ORSA Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Optimal Scheduling of Tasks on Identical Parallel Processors

Mauro Dell'Amico, Silvano Martello,

ORSA Journal on Computing
Vol. 7, No. 2, Spring 1995

0899-1499/95/0702-0191 $01.25

Optimal Scheduling of Tasks on Identical Parallel Processors

MAURO DELL'AMICO / *Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy; Email: dellamico@ipmel1.polimi.it*

SILVANO MARTELLO / *DEIS—University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy; Email: silvano@boder1.cineca.it*

(Received: February 1991; final revision received: June 1992; accepted: January 1994)

We consider the classical problem of scheduling *n* tasks with given processing time on *m* identical parallel processors so as to minimize the maximum completion time of a task. We introduce lower bounds, approximation algorithms and a branch-and-bound procedure for the exact solution of the problem. Extensive computational results show that, in many cases, large-size instances of the problem can be solved exactly.

Given n *tasks* $T_1, \dots, T_n$ with associated processing times $p_1, \dots, p_n$, and m parallel identical *processors* $P_1, \dots, P_m$, each of which can process at most one task at a time, the *Multiprocessor Scheduling Problem* is to assign each task to exactly one processor so that the maximum completion time of a task (*makespan*) is minimized. Using the three-field classification introduced in Graham et al.,[10] the problem is denoted as $P\|C_{\max}$.

We assume, as is usual, that the processing times are positive integers and that $1 < m < n$. $C^*_{\max}$ denotes the optimal solution value, and $C_{\max}(A)$ the solution value produced by an approximation algorithm A.

The problem is known to be NP-hard in the strong sense (see Garey and Johnson[8]). A great variety of approximation algorithms can be found in the literature (see Lawler et al.[12] and Cheng and Sin[4] for recent surveys), while very little has been done on exact algorithms. For small values of m and U (an upper bound on $C^*_{\max}$), the problem can be solved exactly, in $O(nU^m)$ time through dynamic programming, as described, e.g., in Blazewicz.[1] An exact algorithm for a generalization of the problem can be found in Bratley et al.[3]

$P\|C_{\max}$ is known to be closely related to the bin packing problem. Given n items, each having an associated size p_j ($j = 1, \dots, n$), and an unlimited number of identical bins of capacity C, the *Bin Packing Problem* (*BPP*) is to assign each item to one bin, without exceeding the capacity, so that the number of bins used is minimized. This analogy has been used by Coffman et al.[5] to obtain the so-called *Multifit* approximation algorithm for $P\|C_{\max}$, which finds, through binary search, the smallest value C such that the solution found for *BPP* by the well known *First-Fit Decreasing* (*FFD*) approximation algorithm has value not greater than m. Hochbaum and Shmoys[11] have obtained a polynomial-time approximation scheme for $P\|C_{\max}$ by replacing *FFD* with a dual approximation algorithm. These ideas can also be used to solve $P\|C_{\max}$ exactly. Given a lower bound L and an upper bound U on $C^*_{\max}$, we can determine, through binary search, the smallest value C ($L \leqslant C \leqslant U$) such that the exact solution to *BPP* uses no more than m bins. The bin packing problem is NP-hard in the strong sense. Martello and Toth,[15] however, have recently given a branch-and-bound effective algorithm, and the corresponding Fortran listing (MTP), showing that in many cases it can find the exact bin packing solution. Our computational results (see Section 4, Table 1) show that this simple approach to $P\|C_{\max}$ can solve several instances for which dynamic programming is inadequate.

In the following sections we develop a branch-and-bound algorithm for the exact solution of $P\|C_{\max}$ and show, through computational experiments, that it outperforms the bin packing approach and solves, within a few seconds, large instances of the problem. In Section 1 we present lower bounds and analyze some of them through an extension of the concept of worst-case performance. In Section 2 the structure of the optimal solutions to the problem is analyzed, and sufficient conditions to determine upper and lower bounds on the number of tasks per processor are obtained. The branch-and-bound algorithm is introduced in Section 3 and experimentally analyzed in Section 4.

Unless otherwise specified, we will always assume that the tasks are ordered so that

$$p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n. \tag{1}$$

1. Lower Bounds

In this section lower bounds for $P\|C_{\max}$ are examined and, for some of them, the worst-case performance ratio is obtained. Given a lower bound procedure L for a problem P, let $L(I)$ and $z(I)$ be the value produced by L and the optimal solution value, respectively, for an instance I of P. The *worst-case performance ratio* of L is then the maximum value $R(L)$ such that

$$\frac{L(I)}{z(I)} \geqslant R(L) \quad \text{for all } I.$$

When no confusion arises, we denote with L both the lower bound procedure and the value it produces for a specific problem instance.

Subject classifications: Production/scheduling, sequencing, deterministic, multiple machine: identical machines. Programming, integer, algorithms: lower bounds, branch-and-bound.

1.1. Simple Bounds

The most immediate lower bound for $P\|C_{max}$ is the solution value of the relaxation we obtain by assuming that each task can be preempted and assigned to more than one processor. If we also assume that a preempted task can be processed in parallel, such a value is clearly provided by the continuous relaxation of the problem, computable as

$$L_0 = \left\lceil \frac{1}{m} \sum_{j=1}^{n} p_j \right\rceil. \tag{2}$$

The worst-case performance of L_0 is arbitrarily bad (i.e. $R(L_0) = 0$), as shown by the series of instances with $n = m + 1$, $p_1 = m$, $p_2 = \cdots = p_n = 1$, for which $L_0 = 2$ and $C^*_{max} = m$, so L_0/C^*_{max} is arbitrarily close to 0 for m sufficiently large. A better bound is obtained by preventing a preempted task from being processed in parallel. The solution value of the resulting *preemptive relaxation* is then (McNaughton[16])

$$L_1 = \max\left(L_0, \max_j \{p_j\}\right). \tag{3}$$

The worst-case performance ratio of L_1 can easily be derived from the Graham[9] analysis of the *List Scheduling* (LS) approximation algorithm for $P\|C_{max}$. This consists of sequentially assigning each task (in some prespecified order) to the processor whose current workload is a minimum, without introducing idle times. Let $C_{max}(LS)$ denote the solution value found, and T_l the task with maximum completion time. Graham proved that, since no processor can be idle before time $C_{max}(LS) - p_l$,

$$C_{max}(LS) \leqslant \frac{1}{m} \sum_{\substack{j=1 \\ j \neq l}}^{n} p_j + p_l = \frac{1}{m} \sum_{j=1}^{n} p_j + \frac{m-1}{m} p_l, \tag{4}$$

from which one easily has $L_1/C^*_{max} \geqslant m/(2m-1) > {}^1/_2$. The series of instances with $n = m + 1$ and $p_1 = \cdots = p_n = m$, for which the ratio L_1/C^*_{max} is arbitrarily close to ${}^1/_2$ for m sufficiently large, shows that the worst-case performance ratio of L_1 is $R(L_1) = {}^1/_2$.

The time complexity for the computation of L_0 and L_1 is $O(n)$, since it is not necessary to sort the tasks according to (1).

