Discrete Applied Mathematics 42 (1993) 123-138
North-Holland

Algorithms for minimizing maximum lateness with unit length tasks and resource constraints

J. Błazewicz

Instytut Informatyki, Politechnika Poznańska, Poznań, Poland

W. Kubiak

Faculty of Management, University of Toronto, Toronto, Ont., Canada

S. Martello

Dipartimento di Informatica, Università di Torino, Torino, Italy

Received 1 June 1990
Revised 1 March 1991

Abstract

Błazewicz, J., W. Kubiak and S. Martello, Algorithms for minimizing maximum lateness with unit length tasks and resource constraints, Discrete Applied Mathematics 42 (1993) 123–138.

The problem we consider is that of scheduling n unit length tasks on identical processors in the presence of additional scarce resources. The objective is to minimize maximum lateness. It has been known for some time that the problem is NP-hard even for two processors and one resource type. In the present paper we show that the problem can be solved in $O(n \log n)$ time, even for an arbitrary number of resources if the instance of the problem has the saturation property (i.e., no resource unit is idle in an optimal schedule). For the more general problem without saturation, two heuristic algorithms and a branch and bound approach are proposed. The results of computational tests of the above methods are also reported.

1. Introduction

One of the most important problems arising in the context of deterministic scheduling is that of scheduling under resource constraints, where each task may also re-

Correspondence to: Professor S. Martello, DEIS-University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy.

quire, besides a processor, the use of additional scarce resources [3, 6]. Several papers have been devoted to the analysis of various aspects of the problem with the following criteria: schedule length [4, 6, 8, 9, 11–16], mean flow time [5] and maximum lateness [1, 2], respectively. Among these criteria the third has the strongest practical motivation, which follows from many applications of computer control systems [7]. Unfortunately, most of the scheduling subproblems for the last criterion have been proved to be NP-hard. In fact, in the case of nonpreemptive scheduling, a polynomial time optimization algorithm has been constructed for unit processing times and zero-one resource requirement concerning one resource type only [1]. All other nonpreemptive scheduling problems, including two processor, one resource or several resources and zero-one requirement cases, respectively, have been proved to be NP-hard. This paper is concerned with the algorithms for solving these problems. It appears that two processor nonpreemptive scheduling of unit length tasks to minimize maximum lateness is solvable in polynomial time, if all the resources are saturated. The saturation property, defined more formally later, generally means that no idle resource unit in an optimal schedule exists. This result, mainly of theoretical interest, is quite surprising if we take into account previous complexity results concerning three processors and schedule length criterion [9]. The last problem was somehow symmetrical to the two processor case when maximum lateness was minimized and the two problems usually were of the same complexity (cf. analysis carried on in [3]). However, this is not the case of the saturation property. The three processor schedule length minimization problem is NP-hard in both cases with saturated and nonsaturated resources, respectively [9][1]. On the other hand, the two processor maximum lateness minimization problem is NP-hard for nonsaturated resources only [2]; the saturated case being proved to be solvable in polynomial time in the present paper. (From the practical point of view we see that the last problem is as frequent as 3-partition.) For a nonsaturated version of the latter problem two heuristic algorithms are proposed and compared computationally with a branch and bound approach. Before presenting the results we will set up the subject more precisely.

We are given a set of *n tasks* $\mathcal{T}=\{T_1, T_2, \ldots, T_n\}$ to be processed on a set of two identical *processors* $\mathcal{P}=\{P_1, P_2\}$ with the use of certain amounts of *additional resources* $R_1, R_2, \ldots, R_s$ available in $m_1, m_2, \ldots, m_s$ units, respectively. To be processed, each task T_j requires an arbitrary processor and additional resources specified by the vector $[R(T_j)] = [R_1(T_j), R_2(T_j), \ldots, R_s(T_j)]$, where $R_k(T_j)$ is the amount of resource R_k required by task T_j. In the following we will assume that all the tasks must be processed in a nonpreemptive mode, i.e., once assigned to a processor, each task must be processed until completion without any break. For task T_j ($j=1, 2, \ldots, n$), a *processing time* $p(T_j)$ and a *due date* $d(T_j)$ are given. As is usually done in scheduling theorey we may assume that all $p(T_j)$, $d(T_j)$, m_k, $R_k(T_j)$ are nonnegative integers.

[1] Let us note, that the saturated one resource case is simply the famous 3-partition problem [9].

A *feasible schedule* is a function $f: \mathcal{T} \to Z^+$ which obeys the following conditions:

(a) For each integer u, $u \geq 0$,

$$|S_f(u)| \leq 2,$$

where $S_f(u) = \{T_j \in \mathcal{T}: f(T_j) \leq u < f(T_j) + p(T_j)\}$ is the set of tasks assigned to processors at time u.

