, SIAM J. Discr. Math. 2(1989)473–487.
[14] R.I. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Ann. Discr. Math. 5(1979)287–326.
[15] D.G. Feitelson and L. Rudoloh, Gang scheduling performance benefits for fine-grain synchronization, J. Parallel Distributed Comput. 16(1992)306–318.
[16] R. Jain, K. Somalwar, J. Werth and J.C. Browne, Scheduling parallel I/O operations in multiple bus systems, J. Parallel Distributed Comput. 16(1992)352–362.
[17] H. Krawczyk and M. Kubale, An approximation algorithm for diagnostic test scheduling in multicomputer systems, IEEE Trans. Comput. C-34(1985)869–872.
[18] M. Kubale, The complexity of scheduling independent two-processor tasks on dedicated processors, IPL 24(1987)141–147.
[19] M. Kubale, Preemptive scheduling of two-processor tasks on dedicated processors, Zeszyty Naukowe Politechniki Śląskiej, Seria: Automatyka z. 100, No. 1082 (1990) 147–153, in Polish.
[20] J. Plehn, Preemptive scheduling of independent jobs with release times and deadlines on a hypercube IPL 34(1990)161–166.
[21] B. Veltman, B.J. Lageweg and J.K. Lenstra, Multiprocessor scheduling with communication delays, Parallel Comput. 16(1990)173–182.
[22] B. Veltman, Multiprocessor scheduling with communication delays, Ph.D. Thesis, CWI, Amsterdam (1993).
[23] Q. Wang and K.H. Cheng, A heuristic of scheduling parallel tasks and its analysis, SIAM J. Comput. 21(1992)281–294.