Lower bound L_1 can be further improved by considering the relaxation of $P\|C_{max}$ we obtain by eliminating the $n - m - 1$ smallest tasks $T_{m+2}, \ldots, T_n$. The optimal solution value of the relaxed problem is clearly no less than $p_m + p_{m+1}$, so an improved bound is

$$L_2 = \max(L_1, p_m + p_{m+1}). \tag{5}$$

Theorem 1. $R(L_2) = {}^2/_3$.

Proof. We first show that $L_2/C^*_{max} \geqslant {}^2/_3$ for any instance of the problem. Let $C_{max}(LS)$ be the solution value given by the List Scheduling algorithm when the m largest tasks are considered first, hence scheduled one per processor. Let T_l be the task with maximum completion time. Two cases may occur:

(a) if T_l is one of the m largest tasks, then $C^*_{max} = \max_j\{p_j\} = L_1 = L_2$;

(b) otherwise we observe that $p_l \leqslant p_{m+1} \leqslant L_2/2$; since $L_2 \geqslant (1/m)\Sigma_{j=1}^{n} p_j$ and $C^*_{max} \leqslant C_{max}(LS)$, we then have from (4)

$$\frac{L_2}{C^*_{max}} \geqslant \frac{2m}{3m-1} > \frac{2}{3}.$$

To see that the bound is tight, consider the series of instances with m even, $n = m + 2$, $p_1 = \cdots = p_{m-1} = m$, $p_m = p_{m+1} = p_{m+2} = m/2$. We have $L_2 = m + 1$ and $C^*_{max} = {}^3/_2 m$ so the ratio L_2/C^*_{max} is arbitrarily close to ${}^2/_3$ for m sufficiently large.

The time complexity for the computation of L_2 is $O(n)$. In this case too, in fact, no sorting is needed, since the $m - th$ and $(m+1) - th$ smallest processing times can be found in $O(n)$ time (see, e.g., Blum et al.[2] and Fischetti and Martello[7]).

A different lower bound, based on the possible configurations of the tasks with large p_j value, can be derived from the results in Hochbaum and Shmoys,[11] i.e.,

$$L_{HS} = \max(L_1, \max\{L + 1: B_\gamma(L) > m\}),$$

where $B_\gamma(L)$ is the number of bins used by their algorithm ${}^1/_5$-dual, when applied to the bin-packing instance determined by the tasks having $p_j > L/5$, with bin capacity L. Since the ${}^1/_5$-dual algorithm runs in $O(n)$ time, the time complexity for the computation of L_{HS} is $O(n \log U)$.

1.2. A Better Bound

Given an instance of $P\|C_{max}$ and an integer value L, let $BPP[p_j; L]$ denote the bin packing problem defined by item sizes $p_1, \ldots, p_n$ and bin capacity L. If the corresponding optimal solution value $z(BPP[p_j; L])$ exceeds m, it is clear that $L + 1$ is a valid lower bound value for $P\|C_{max}$. This idea is used in the following

Theorem 2. *Given an instance of* $P\|C_{max}$, *and two values* L *and* $\bar{p} \leqslant L/2$, *let*

$$J_1 = \{j: L - \bar{p} < p_j\};$$
$$J_2 = \{j: L/2 < p_j \leqslant L - \bar{p}\};$$
$$J_3 = \{j: \bar{p} \leqslant p_j \leqslant L/2\},$$

and define

$$B_\alpha(L, \bar{p}) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{\Sigma_{j \in J_3} p_j - (L|J_2| - \Sigma_{j \in J_2} p_j)}{L} \right\rceil\right), \tag{6}$$

$$B_\beta(L, \bar{p}) = |J_1| + |J_2| + \max\left(0, \left\lceil \frac{|J_3| - \Sigma_{j \in J_2} \left\lfloor \frac{L - p_j}{\bar{p}} \right\rfloor}{\left\lfloor \frac{L}{\bar{p}} \right\rfloor} \right\rceil\right); \tag{7}$$

if $B_\alpha(L, \bar{p}) > m$ *or* $B_\beta(L, \bar{p}) > m$ *then* $L + 1$ *is a valid lower bound for the instance.*

Proof. In any feasible solution to $BPP[p_j; L]$, no two items of $J_1 \cup J_2$ can be assigned to the same bin, and no item of J_3 can be assigned to a bin containing an item of J_1. Hence $z(BPP[p_j; L]) \geqslant |J_1| + |J_2| + b(J_3)$ where $b(J_3)$ is any lower bound on the number of additional bins needed for those items of J_3 which cannot be assigned to the bins containing items of J_2. For $B_\alpha(L, \bar{p})$ (which was already proved in Martello and Toth[14] to be a lower bound on $z(BPP[p_j; L])$) $b(J_3)$ is computed by assuming that the items of J_3 can be split and assigned to different bins. For $B_\beta(L, \bar{p})$ we relax the instance by assuming that all the items of J_3 have size $\bar{p}$, so $\lfloor (L - p_j)/\bar{p} \rfloor$ is the maximum number of items of J_3 which can be assigned to the bin containing the item of size p_j $(j \in J_2)$; hence $|J_3| - \sum_{j \in J_2} \lfloor (L - p_j)/\bar{p} \rfloor$ is the minimum number of items of J_3 that must be assigned to additional bins. Since each additional bin can contain at most $\lfloor L/\bar{p} \rfloor$ items of J_3, $B_\beta(L, \bar{p})$ is a valid lower bound on $z(BPP[p_j; L])$. The claim follows.

Corollary 1. *A valid lower bound for* $P\|C_{\max}$ *is*

$$L_3 = \max\left\{ L + 1 \colon \exists \bar{p} \leqslant \frac{L}{2} \text{ for which } B_\alpha(L, \bar{p}) > m \text{ or } B_\beta(L, \bar{p}) > m \right\}.$$

Proof. Obvious.

Example. Let $m = 4$, $n = 10$, $(p_j) = (99, 76, 76, 75, 25, 13, 13, 13, 1, 1)$. Computation of the lower bounds described in Section 1.1 gives:

$$L_0 = 98;$$
$$L_1 = 99;$$
$$L_2 = 100.$$

Applying Theorem 2 with $L = 100$ and $\bar{p} = 13$, we get

$$J_1 = \{1\},\ J_2 = \{2, 3, 4\},\ J_3 = \{5, 6, 7, 8\};$$

$$B_\alpha(100, 13) = 1 + 3 + \max\left(0, \left\lceil \frac{64 - (300 - 227)}{100} \right\rceil\right) = 4;$$

$$B_\beta(100, 13) = 1 + 3 + \max\left(0, \left\lceil \frac{4 - \left(\left\lfloor \frac{24}{13} \right\rfloor + \left\lfloor \frac{24}{13} \right\rfloor + \left\lfloor \frac{25}{13} \right\rfloor\right)}{\left\lfloor \frac{100}{13} \right\rfloor} \right\rceil\right) = 5 > m;$$

hence 101 is a valid lower bound value.

Computing L_3 by trying all possible pairs $(L, \bar{p})$ would clearly be inefficient. In the next section we show how the search can be limited.