(b) For each integer u, $u \geq 0$, and each k, $k = 1, 2, \ldots, s$,

$$\sum_{T_j \in S_f(u)} R_k(T_j) \leq m_k.$$

For a given schedule, let $C(T_j)$ denote the completion time of task T_j, i.e., $C(T_j) = f(T_j) + p(T_j)$. Let the schedule length be defined as $C_{\max} = \max_j \{C_j\}$. Then, we say that a schedule has the *saturation property* if

$$\sum_{j=1}^{n} p(T_j) R_k(T_j) = m_k C_{\max}, \quad k = 1, 2, \ldots, s.$$

It is easy to see that in a saturated schedule no resource unit is idle until $C_{\max}$ is reached.

The criterion we will use to evaluate schedules will be *maximum lateness* defined as follows:

$$L_{\max} = \max_j \{C(T_j) - d(T_j)\}.$$

The paper is organized in the following way. In Section 2 an $O(n \log n)$ algorithm for nonpreemptive scheduling of unit length tasks on two processors assuming saturated resources, is presented. Section 3 is concerned with heuristic procedures based on a greedy approach for an unsaturated case. Section 4 presents some basic ideas of a branch and bound algorithm. Finally, Section 5 summarizes computational experiments with both approaches performed on HP 9000/840.

2. Minimizing maximum lateness with saturated resources

In this section we will consider the problem of minimizing maximum lateness on two processors with unit length, nonpreemptable tasks on the assumption that an optimal schedule has saturated resources. Slightly modifying the notation presented in [6,10] this problem may be denoted as $P2 \mid \text{res} \ldots (\text{sat}), p_j = 1 \mid L_{\max}$. Firstly, a problem with one resource type will be solved in $O(n \log n)$ time (this can be denoted as $P2 \mid \text{res1} \ldots (\text{sat}), p_j = 1 \mid L_{\max}$). Then, a more general problem will be considered.

Observe that the number, say n^0, of tasks with zero resource requirement cannot be greater than the number, say n^1, of tasks requiring m_1 resource units. If $n^0 < n^1$, we add $n^1 - n^0$ dummy tasks with zero resource requirement and infinite due date. The following algorithm then finds a schedule for $P2 \mid \text{res1} \ldots (\text{sat}), p_j = 1 \mid L_{\max}$. (Obviously, this problem has a feasible saturated solution if the numbers of tasks with complementary resource requirements (i.e., summing up to m_1) are equal.)

Algorithm S.

1 Let $\{\bar{R}_1, \bar{R}_2, \ldots, \bar{R}_{\bar{n}}\}$ be the set of distinct $R_1(T_j)$ values, sorted so that $\bar{R}_i < \bar{R}_{i+1}$ $(i = 1, \ldots, \bar{n}-1)$;
for $i := 1$ **to** $\bar{n}$ **do** $Z_i := \{T_j\colon R_1(T_j) = \bar{R}_j\}$;
sort each set Z_j in nonincreasing order of the task due dates, and denote by T_{ij} the jth task on the list in set Z_i;
for $i := 1$ **to** $\lfloor \bar{n}/2 \rfloor$ **do** $Z_i' := \{(T_{ij}, T_{\bar{n}-i+1,j})\colon 1 \le j \le |Z_i|\}$
comment: $R_1(T_{ij}) + R_1(T_{\bar{n}-i+1,j}) = m_1$, for all i, j;
if $\bar{n}$ is odd **then** $Z_{\lceil \bar{n}/2 \rceil}' := \{(T_{\lceil \bar{n}/2 \rceil, 2j-1}, T_{\lceil \bar{n}/2 \rceil, 2j})\colon 1 \le j \le |Z_{\lceil \bar{n}/2 \rceil}|/2\}$;

2 (EDD rule) construct a schedule f' on one processor for the task set $\mathscr{T}' = \bigcup_{i=1}^{\lceil \bar{n}/2 \rceil} Z_i'$, where each pair of tasks $\{T', T''\}$ is treated as composite with due date $d^*(T) = \min\{d(T'), d(T'')\}$; composite tasks are assigned in EDD order;

3 convert the schedule f' into f on two processors replacing any composite T by two tasks T' and T'' (if any).

We now prove a theorem concerning the optimality of Algorithm S.

Theorem 2.1. *For any instance of the problem* $P2 \mid \text{res1} \,..\, (\text{sat}), p_j = 1 \mid L_{\max}$ *Algorithm* S *finds an optimal schedule.*

Proof. Let the set of tasks $\mathscr{T}$ be given. Because of the known properties of the EDD rule, schedule f' constructed in step 2 of Algorithm S is an optimal one for set $\mathscr{T}'$. Let the value of maximum lateness in this schedule be equal to $L'_{\max}$. We will show that schedule f constructed in step 3 is an optimal saturated schedule for task set $\mathscr{T}$.