1.3. Efficient Implementation of L_3

Since lower bound L_2 can be computed very easily, it is convenient to apply Theorem 2 only for values $L \geqslant L_2$. Observe now that: (a) both $B_\alpha(L, \bar{p})$ and $B_\beta(L, \bar{p})$ are computed over a relaxed instance obtained by eliminating all the tasks having processing time $p_j < \bar{p}$ (see (6), (7)); (b) if an instance is relaxed by eliminating tasks $T_{m+2}, \ldots, T_n$ then L_2 is the optimal solution value. We have thus proved the following

Proposition 1. *Once* L_2 *has been computed, better lower bound values can be obtained through Theorem 2 only for values* L *and* $\bar{p}$ *such that*

$$L \geqslant L_2 \text{ and } \bar{p} \leqslant p_{m+2}. \tag{8}$$

Now let $\bar{P} = \{p_{j_1}, \ldots, p_{j_l}\}$ be the set of distinct values $p_j \leqslant p_{m+2}$, sorted in decreasing order. For any $L \geqslant L_2$, given a processing time $p_{j_k} \in \bar{P}$ and a value q such that $p_{j_k} > q > p_{j_{k+1}}$, consider the values $B_\alpha(L, p_{j_k})$, $B_\alpha(L, q)$, $B_\beta(L, p_{j_k})$ and $B_\beta(L, q)$. The quantity $|J_1| + |J_2|$ is independent of $\bar{p}$. Set J_3 contains the same elements for both values of $\bar{p}$ (namely, it includes the items of size p_{j_k}, but not those of size $p_{j_{k+1}}$). Set J_2 induced by p_{j_k} is a subset of that induced by q. So, from (6), $B_\alpha(L, p_{j_k}) \geqslant B_\alpha(L, q)$ (as already proved in Martello and Toth[14]) and, from (7), $B_\beta(L, p_{j_k}) \geqslant B_\beta(L, q)$. Hence

Proposition 2. *Only values in* $\{p_{m+2}, \ldots, p_n\}$ *must be considered for* $\bar{p}$ *when applying Theorem 2.*

In order to limit the number of values to be considered for L, we first prove the following

Lemma 1. *For any given* $\bar{p}$ *value,* $B_\alpha(L, \bar{p})$ *and* $B_\beta(L, \bar{p})$ *are monotonically nonincreasing as* L *increases.*

Proof. Given the sets J_1, J_2 and J_3 produced by values $\bar{p}$ and L, let R and S (with $|R| = r$, $|S| = s$) denote the sets of those elements which move from J_1 to J_2 and from J_2 to J_3, respectively, when values $\bar{p}$ and $L + 1$ are used. (By definition, no element can move from J_1 to J_3 since $p_j = L + 1 - \bar{p}$ for all $j \in R$.) $B_\alpha(L, \bar{p})$ can be written as $\max(|J_1| + |J_2|, |J_1| + \lceil \sum_{j \in J_2 \cup J_3} p_j / L \rceil)$, so $B_\alpha(L + 1, \bar{p}) = \max(|J_1| + |J_2| - s, |J_1| - r + \lceil \sum_{j \in J_2 \cup J_3} p_j/(L + 1) + \sum_{j \in R} p_j/(L + 1) \rceil) \leqslant B_\alpha(L, \bar{p})$ (since $p_j < L + 1$ for all $j \in R$). Similarly,

$$B_\beta(L + 1, \bar{p}) = |J_1| + |J_2| - s + \max\left(0, \left\lceil \frac{|J_3| + s - \sum_{j \in J_2} \left\lfloor \frac{L + 1 - p_j}{\bar{p}} \right\rfloor - \sum_{j \in R} \left\lfloor \frac{L + 1 - p_j}{\bar{p}} \right\rfloor + \sum_{j \in S} \left\lfloor \frac{L + 1 - p_j}{\bar{p}} \right\rfloor}{\left\lfloor \frac{L + 1}{\bar{p}} \right\rfloor} \right\rceil\right);$$

since $\sum_{j \in R} \lfloor (L + 1 - p_j)/\bar{p} \rfloor = r \geqslant 0$ and $s + \sum_{j \in S} \lfloor (L + 1 - p_j)/\bar{p} \rfloor = \sum_{j \in S} \lfloor (\bar{p} + L + 1 - p_j)/\bar{p} \rfloor \leqslant \sum_{j \in S} \lfloor (L + 1)/\bar{p} \rfloor$, we have

$$B_\beta(L+1,\bar{p}) \leqslant |J_1| + |J_2| - s$$

$$+\max\left(0, \left\lceil \frac{|J_3| - \sum_{j\in J_2}\left\lfloor \frac{l-p_j}{\bar{p}} \right\rfloor}{\left\lfloor \frac{L}{\bar{p}} \right\rfloor} + \frac{\sum_{j\in S}\left\lfloor \frac{L+1}{\bar{p}} \right\rfloor}{\left\lfloor \frac{L+1}{\bar{p}} \right\rfloor} \right\rceil\right)$$

$$= \max\left(|J_1| + |J_2| - s, |J_1| + |J_2| + \left\lceil \frac{|J_3| - \sum_{j\in J_2}\left\lfloor \frac{L-p_j}{\bar{p}} \right\rfloor}{\left\lfloor \frac{L}{\bar{p}} \right\rfloor} \right\rceil \right) \leqslant B_\beta(L,\bar{p}).$$

Hence, for each value of $\bar{p}$, the value of L producing the best lower bound can be found, through binary search, in $O(\log U)$ iterations, where U is any upper bound on $C^*_{\max}$. Following Proposition 2, for each value of L, $O(n)$ different values must be considered for $\bar{p}$. For each pair $(L, \bar{p})$, the computation of $B_\alpha(L,\bar{p})$ and $B_\beta(L,\bar{p})$ requires $O(n)$ time. (Note that this computation cannot be parametrized, since, whenever L or $\bar{p}$ change, all the addenda of $\sum_{j\in J_2}\lfloor(l - p_j)/\bar{p}\rfloor$ in (7) change too.) We have thus proved the following

Theorem 3. *Lower bound L_3 can be computed in $O(n^2 \log U)$ time, where U is any upper bound on $C^*_{\max}$.*

2. Solution Structure and Other Bounds

In this section we analyze the configuration of the optimal task-processor assignment, and, in particular, we determine an upper bound Θ and a lower bound ϑ on the number of tasks per processor.

Assume that a feasible solution of value U is known for an instance of $P\|C_{\max}$, and define

$$\Theta = \max\left\{q\colon \sum_{j=n-q+1}^{n} p_j < U\right\}; \tag{9}$$

it is clear that no solution of value less than U can have more than Θ tasks assigned to the same processor.

Observe that if $\Theta = 2$ (implying $n \leqslant 2m$), the instance is optimally solved by assigning task T_j to processor P_j for $j = 1,\dots,m$, and task T_{m+k} to processor P_{m-k+1} for $k = 1,\dots,n-m$.

In order also to determine a lower bound ϑ on the number of tasks assigned to any processor, let us define the (truncated) average number of tasks per processor $\mu = \lfloor n/m \rfloor$, and observe that $\vartheta \leqslant \mu$. We give two sufficient conditions for determining values σ for which $\vartheta \geqslant \sigma$.

Proposition 3. *Given any instance of $P\|C_{\max}$ and an integer $\sigma \leqslant \mu$, let $L(\sigma)$ be a lower bound on the solution value of the sub-instance consisting of $m-1$ processors and tasks $T_\sigma,\dots,T_n$. If $L(\sigma) \geqslant U$, then any solution of value less than U for $P\|C_{\max}$ has at least σ tasks assigned to each processor.*

Proof. Consider any solution in which a processor, say P_1, has $\sigma - 1$ or less tasks assigned. Since $P_2,\dots,P_m$ must process at least $n-(\sigma-1)$ tasks, the corresponding makespan is at least $L(\sigma)$, so the overall solution cannot have a value less than U.

Proposition 4. *Given an instance of $P\|C_{\max}$, a lower bound L on its solution value and an integer $\sigma \leqslant \mu$, if $\sum_{j=1}^{\sigma} p_j \leqslant L$ then there exists an optimal solution having at least σ tasks assigned to each processor.*

Proof. Given an optimal solution, let r_i denote the number of tasks processed by P_i ($i \in M = \{1,\dots,m\}$), and define $M^- = \{i\colon r_i < \sigma\}$, $M^+ = M \setminus M^-$. By definition of μ, the number of tasks assigned to processors P_i ($i \in M^+$) is at least $\mu m - \sum_{i\in M^-} r_i \geqslant \mu|M^+| + \sigma|M^-| - \sum_{i\in M^-} r_i$. Hence, if $M^- \neq \varnothing$, we can easily obtain an equivalent solution by moving $\sigma|M^-| - \sum_{i\in M^-} r_i$ tasks from processors P_i ($i \in M^+$) to processors P_i ($i \in M^-$) in such a way that each processor P_i ($i \in M^+$) has at least $\mu \geqslant \sigma$ tasks, while each processor P_i ($i \in M^-$) has exactly σ tasks. In the resulting solution, the makespan relative to processors P_i ($i \in M^-$) does not exceed L, so the solution value is no worse than the given one.

Propositions 3 and 4 imply the following

Theorem 4. *For any instance of $P\|C_{\max}$, given a feasible solution of value U and a lower bound value $L < U$, let*

$$\vartheta = \max\left\{\sigma\colon L(\sigma) \geqslant U \text{ or } \sum_{j=1}^{\sigma} p_j \leqslant L\right\} \tag{10}$$

(where $L(\sigma)$ is defined as in Proposition 3). The search for $C^*_{\max}$ can then be restricted to solutions in which at least ϑ tasks are assigned to each processor.

The time complexity for the computation of ϑ and Θ depends on the way U, L and $L(\sigma)$ are computed. If: (a) the tasks are sorted according to (1); (b) U is determined through algorithm LS (see Section 1.1); (c) L and $L(\sigma)$ are computed through L_2 (so, given $L(\sigma)$, $L(\sigma+1)$ can be calculated in constant time), then ϑ and Θ can be determined in $O(n \log n)$ time.

When an enumerative algorithm is used to solve $P\|C_{\max}$, determining Θ and ϑ can be very useful in limiting the search.

When the special case $\Theta = \vartheta + 1$ occurs, further information can be obtained. We can in fact determine the number m_ϑ (resp. m_Θ) of processors having ϑ (resp. Θ) tasks assigned, by solving the system ($m_\vartheta + m_\Theta = m$, $\vartheta m_\vartheta + (\vartheta+1)m_\Theta = n$):

$$m_\vartheta = (\vartheta+1)m - n; \tag{11}$$

$$m_\Theta = n - \vartheta m. \tag{12}$$

We thus know that the optimal solution is given by the union of the solutions of two separate and easier subproblems having ϑm_ϑ (resp. $(\vartheta+1)m_\Theta$) tasks and m_ϑ (resp. m_Θ) processors, with exactly ϑ (resp. $\vartheta+1$) tasks assigned to each processor. Unfortunately, there is no easy way to determine the bipartition of the tasks between the two subproblems. It is however possible to heuristically determine a tentative partition and solve the resulting subproblems, thus obtaining an approximate solution for the original problem (as will be seen in Section 3).

2.1. Bounds from the Solution Structure

The considerations introduced above lead to other lower bounds for $P\|C_{\max}$.

Given the average number of tasks per processor, n/m, we have that at least one processor must have $\nu = \lceil n/m \rceil$ or more tasks assigned, so

$$L_\nu = \sum_{j=n-\nu+1}^{n} p_j \tag{13}$$

is a valid lower bound.

When the special case $\Theta = \vartheta + 1$ occurs, we have another immediate bound depending on the current lower and upper bound values (since Θ is a function of U, and ϑ a function of L and U):

$$L_\vartheta(L, U) = \max\left(\left\lceil \sum_{j=n-\vartheta m_\vartheta+1}^{n} p_j/m_\vartheta \right\rceil, \left\lceil \sum_{j=n-(\vartheta+1)m_\Theta+1}^{n} p_j/m_\Theta \right\rceil\right) \tag{14}$$

(but when $\Theta = 2$ we give to $L_\vartheta(L, U)$ the optimal solution value, determined as previously described).

When $\Theta > \vartheta + 1$, we can consider relaxed instances obtained by removing the last $\bar{n}$ tasks ($\bar{n} = 1, 2, \dots$), determining $\bar{n}$ in such a way that the special case occurs, and computing $L_\vartheta(L, U)$ for the relaxed instance: let $\tilde{L}_\vartheta(L, U)$ denote the maximum value obtained for $L_\vartheta(L, U)$. It is not difficult to see that, given the values of Θ and ϑ for a current value $\bar{n}$, the computation of Θ and ϑ for $\bar{n} + 1$ can be done in constant time. Hence $\tilde{L}_\vartheta(L, U)$ has time complexity $O(n)$, plus $O(n \log n)$ for the initial sorting. The bound can be strengthened by determining, through binary search, the maximum value $\overline{U}(L < \overline{U} < U)$ such that $\tilde{L}_\vartheta(L, \overline{U}) \geqslant \overline{U}$: this implies that no solution of value $\overline{U} - 1$ or less can exist, so

$$L_\vartheta = \max(\overline{U}, L_\nu)$$

is a valid lower bound on $C^*_{\max}$. The time complexity for the computation of L_ϑ is clearly $O(n \log U)$, plus $O(n \log n)$ for the initial sorting.

We show that lower bounds L_ϑ, L_{HS} (Section 1.1) and L_3 (Section 1.2) do not dominate each other. In the example considered in Section 1.2, L_3 dominates the other bounds, since we have $L_3 = 101 > L_\vartheta = L_{HS} = 100$ (by using algorithm LS to obtain the value of U). Consider now the following instance: $n = 10$, $(p_j) = (98, 98, 98, 76, 69, 58, 55, 55, 52, 50)$. If $m = 3$, then L_{HS} dominates the other bounds since $L_{HS} = 245 > L_3 = L_\vartheta = 237$. If instead $m = 5$, then L_ϑ dominates the other bounds since $L_\vartheta = 153 > L_{HS} = 150 > L_3 = 148$.