To prove the theorem by contradiction, let us assume that there exists schedule f_0 for task set $\mathscr{T}$ having maximum lateness $L^0_{\max} < L'_{\max}$. Let o_t, $t = 1, 2, \ldots$ denote the pair of tasks assigned to processors at moment t in schedule f_0. Let O_i^0 be a set of all o_t containing at least one task belonging to Z_i, $i = 1, \ldots, \lceil \bar{n}/2 \rceil$. Let $O_i^0 = \{o_{t_{i,1}}, \ldots, o_{t_{i,k_i}}\}$, where $t_{i,1} < \cdots < t_{i,k_i}$, observe that, since both f^0 and f' are saturated schedules, we have $|O_i^0| = K_i = |Z_i'|$ $(i = 1, \ldots, \lceil \bar{n}/2 \rceil)$. Hence, let $Z_i' = \{Z_{i,1}, \ldots, Z_{i,k_i}\}$, with $d^*(Z_{i,1}) \le \cdots \le d^*(Z_{i,k_i})$. Consequently, we have $\min_{T \in o_{t_{i,k_i}}}\{d(T)\} \le \min_{T \in Z_{i,k_i}}\{d(T)\}$, so we can interchange tasks (or pairs of tasks) in O_i^0 so as to obtain a new schedule f_0', with $O_i^{0'} = \{o'_{t_{i,1}}, \ldots, o'_{t_{i,k_i}}\}$ such that $o'_{t_{i,k_i}} = Z_{i,k_i}$ and $L^{0'}_{\max} \le L^0_{\max}$. Repeating this procedure for $i = 1, \ldots, \lceil \bar{n}/2 \rceil$ we will get a schedule f_0'' for task set $\mathscr{T}'$ with $L^{0''}_{\max} < L'_{\max}$, which is a contradiction. Thus, f is an optimal saturated schedule. □

It is not hard to see that Algorithm S constructs schedules in $O(n \log n)$ time. Note, in fact, that sorted sets Z_i can easily be obtained by sorting the tasks according to nondecreasing resource requirements, breaking ties by nonincreasing due dates. As an example of its application let us consider the following instance of the problem: $n = 8$, $m_1 = 5$, $[R_1(T_j)] = [3, 4, 2, 1, 2, 3, 5, 0]$, $[d(T_j)] = [5, 2, 1, 3, 4, 2, 2, \infty]$.

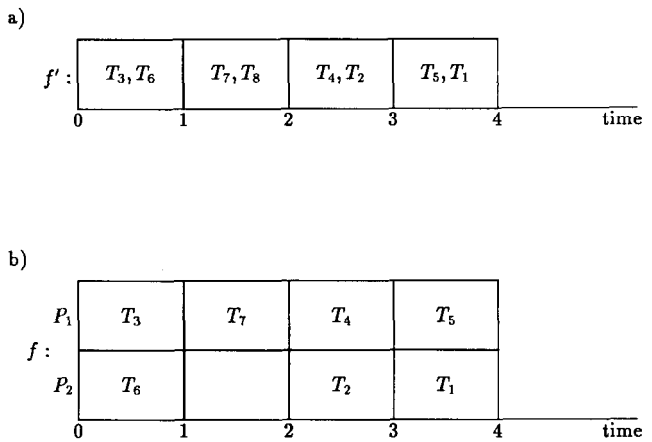

Fig. 1. Schedules to the example.

Performing step 1 of Algorithm S we get $\bar{n}=6$, $[\bar{R}_j]=[0,1,2,3,4,5]$, hence $Z_1'=\{(T_8,T_7)\}$, $Z_2'=\{(T_4,T_2)\}$, $Z_3'=\{(T_3,T_6),(T_5,T_1)\}$. The due dates for the corresponding composite tasks are $[d^*(T)]=[2,2,1,4]$. From there we get schedules f' and f as drawn in Fig. 1(a) and (b), respectively. Note that $L_{\max}=1$ for f and this is attained by T_2.

Now, let us consider a more general problem, $P2 \mid \text{res} \ldots (\text{sat}), p_j=1 \mid L_{\max}$. It is not hard to see that by slightly modifying the transformation given by Garey and Johnson [9] for the schedule length criterion, one may easily prove that this problem polynomially transforms to $P2 \mid \text{res1} \ldots (\text{sat}), p_j=1 \mid L_{\max}$ (the transformation preserves the saturation property). Thus, we have the following corollary.

Corollary 2.2. *Problem* $P2 \mid \text{res} \ldots (\text{sat}), p_j=1 \mid L_{\max}$ *is solvable in* $O(n \log n)$ *time.*

As already mentioned the saturation property is crucial here, since problem $P2 \mid \text{res} \ldots, p_j=1 \mid L_{\max}$ is NP-hard. In the next sections heuristic, and branch and bound approaches for this more general problem will be described.

3. Heuristic algorithms

3.1. The basic greedy approach

Let $[A_{jk}]$ be an $n \times n$ symmetric matrix defined by

$$A_{jk} = \begin{cases} 1, & \text{if } j \neq k \text{ and } R_l(T_j)+R_l(T_k) \leq m_l \text{ for } l=1,\ldots,s; \\ 0, & \text{otherwise.} \end{cases}$$

The approximate algorithms we present are based on the following greedy approach:

1 Determine A_{jk} $(j=1,\ldots,n;\ k=1,\ldots,n)$;
sort $\mathscr{T}$ so that

$$d(T_1) \le d(T_2) \le \cdots \le d(T_n), \tag{3.1}$$

breaking ties by $\max_{1\le l\le s}\{R_l(T_j)/m_l\}$, then by the second $\max_{1\le l\le s}\{R_l(T_j)/m_l\}$;

2 $t:=0$;
repeat
$t:=t+1$;
assign to P_1, in time slot t, the first nonassigned task T_j;
assign to P_2, in time slot t, the first nonassigned task T_k such that $A_{jk}=1$ (if any)
until all tasks are assigned.

This approach has two interesting properties. First, the assignments to processor P_1 are locally optimal in the following sense.

Theorem 3.1. *In the schedule produced by the greedy approach, no interchange between a task T_a assigned to P_1 and any other task T_b, $b>a$, can improve the value of $L_{\max}$.*

Proof. Let T_a be assigned to P_1 in time slot t, i.e., $L(T_a)=t-d(T_a)$ and $L(T_b)=t+\Delta-d(T_b)$, with $\Delta>0$. Then, from (3.1), the interchange would produce $L'(T_a)=t+\Delta-d(T_a)\ge\max\{L(T_a),L(T_b)\}$. □

The second property of the greedy approach is that the assignments producing *nonidle slots* (i.e., time slots in which both processors perform a task) are pairwise optimal, in the following sense.

Theorem 3.2. *In the schedule produced by the greedy approach no single interchange involving only nonidle slots can improve the value of $L_{\max}$.*

Proof. Let T_a be assigned to P_1 and T_b to P_2 in time slot t, T_c to P_1 and T_d to P_2 in time slot $t+\Delta$, $\Delta>0$. From Theorem 3.1, the assignment of T_a is locally optimal, so we consider the possibility of interchanging T_b and T_q ($q=c$ or d). If $d(T_b)>d(T_q)$, the interchange is impossible, since due to the greedy approach, T_q cannot be paired with T_a. On the other hand, if $d(T_b)=d(T_q)$ the operation has no effect. Hence, assume $d(T_b)<d(T_q)$. The interchange could improve the value of $L_{\max}$ only if $L(T_q)=L_{\max}$. The lateness of T_b would become $t+\Delta-d(T_b)>t+\Delta-d(T_q)=L_{\max}$. □

3.2. On-line interchanges and algorithm H1

The algorithm we consider in the present section is a modification of the greedy approach, in which, in a first phase after the assignment of a task to P_1, only tasks having the same due date are considered for possible assignment to P_2. When all the tasks with equal due date have been assigned, interchanges are attempted between them and previously assigned tasks, in order to improve the current solution. The algorithm works as follows.

Procedure H1.
Determine A_{jk} $(j=1,\dots,n;\ k=1,\dots,n)$;
subdivide the tasks, sorted according to (3.1), into consecutive *blocks* $B_1,\dots,B_q$ of tasks having the same due date;
for $l:=1$ **to** q **do**
begin
repeat
assign to P_1 the first nonassigned task $T_j \in B_l$;
assign to P_2 the first nonassigned task $T_k \in B_l$ such that $A_{jk}=1$ (if no such T_k exists leave the current time slot idle for P_2);
until all $T_j \in B_l$ are assigned;
interchange tasks of B_l with tasks of B_{l-1} in order to improve the current value of $L_{\max}$;
for each $T_j \in B_l$ assigned to P_1 with idle time for P_2 **do**
assign to P_2 the first task $T_k \in B_{l+1} \cup \cdots \cup B_q$ such that $A_{jk}=1$ (if any)
end

The interchanges are determined as follows. Let d_{l-1} and d_l denote the due dates of the tasks in B_{l-1} and B_l, respectively. Let I_{l-1} be the set of the tasks $T_j \in B_{l-1}$ which are currently paired with another task of B_{l-1}. Let $\tilde{I}_l$ be the set of those tasks $T_k \in B_l$ which are currently not paired (according to Theorem 3.2, "single" interchanges between nonidle time slots do not improve the solution). Determine a subset $E_l \subseteq I_l$, of even cardinality, containing pairs (T_{k_1}, T_{k_2}) for each of which there is a different nonidle time slot assigned, say, to T_{j_1} and T_{j_2} $(T_{j_1}, T_{j_2} \in I_{l-1})$, such that T_{j_1} can be paired with T_{k_1} and T_{j_2} with T_{k_2}. Assuming, without loss of generality, that the tasks in $B_{l-1} \setminus I_{l-1}$ precede those in I_{l-1}, Fig. 2(a) shows the current schedule, and Fig. 2(b) an interchange we can perform. The operation improves the current value of $C_{\max}$. In order to establish whether the current value of $L_{\max}$ is also improved, let c denote the completion time for B_{l-1} and r the number of time slots used for $B_l \setminus E_l$. Then the current maximum lateness for B_{l-1} and B_l is, respectively,

$$L(B_{l-1}) = c - d_{l-1}, \qquad L(B_l) = c + |E_l| + r - d_l.$$

If $L_{\max} = L(B_{l-1})$, then no improvement can be obtained, since the interchange