An empirical comparison of the three bounds, performed over all the instances generated for the computational experiments of Section 4, showed that L_3 gives, on average, better values but requires higher running times. Using a simplified version of L_3, which only tries the value $\bar{p} = \max\{p_j\colon p_j \leqslant L/5\}$ (thus reducing the time complexity to $O(n \log U)$), we obtained for the three bounds the same average performances, both for values and running times. Observe that the dominance relations hold for the simplified version too, since value $L_3 = 101$ in the example of Section 1.2 was obtained using the above value of $\bar{p}$.

3. A Branch-and-Bound Algorithm

The results of the previous sections have been imbedded into a depth-first branch-and-bound algorithm for the exact solution of $P\|C_{\max}$. The branching strategy is as follows.

The tasks, sorted according to (1), are assigned to processors by increasing index. Let $C_{\max}$ denote the best incumbent solution value, and c_i ($i = 1, \dots, m$) the sum of the processing times of tasks currently assigned to processor P_i. At level k of the branch-decision tree, the current node generates $\overline{m} \leqslant m$ son nodes by assigning task T_k to processors P_i such that $c_i + p_k < C_{\max}$, by increasing c_i values. Since the assignment of T_k to processors with equal c_i value would obviously lead to equivalent solutions, for each subset of processors having identical c_i values, only the one of lowest index is considered.

For the root node, lower bound $\max(L_3, L_\vartheta)$ is computed as described in Sections 1.3 and 2.1. For each of the other nodes, generated, say, by assignment of task T_k, only lower bound L_3 is computed, by taking into account the current assignment in the following way. Theorem 2 is applied to a transformed instance, obtained by replacing tasks $T_1, \dots, T_k$ with m fictitious tasks, having processing times $c_1, \dots, c_m$. The computational effort is considerably decreased by applying the theorem for the unique value $L = C_{\max} - 1$: if a $\bar{p}$ value is found for which $B_\alpha(L, \bar{p}) > m$ or $B_\beta(L, \bar{p}) > m$, then a backtracking must occur; otherwise the depth-first search must continue. (Note that trying any lesser value of L could never allow the search to be stopped.) We also performed a series of computational tests by using, at the nodes, various combinations of weaker but quicker bounds (L_2, L_{HS}, L_ϑ) instead of L_3: the results indicated that the choice of L_3 produces the best computing times.

When task T_n is assigned, a new incumbent solution is determined, so $C_{\max}$ is updated. Let j_{max} be the lowest index of a task whose current completion time is $C_{\max}$. A series of backtracking is then performed, until task T_{jmax-1} is encountered and assigned to the next feasible processor. Note in fact that: (a) all tasks having completion time $C_{\max}$ must be removed from their current processor; (b) assigning T_{jmax} to the next processor without changing the previous assignments would produce a solution value higher than $C_{\max}$.

3.1. Dominance Criteria

The following criteria can be used to reduce the number of decision nodes generated.

Criterion 1. *If two (consecutive) tasks T_j, T_{j+1} have the same processing time, and T_j is currently assigned to processor P_k, at level $j+1$ only processors P_i such that $c_i \geqslant c_k - T_j$ must be considered for the assignment of T_{j+1}.*

In fact, for any P_h such that $c_h < c_k - p_j$, the solution corresponding to the assignment of T_{j+1} to P_h is identical to that (already generated) corresponding to the assignment of T_j to P_h and T_{j+1} to P_k.

Criterion 2. *At level j, let $\tilde{T} = \{T_j, \dots, T_n\}$ denote the set of unassigned tasks. If $|\tilde{T}| < m$, only the $|\tilde{T}|$ processors with smallest c_i values must be considered for the assignment of T_j.*

In such a situation, in fact, no more than $|\tilde{T}|$ processors will be used to complete the current solution.

The following observations can be used to further reduce the number of decision-nodes at the final levels of the tree. At level j, let $P_{min}(j)$ and $P_{smin}(j)$ be the two processors with minimum and second minimum c_i value, respectively. Observe that in the optimal completion of the current solution: (α) at least one task $T_k \in \tilde{T}$ is assigned to $P_{min}(j)$; (β) if exactly one task $T_k \in \tilde{T}$ is assigned to $P_{min}(j)$ then there is an optimal completion in which $T_k \equiv T_j$ (since the value of any solution in which $T_k \not\equiv T_j$ does not increase by interchanging T_j and T_k); so (γ) if T_j is not assigned to $P_{min}(j)$ then at least two tasks of $\tilde{T}$ are assigned to $P_{min}(j)$; hence (δ) task T_{n-1} must be assigned to $P_{min}(n-1)$. We thus obtain

Criterion 3. *At level $n-2$, the optimal completion of the current solution is the best between that obtained by sequentially assigning T_j to $P_{min}(j)$ (for $j = n-2, n-1, n$) and that obtained by assigning T_{n-2} to $P_{smin}(n-2)$, T_{n-1} and T_n to $P_{min}(n-2)$.*

Note in fact that two situations can occur: if T_{n-2} is assigned to $P_{min}(n-2)$ then the former completion is optimal, for the resulting solution, by observations (δ) and (α); otherwise the latter is optimal by observation (γ).

3.2. Initialization Phase

Before starting the branch-and-bound process, it is convenient to determine a good heuristic solution to the problem.

Many approximation algorithms for $P||C_{\max}$ are available from the literature (the interested reader is referred to Lawler et al.[12]). We have described in Section 1.1 the List Scheduling Algorithm *LS*. It is known from probabilistic analysis (see, e.g., Coffman et al.[6]) that good results are generally obtained if *LS* considers the items by decreasing p_j values; the resulting algorithm is called *Longest Processing Time* (*LPT*). A different approach is *Multifit* (*MF*) (see Coffman et al.,[5] Hochbaum and Shmoys[11]), which finds, through binary search, the smallest value U such that an approximate solution to $BPP[p_j; U]$ uses no more than m bins.

We have implemented the *MF* approach by solving the $BPP[p_j; U]$ instances with the approximation version of the Martello and Toth[15] code *MTP*. This is a branch-and-bound algorithm for the exact solution of the bin packing problem, which includes the First-Fit Decreasing approximation algorithm in the initialization step; its approximation version is obtained through an input parameter *BACK*, a limit on the number of backtrackings to be performed. Computational experiments showed that good average results are obtained with the value $BACK = 1500$. In addition, this approach can provide a lower bound value whenever, during the binary search, the solution value returned for the current U is greater than m and the number of backtrackings performed is less than *BACK*: we know in this case that the solution provided by *MTP* is exact, so $U+1$ is valid lower bound value for $P||C_{\max}$.

We have derived another heuristic approach from the experimental observation that lower bound L_3 gives in many cases the optimal solution value. The algorithm, called *Multi-Subset* (*MS*), operates in two phases. *Phase 1* tries to determine a solution of value L_3 by considering one processor at a time: each processor is assigned a subset of the currently unassigned tasks, such that the sum of the corresponding processing times is closest to, without exceeding, L_3. Determining such a subset is an NP-hard problem, known as *Subset Sum*, for which, however, efficient approximation algorithms exist in the literature; we used algorithm G^2 proposed by Martello and Toth.[13] After G^2 has been applied m times, let $\bar{T}$ be the set of unassigned tasks. If $\bar{T} = \varnothing$, we have an optimal solution of value L_3. Otherwise, let c_i be the sum of the processing times currently assigned to processor P_i ($i = 1, \dots, m$) and observe that $c_i + \min\{p_j: T_j \in \bar{T}\} > L_3$ for all i (since the solutions determined by G^2 are maximal), so $|\bar{T}| < m$ (proof: $|\bar{T}| \geqslant m$ would imply $\sum_{j=1}^{n} p_j = \sum_{i=1}^{m} c_i + \sum_{T_j \in \bar{T}} p_j > mL_3 \geqslant mL_0$). Hence *Phase 2* considers the tasks of $\bar{T}$ according to decreasing p_j values and the processors according to increasing c_i values, assigning one task per processor. Note that this is the optimal completion of the solution found in Phase 1, since each assignment increases the value of c_i, for the interested processor, to more than L_3, so it cannot be convenient to assign two tasks to the same processor.

The following procedure determines a heuristic solution by subsequently applying algorithms *LPT*, *MS* and *MF*, and produces a lower bound value $L = L_3$.

procedure H($m, T, C_{\max}, L$):
input: an instance of $P||C_{\max}$ defined by m processors and tasks set T;
output: an approximate solution of value $C_{\max}$ and a lower bound value L;
begin
  $C_{\max} := C_{\max}(LPT)$;
  $L := L_2$ (see Section 1.1);
  if $L = C_{\max}$ **then return;**
  compute L_3 (see Section 1.3) with binary search between L and $C_{\max}$, and set $L := L_3$;
  if $L = C_{\max}$ **then return;**
  $C_{\max} := \min(C_{\max}, C_{\max}(MS))$;
  if $L = C_{\max}$ **then return;**
  apply *MF* with binary search between L and $C_{\max}$ (possibly increasing the value of L), and set $C_{\max} := \min(C_{\max}, C_{\max}(MF))$
end.

The initialization phase can be summarized as follows. Procedure H is first executed for the input instance. The values of ϑ and Θ (see Section 2) are then computed. If $\vartheta = \Theta - 1$, we compute m_ϑ and m_Θ (see (11), (12)) and split, in a greedy way, the instance into two proper subinstances (having m_ϑ and m_Θ processors, respectively). A new approximate solution is then determined by applying procedure H to both subinstances. The pseudocode follows.

procedure INIT:
begin
  H($m, \{T_1, \dots, T_n\}, C_{\max}, L$);
  if $C_{\max} = L$ **then stop** (optimal solution);
  compute ϑ and Θ;
  if $\Theta = 2$ **then** determine the optimal solution (see Section 2) and **stop**;
  $L := \max(L, L_\vartheta)$ (see Section 2.1);
  if $\vartheta = \Theta - 1$ **then**
    begin
      compute m_ϑ and m_Θ;
      determine, in a greedy way, a subset S_ϑ of $\{T_1, \dots, T_n\}$
      such that $|S_\vartheta| = \vartheta m_\vartheta$ and $\Sigma_{T_j \in S_\vartheta} p_j$ is as close as possible to $(m_\vartheta/m)\Sigma_{j=1}^{n} p_j$;
      $S_\Theta := \{T_1, \dots, T_n\} \setminus S_\vartheta$;
      H($m_\vartheta, S_\vartheta, C^\vartheta_{\max}, L^\vartheta$);
      if $C^\vartheta_{max} \geqslant C_{\max}$ **then return**;
      H($m_\Theta, S_\Theta, C^\Theta_{max}, L^\Theta$);
      $C_{\max} := \min(C_{\max}, \max(C^\vartheta_{max}, C^\Theta_{max}))$;
      if $C_{\max} = L$ **then stop** (optimal solution)
    end
end.

4. Computational Experiments

We have coded in C language the following algorithms for the exact solution of $P||C_{\max}$:

DP: the dynamic programming algorithm described in Blazewicz[1];
BIN: the algorithm based on iterative solutions of bin packing problems (see the introduction), with $L = L_2$ and U computed through algorithm *LPT*;
B & B: the branch-and-bound algorithm described in Section 3.

We did not consider the enumerative algorithm proposed by Bratley et al.[3] for a generalization of $P||C_{\max}$, since their computational results show that it can solve only instances of very limited size.

We executed a series of computational experiments on a Digital VAXstation 3100, by considering five classes of test problems obtained by randomly generating the p_j values according to the following distributions:

Class 1: uniform in range [1, 100];
Class 2: uniform in range [20, 100];
Class 3: uniform in range [50, 100];
Class 4: normal with mean 100 and standard deviation 50;
Class 5: normal with mean 100 and standard deviation 20,

where classes 1–3 are derived from generations used to test bin-packing algorithms (see, e.g., Martello and Toth[14]).

For each class, and for different values of n and m, the entries in the tables give the average CPU time, computed over 10 problem instances. The bin packing code *MTP* used in algorithm *BIN* had, at each call, a limit of 5000 backtrackings assigned. Moreover, *MTP* was modified so as to fathom a decision node if the corresponding lower bound is greater than m, and to terminate as soon as a solution of value m is found. Algorithm *B & B* had a limit of 4000 backtrackings. For the cases where some of the 10 problems was not solved within the backtracking limit, we give, in brackets, the number of solved problems and compute the average time over them.

Table I compares the three algorithms on small-size problems. The results show that *DP* is clearly inefficient and cannot be used for larger instances, since its running time is exponential in m.

Table II compares *BIN* and *B & B* on large-size problems. *B & B* proves to be much better than *BIN* and capable of solving (with few exceptions) all types of problems in a few seconds. *BIN* solved quite easily all the problems of Class 1 (but 3), while its computational behaviour was bad for Classes 2 and 4, and very bad for Classes 3 and 5. The computational performance of *B & B* was in general satisfactory for all cases the only exception being small-size problems (n = 25 or 50, m = 10 or 15) of Classes 3 and 5. Apart from these, the average computing times of *B & B* grow with n (almost linearly) and m. The instances of Classes 1, 2 and 4 were solved more easily since the lower

Table I. VAXstation 3100 Seconds: Average Time Over 10 Problems

		Class 1			Class 2			Class 3			Class 4			Class 5		
m	n	DP	BIN	B & B	DP	BIN	B & B	DP	BIN	B & B	DP	BIN	B & B	DP	BIN	B & B
	10	0.03	0.01	0.01	0.02	0.01	0.01	0.03	0.02	0.01	0.03	0.02	0.01	0.03	0.03	0.01
2	25	0.15	0.02	0.01	0.14	0.03	0.01	0.17	0.04	0.01	0.13	0.04	0.01	0.14	0.04	0.01
	50	2.83	0.01	0.01	0.34	0.02	0.01	0.35	0.02	0.01	0.51	0.04	0.01	0.46	0.09	0.01
	10	2.68	0.01	0.01	4.09	0.03	0.03	12.66	0.02	0.02	13.64	0.03	0.04	14.78	0.04	0.03
3	25	42.57	0.01	0.01	65.28	0.04	0.01	183.22	0.07	0.01	197.25	0.15	0.01	212.45	0.48	0.01
	50	276.05	0.01	0.01	520.71	0.05	0.01	911.13	0.17	0.01	1369.04	0.07	0.01	1458.32	0.11	0.01