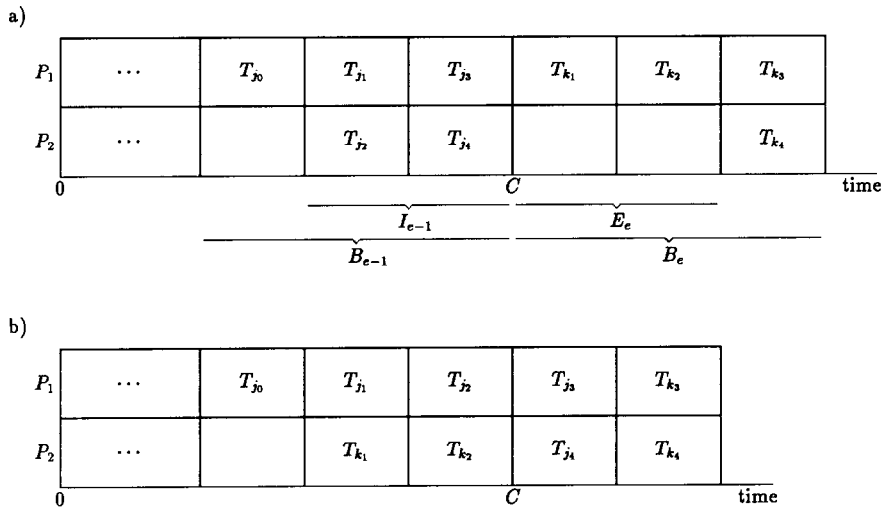

Fig. 2.

delays tasks of B_{l-1}. If, instead, $L_{\max} = L(B_l) > L(B_{l-1})$ then, after the interchange, the maximum lateness would be

$$L'(B_{l-1}) = c + |E_l|/2 - d_{l-1},$$

$$L'(B_l) = c + |E_l|/2 + r - d_l,$$

so the operation is profitable if

$$\max\{(-d_{l-1}), (r-d_l)\} \leq |E_l|/2 + r - d_l. \tag{3.2}$$

(When in (3.2), the = sign holds, the interchange does not improve $L_{\max}$ but $C_{\max}$ decreases.)

Example 3.3. Consider the situation of Fig. 3 with $l=2$, $d(T_{ji}) = d_1 = 3$, $d(T_{ki}) = d_2 = 4$. We have $L(B_1) = 0$, $L(B_2) = 3$.

(a) If $E_2 = \{T_{k_1}, T_{k_2}\}$, i.e., $r=2$, pairing T_{j_1} with T_{k_1} and T_{j_2} with T_{k_2} as in Fig. 2(b) is profitable since, from (3.2), $\max\{(-d_1), (r-d_2)\} = r - d_2 = -2 < -1$.

(b) If $E_2 = B_2$, i.e., $r=0$, interchanging the tasks as in Fig. 4 is profitable since $\max\{(-d_1), (r-d_2)\} = -d_1 = -3 < -2$.

(c) If we had $d_2 = \delta$, then $E_2 = \{T_{k_1}, T_{k_2}\}$ would not change the local maximum lateness, while $E_2 = B_2$ would increase it.

The initialization phase of H1 takes $O(n^2 s)$ time to determine $[A_{jk}]$. Apart from the interchange phase, the time complexity of the iterative part is $O(n^2)$, since $O(n)$ tasks are assigned to P_1 and, for each of them, $O(n)$ tasks are considered for possible pairing. The time required by the interchange phase depends on the method used for determining E_l. This can be done, in a greedy way, as follows:

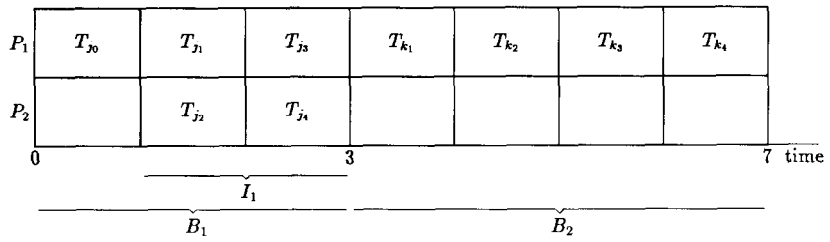

Fig. 3.

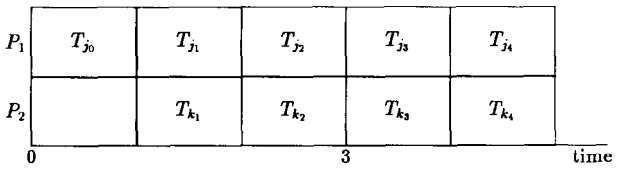

Fig. 4.

$E := \emptyset$;
for each nonidle time slot assigned, say, to $T_{j_1}, T_{j_2} \in I_{l-1}$ **do**
  if $\exists T_{k_1} \in \tilde{I}_l$ such that $A_{j,k_1} = 1$ **then**
    if $\exists T_{k_2} \in \tilde{I}_l \setminus \{T_{k_1}\}$ such that $A_{j_2,k_2} = 1$ **then**
      begin
        $\tilde{I}_l := \tilde{I}_l \setminus \{T_{k_1}, T_{k_2}\}$;
        $E_l := E_l \cup \{(T_{k_1}, T_{k_2})\}$
      end

In this way $O(n)$ operations are required for each time slot, so the overall time complexity of H1 is $O(n^2 s)$.

It is worth noting that the above interchange technique can easily be generalized to interchanges between the current block B_l and any block B_q, $q<1$. Computational experiments, however, indicated that such an extension produces very limited improvements with considerably higher average computing times.

3.3. Off-line interchanges and algorithm H2

The second algorithm we consider operates on the final schedule produced by H1, in an attempt to reduce the final value of $L_{\max}$ through an interchange technique similar to that used in the previous section, but independent of the subdivision into blocks.

Interchanges of this kind can be computationally very expensive, requiring rearrangement of a large portion of the final schedule. We adopted the following strategy, which produces a comparatively simple rearrangement.

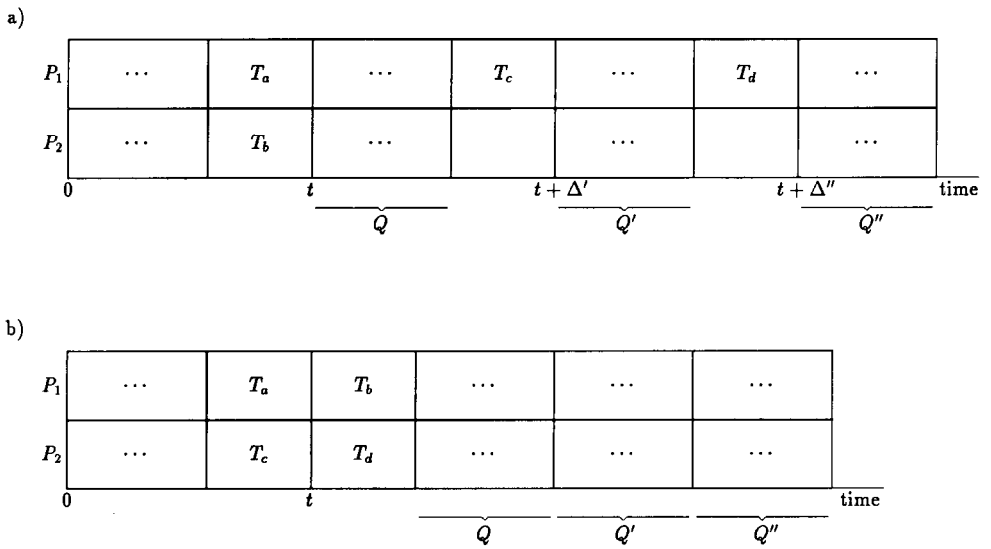

Fig. 5.

Let T_a and T_b be assigned to P_1 and P_2, respectively, in time slot t. Let T_c and T_d be nonpaired tasks assigned to P_1 in time slots $t+\Delta'$ and $t+\Delta''$, respectively, with $\Delta''>\Delta'>0$. Let Q, Q' and Q'' be the sets of tasks assigned to the time slots between t and $t+\Delta'$, between $t+\Delta'$ and $t+\Delta''$, and following $t+\Delta''$, respectively (Fig. 5(a)). Assuming $A_{ac}=A_{bd}=1$, Fig. 5(b) shows an interchange which decreases the value of $C_{\max}$ by one unit. It also decreases the value of $L_{\max}$ if $L_{\max}$ is determined by tasks in $\{T_c\}\cup\{T_d\}\cup Q''$, i.e., there is no task t outside $\{T_c, T_d\}\cup Q''$ with $L(t)=L_{\max}$. It cannot be performed, instead, if the lateness of some task in $\{T_b\}\cup Q$ has value $L_{\max}$. The resulting algorithm is as follows.

Procedure H2.
Execute procedure H1;
for each nonidle time slot t (with T_a assigned to P_1 and T_b to P_2) **do**
  if $d(T_b)<L_{\max}$ **then**
    begin
      $\tau:=t+1$;
      hope:="yes";
      found:=0;
      while $\tau\leq C_{\max}$ and *hope*="yes" and *found*<2 **do**
        begin
          if *found*=0 **then**
            begin
              let L=maximum lateness of a task in time slot τ;
              if $L=L_{\max}$ **then** *hope*:="no";

```
            if time slot τ is idle (with T_c assigned to P_1)
              then if A_ac = 1 or A_bc = 1 then found := 1
          end
        else
          if time slot τ is idle (with T_d assigned to P_1)
            then if A_ac = A_bd = 1 or A_bc = A_ad = 1
              then found := 2;
          τ := τ + 1
        end
      if found = 2 then
        if A_ac = A_bd = 1 then rearrange the schedule as in Fig. 5(b)
        else
          rearrange the schedule as in Fig. 5(b) with T_c and T_d inter-
          changed
    end
```

The off-line interchanges clearly require $O(n^2)$ time. Hence the overall time complexity of H2 is $O(n^2 s)$. The quality of the solutions obtained by algorithm H2 has been evaluated experimentally. The results are presented in Section 5.