Table II. VAXstation 3100 Seconds: Average Time Over 10 Problems

m	n	Class 1		Class 2		Class 3		Class 4		Class 5	
		BIN	B & B	BIN	B & B	BIN	B & B	BIN	B & B	BIN	B & B
	10	0.01	0.01	0.03	0.03	0.02	0.02	0.03	0.04	0.07	0.03
	25	0.01	0.01	0.04	0.01	0.07	0.01	0.14	0.01	0.48	0.01
	50	0.01	0.01	0.05	0.01	0.17	0.01	0.08	0.01	0.16	0.01
	100	0.01	0.01	0.23	0.01	0.57	0.01	0.12	0.01	0.43	0.01
3	250	0.01	0.01	0.76	0.01	6.08	0.01	0.02	0.01	2.59	0.01
	500	0.01	0.01	4.37	0.02	47.56	0.02	0.23	0.01	23.47	0.02
	1000	0.01	0.01	28.22	0.03	107.19	0.04	0.02	0.02	149.44	0.05
	2500	0.03	0.03	470.66	0.08	4445.19	0.10	0.05	0.04	937.99	0.09
	5000	0.07	0.07	2189.15 (8)	0.17	—	0.30	0.11	0.07	2048.80 (9)	0.69
	10000	0.16	0.14	—	0.57	—	0.63	0.23	0.14	8172.88 (7)	1.30
	10	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.22	0.01
	25	0.15	0.01	0.08	0.01	1.07 (9)	0.01	0.66 (7)	0.01	1.42 (7)	0.02
	50	0.03	0.01	0.10	0.01	0.12	0.01	0.21 (8)	0.01	0.24 (8)	0.01
	100	0.06	0.01	0.23	0.01	1.07	0.01	0.27	0.01	1.61 (7)	0.01
5	250	0.01	0.01	0.01	0.01	0.01	0.01	0.18 (9)	0.01	6.22 (9)	0.01
	500	0.01	0.01	0.01	0.01	0.01	0.01	0.03	0.02	27.64 (9)	0.03
	1000	0.02	0.02	0.02	0.02	0.01	0.02	0.03	0.03	171.44	0.10
	2500	0.04	0.04	0.05	0.05	0.04	0.05	0.09	0.05	791.39	0.28
	5000	0.09	0.09	0.09	0.09	0.09	0.09	0.15	0.09	843.56 (2)	0.87
	10000	0.18	0.17	0.18	0.18	0.18	0.17	0.30	0.18	2096.57 (4)	2.12
	25	0.02	0.05	0.49	0.59	2.35	1.98	0.60	0.81	3.72 (8)	4.60 (9)
	50	0.04 (9)	0.01	0.46	0.01	0.21 (3)	1.66	1.29 (8)	0.01	— (0)	0.20 (8)
	100	0.07	0.01	0.67	0.01	0.20 (7)	0.01	0.69 (8)	0.01	8.47 (2)	0.02
	250	0.03	0.01	0.42	0.01	0.01 (9)	0.01	0.27	0.01	12.60 (8)	0.03
10	500	0.01	0.02	0.01	0.01	0.01	0.01	0.37	0.03	71.53 (8)	0.04
	1000	0.03	0.03	0.03	0.03	0.03	0.03	0.06	0.05	466.32	0.15
	2500	0.07	0.07	0.07	0.07	0.07	0.07	0.14	0.10	3557.45	0.73
	5000	0.14	0.13	0.14	0.14	0.14	0.14	0.25	0.14	534.84 (4)	1.92
	10000	0.28	0.26	0.29	0.28	0.28	0.27	0.50	0.29	190.48 (3)	0.99
	25	0.01	0.01	0.01	0.04	0.01	0.03	0.01	0.03	0.01	0.03
	50	0.20 (8)	0.05	2.72 (7)	0.52	3.86 (4)	11.65 (8)	14.27 (1)	0.05	— (0)	23.11 (7)
	100	0.30	0.01	1.59 (7)	0.01	— (0)	0.02	0.94 (5)	0.01	— (0)	2.16
	250	0.02	0.02	3.68	0.02	27.14 (3)	0.03	0.98	0.03	14.94 (2)	0.03
15	500	0.02	0.02	9.49	0.04	84.45 (1)	0.06	2.71	0.04	108.86 (8)	0.05
	1000	0.04	0.04	97.27	0.07	702.48 (5)	0.15	0.10	0.07	410.50 (7)	0.09
	2500	0.09	0.10	897.36	0.18	—	1.09	0.20	0.15	5030.91 (8)	0.90
	5000	0.19	0.18	—	0.64	—	3.84	0.36	0.21	—	1.56
	10000	0.39	0.37	—	1.32	—	1.91	0.68	0.39	—	9.17

In brackets, numbers of solved problems, if less than 10.

bound value computed at the root node was always very close to the optimal solution value.

It is worth noting that the problems of Class 3, for which finding the exact solution is comparatively harder, are easier to handle for an approximation algorithm because the number of tasks per processor is determined within a factor of two and the number of tasks of different size is small (as noted by Hochbaum and Shmoys[11]). Several probabilistic results on the *LPT* algorithm can be found in the literature (see, e.g., Coffman et al.[6]), obtained by using real values for the processing times. It is known in particular that if the processing times are drawn from the uniform distribution on [0, 1], both the absolute error $C_{\max}(LPT) - C^*_{\max}$ and the expected value of $C_{\max}(LPT) - L_0$ tend to 0 as $n \to \infty$. Although these results cannot be immediately extended to our distributions, the behavior of algorithm *BIN* for Class 1 seems to confirm them.

In order to obtain harder problems, we considered *per-*

fect packing instances, i.e., instances for which the optimal schedule has equal completion time on each processor. Given an integer value Q, we considered the interval $[0, nQ]$ and subdivided it into n subintervals through $m-1$ fixed values $Q, 2Q, \ldots, (m-1)Q$, plus $n-m$ distinct values uniformly random in $[1, nQ-1]$: the lengths of the n subintervals have been used for the processing times. Table III gives the results obtained for different values of Q. We see that the problems are generally very hard for algorithm *BIN*, except for very large values of n. Algorithm $B\&B$ easily solves the instances with $Q \leqslant 100$, while it cannot solve some small instances for larger values of Q.