4. Branch and bound algorithm

We implemented a depth first branch and bound algorithm for the exact solution of the problem. The execution starts by performing Procedure H2 of the previous section, thus determining an initial feasible solution of value, say, $L^0_{\max}$. The enumerative phase is then applied.

4.1. Branching scheme

The algorithm generates decision nodes by assigning a task to one of the two processors. Since the tasks are considered in the EDD order we have, by the same argument as used in the proof of Theorem 3.1, all the assignments to P_1 optimal, i.e., they need no branching on brother nodes. The node corresponding to an assignment to P_1 (of task say, T_j) generates a son node for each nonassigned task T_k such that $A_{jk} = 1$. These are explored according to the EDD order. The branching scheme is shown in Fig. 6 (P_a on a branch indicates assignment of the next feasible task to processor P_a).

At each node we determine a lower bound on the best solution obtainable from the descending nodes. If this is less than the value $L^0_{\max}$ of the best solution so far, the next son node is generated. Otherwise we backtrack to the first ancestor node (generated by an assignment to P_1) whose associated bound is less than $L^0_{\max}$, and start a new branching process. The execution terminates when a backtracking

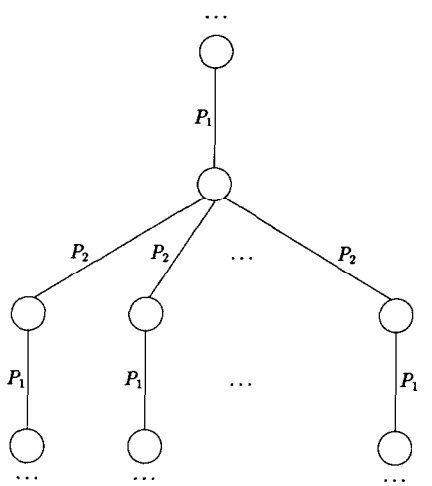

Fig. 6.

reaches the root node, or when a feasible solution is found whose value equals that of the lower bound computed for the root node.

4.2. *Lower bounds*

At each decision node N, let $\mathscr{T}_N$ be the set of those tasks which are currently nonassigned, and $D_N = \{d_1, d_2, \ldots, d_\delta\}$ ($\delta = |D_N|$) the set of different due dates of tasks in $\mathscr{T}_N$. For $k = 1, \ldots, \delta$, let

$$\mathscr{T}_N(k) = \{T_j \in \mathscr{T}_N : d(T_j) \leq d_k\}.$$

Let us assume, for the time being, that node N has been generated by assigning a task to P_2 in time slot C_N (current completion time). We relax the current problem by assuming that, for a given k, all tasks in $\mathscr{T}_N(k)$ have due date d_k. For each k ($k = 1, \ldots, \delta$) different lower bounds on the solution value obtainable from node N can then be computed as follows.

A first bound immediately derives from the cardinality of $\mathscr{T}_N(k)$:

$$L'_N(k) = C_N + \left\lceil \frac{|\mathscr{T}_N(k)|}{2} \right\rceil - d_k.$$

A second lower bound can be obtained by recalling that, in each time slot, at most m_i units of resource R_l ($l = 1, \ldots, s$) are available:

$$L''_N(k) = C_N + \left\lceil \max_{1 \leq l \leq s} \frac{\sum_{T_j \in \mathscr{T}_N(k)} R_l(T_j)}{m_l} \right\rceil - d_k.$$

By noting that, for each resource R_l, all tasks requiring more than $m_l/2$ units of

resource R_l must be assigned to different processors, we have a third lower bound:

$$L'''_N(k) = C_N + \max_{1 \le l \le s} \{|\{T_j \in \mathscr{T}_N(k): R_l(T_j) > m_l/2\}|\} - d_k.$$

None of the three bounds dominates the others as shown by the following

Example 4.1. Let $n=7$, $s=2$, $[d_j]=[1,1,1,1,1,2,2]$, $[R_1(T_j)]=[4,4,4,4,4,8,8]$, $[R_2(T_j)]=[5,5,5,5,5,8,8]$. Consider the root node (i.e., $\mathscr{T}_N=\mathscr{T}$, $D_N=\{d_1,d_2\}=\{1,2\}$, $C_N=0$) and $k=2$ (i.e., $\mathscr{T}_N(k)=(T_1,\ldots,T_7)$).

(a) If $m_1=m_2=15$ we have $L'_N(k)=2$ (optimal solution value), $L''_N(k)=1$, $L'''_N(k)=0$.

(b) If $m_1=m_2=10$ we have $L'_N(k)=2$, $L''_N(k)=3$ (optimal solution value), $L'''_N(k)=0$.