We selected a data set of "medium" difficulty ($m = 10$, Class 3) for analyzing the computational behaviour of $B\&B$ when the magnitude of the processing times varies. To this end we generated the p_j values uniformly random in range $[R/2, R]$, with R growing from 20 to 2000. The results of Table IV show that the computing times grow with R for

Table III. Perfect Packing Instances, VAXstation 3100 Seconds: Average Time Over 10 Problems

		Q = 50		Q = 100		Q = 200		Q = 400	
m	n	BIN	B & B	BIN	B & B	BIN	B & B	BIN	B & B
3	10	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	25	0.04	0.01	0.13	0.01	0.46	0.01	0.50	0.01
	50	0.01	0.01	0.05	0.01	0.08	0.01	0.24	0.01
	100	0.01	0.01	0.01	0.01	0.02	0.01	0.17	0.01
	250	0.01	0.01	0.01	0.01	0.01	0.01	0.09	0.01
	500	0.01	0.01	0.01	0.01	0.01	0.01	0.15	0.01
	1000	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01
	2500	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.04
	5000	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07
	10000	0.14	0.14	0.13	0.14	0.13	0.14	0.13	0.14
5	10	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	25	0.13	0.01	0.18 (7)	0.01	3.04 (7)	0.02	2.73 (3)	0.25 (9)
	50	0.02	0.01	0.09 (8)	0.01	0.02 (8)	0.01	2.35 (5)	0.01
	100	0.01	0.01	0.05	0.01	0.16 (8)	0.01	1.85 (8)	0.01
	250	0.01	0.01	0.02	0.01	0.24 (9)	0.01	1.97 (9)	0.01
	500	0.01	0.01	0.01	0.01	0.01	0.01	0.10	0.02
	1000	0.02	0.02	0.02	0.01	0.02	0.02	0.02	0.02
	2500	0.04	0.05	0.05	0.05	0.04	0.05	0.04	0.04
	5000	0.09	0.09	0.09	0.09	0.09	0.09	0.08	0.09
	10000	0.18	0.19	0.17	0.18	0.17	0.19	0.17	0.18
10	25	0.02	0.02	0.12	0.02	0.09	0.04	0.18	0.06
	50	0.08 (9)	0.01	0.27 (8)	0.01	0.23 (1)	0.01 (9)	0.03 (1)	0.04 (7)
	100	0.05	0.01	0.31	0.01	0.99 (9)	0.01	3.21 (4)	0.01
	250	0.01	0.01	0.03	0.02	1.46 (9)	0.02	3.50 (6)	0.02
	500	0.02	0.02	0.02	0.02	0.03	0.03	9.85	0.04
	1000	0.02	0.03	0.03	0.03	0.03	0.04	0.28	0.06
	2500	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08
	5000	0.15	0.14	0.14	0.14	0.14	0.13	0.13	0.13
	10000	0.29	0.29	0.28	0.29	0.28	0.28	0.27	0.28
15	25	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	50	0.24 (9)	0.01	1.81 (5)	1.64	0.02 (1)	3.72 (7)	— (0)	5.02 (5)
	100	0.12	0.01	2.03 (7)	0.01	9.93 (2)	0.01	12.23 (1)	0.02
	250	0.02	0.02	0.65 (9)	0.02	3.18 (9)	0.02	7.04 (6)	0.03
	500	0.02	0.02	0.11	0.03	4.88 (9)	0.05	22.37 (8)	0.05
	1000	0.04	0.04	0.04	0.05	0.79	0.07	42.71	0.08
	2500	0.09	0.10	0.09	0.09	0.09	0.11	0.12	0.14
	5000	0.19	0.20	0.19	0.20	0.18	0.18	0.20	0.20
	10000	0.40	0.39	0.38	0.39	0.37	0.39	0.38	0.36

In brackets, number of solved problems, if less than 10.

Table IV. Algorithm B & B; p_j Uniformly Random in Range $[0.5R, R]$ VAXstation 3100 Seconds: Average Time over 10 Problems

m	n	$R = 20$	$R = 50$	$R = 100$	$R = 250$	$R = 500$	$R = 1000$	$R = 2000$
	25	0.40	3.41	1.98	2.28	3.15	2.73	7.91
	50	0.01	0.03	1.66	2.25	0.62	1.74	6.58
	100	0.01	0.01	0.01	0.01	0.01	0.02	0.12
	250	0.01	0.01	0.01	0.02	0.03	0.03	0.03
10	500	0.01	0.02	0.01	0.02	0.03	0.03	0.07
	1000	0.03	0.03	0.03	0.03	0.04	0.13	0.17
	2500	0.07	0.07	0.07	0.07	0.07	0.09	0.09
	5000	0.14	0.14	0.14	0.14	0.13	0.15	0.13
	10000	0.29	0.29	0.27	0.28	0.28	0.28	0.27

$n \leqslant 50$, while they are not affected by the value of R for $n \geqslant 100$.

Acknowledgments

This research was supported by Ministero dell'Univeristà e della Ricerca Scientifica e Tecnologica (MURST), Italy and by Consiglio Nazionale delle Ricerche (CNR), Italy.

References

1. J. Blazewicz, 1987. Selected Topics in Scheduling Theory, in S. Martello, G. Laporte, M. Minoux and C. Ribeiro, eds., *Surveys in Combinatorial Optimization, Annals of Discrete Mathematics 31*, 1–60.
2. M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan, 1973. Time Bounds for Selection, *Journal of Computer and System Sciences 7*, 448–461.
3. P. Bratley, M. Florian, and P. Robillard, 1975. Scheduling with Earliest Start and Due Date Constraints on Multiple Machines, *Naval Research Logistics Quarterly 22*, 165–173.
4. T.C.E. Cheng and C.C.S. Sin, 1990. A State-of-the-Art Review of Parallel-Machine Scheduling Research, *European Journal of Operational Research 47*, 271–292.
5. E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson, 1978. An Application of Bin-Packing to Multiprocessor Scheduling, *SIAM Journal on Computing 7*, 1–17.
6. E.G. Coffman, G.S. Lueker, and A.H.G. Rinnooy Kan, 1988. Asymptotic Methods in the Probabilistic Analysis of Sequencing and Packing Heuristics, *Management Science 34*, 266–290.
7. M. Fischetti and S. Martello, 1988. A Hybrid Algorithm for Finding the *k-th* Smallest of n Elements in $O(n)$ Time, in B. Simeone, P. Toth, G. Gallo, F. Maffioli, and S. Pallottino, eds., *FORTRAN Codes for Network Optimization, Annals of Operations Research 13*, 401–419.
8. M.R. Garey and D.S. Johnson, 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco.
9. R.L. Graham, 1966. Bounds for certain multiprocessing anomalies, *Bell System Technical Journal 45*, 1563–1581.
10. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey, *Annals of Discrete Mathematics 5*, 287–326.
11. D.S. Hochbaum and D.B. Shmoys, 1987. Using Dual Approximation Algorithms for Scheduling Problems: Practical and Theoretical Results, *Journal of ACM 34*, 144–162.
12. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, 1993. Sequencing and Scheduling: Algorithms and Complexity, in S.C. Graves et al., eds., *Handbooks in Operations Research and Management Science*, North-Holland, Amsterdam, 445–522.
13. S. Martello and P. Toth, 1984. Worst-case Analysis of Greedy Algorithms for the Subset-Sum Problem, *Mathematical Programming 28*, 198–205.
14. S. Martello and P. Toth, 1990. Lower Bounds and Reduction Procedures for the Bin Packing Problem, *Discrete Applied Mathematics 28*, 59–70.
15. S. Martello and P. Toth, 1990. *Knapsack Problems: Algorithms and Computer Implementations*, Wiley, Chichester.
16. R. McNaughton, 1959. Scheduling with Deadlines and Loss Function, *Management Science 6*, 1–12.