(c) If $m_1=m_2=9$ we have $L'_N(k)=2$, $L''_N(k)=3$, $L'''_N(k)=5$ (optimal solution value).

An overall lower bound for node N is thus

$$L_N = \max_{1 \le k \le \delta} \{\max(L'_N(k), L''_N(k), L'''_N(k))\}.$$

Determining L_N apparently requires $O(n^2 s)$ time. Note however that, since the tasks are sorted, given $T_N(k)=\{T_a, T_{a+1}, \ldots, T_b\}$, we can obtain $T_N(k+1)$ by adding a set of consecutive tasks $T_{b+1}, T_{b+2}, \ldots$. Hence it is not difficult to implement L_N through a series of updatings for $k=1,\ldots,\delta$, so as to have the same time complexity required for computing $L_N(\delta)$, i.e., $O(ns)$.We have obtained L_N by assuming that decision node N derives from an assignment to P_2. If it derives from an assignment to P_1, the formulae above must be modified so as to take into account the possibility that one of the tasks in $\mathscr{T}_N(k)$ is assigned to the current time slot C_N. We do not give the details, which are straightforward.

4.3. Dominance relations

The number of decision node explorations can be decreased by eliminating nodes which are dominated by previously explored nodes. One such situation arises when, after a backtracking on a taks T_j, an identical task T_k (with $d(T_k)=d(T_j)$, $R_l(T_k)=R_l(T_j)$ for $l=1,\ldots,s$) is assigned to the same time slot.

Also when backtracking on the assignment to P_2 of the last task compatible with that assigned to P_1, the decision node generated by leaving the current time slot idle for P_2 is clearly dominated by any of its brother nodes.

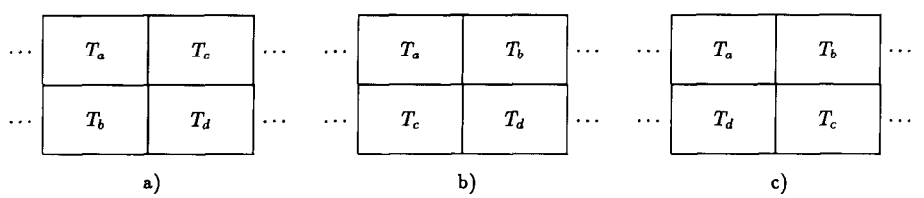

Fig. 7. Schedule (a) dominates schedules (b) and (c).

Table 1: HP 9000/840 seconds. Average times (average errors) over 10 problem instances

n	$s=1$			$s=2$			$s=5$			$s=10$		
	Exact	Approximate	(error)	Exact	Approximate	(error)	Exact	Approximate	(error)	Exact	Approximate	(error)
10	0.005	0.004	(0.0200)	0.005	0.004	(0.0365)	0.013	0.005	(0.0422)	0.011	0.007	(0.0000)
25	0.010	0.010	(0.0000)	0.020	0.016	(0.0000)	0.114	0.020	(0.0744)	0.065	0.033	(0.0202)
50	0.547	0.031	(0.0032)	0.173	0.051	(0.0059)	0.365	0.074	(0.0366)	0.420	0.095	(0.0287)
100	0.171	0.101	(0.0035)	0.183	0.173	(0.0000)	52.791	0.253	(0.0171)	2.440	0.314	(0.0424)
250	0.577	0.563	(0.0000)	-	0.969	(0.0033)	-	1.367	(0.0072)	71.007	1.751	(0.0220)
500	2.231	2.177	(0.0000)	-	3.536	(0.0021)	-	5.385	(0.0051)	-	6.790	(0.0253)
1000	8.423	8.130	(0.0000)	-	13.474	(0.0004)	-	19.963	(0.0031)	-	25.000	(0.0061)

Finally, dominances arise when the same set of four tasks (say T_a, T_b, T_c, T_d with $d(T_a) \leq d(T_b) \leq d(T_c) \leq d(T_d)$) is assigned, in different order, to the same pair of consecutive time slots. No branching being performed on the assignments to P_1, this can happen in the three ways shown in Fig. 7. It immediately follows from Theorem 3.2 that the first schedule produced by the algorithm ((a) in the figure) dominates the following ones, (b) and (c).

5. Computational experiments

The branch and bound algorithm of the previous section and the approximate algorithm H2 of Section 4 have been implemented in Fortran IV and run on an HP 9000/840. Test problems have been obtained by uniformly randomly generating, for different values of n, values $d(T_j)$ in the range $(1, n+1)$, values m_1 in the range $(1, 100)$ and values $R_l(T_j)$ in the range $(0, m_l)$. For each pair (n, s), with $s = 1, 2, 5, 10$, ten problems have been generated. The entries in Table 1 give the average running times expressed in CPU seconds and, in brackets, an evaluation of the average errors produced by H2. For the cases in which the branch and bound algorithm could exactly solve the problem (producing the optimal $L^*_{\max}$ and a value $C^*_{\max}$), the error was evaluated as $(L(\mathrm{H2}) - L^*_{\max})/C^*_{\max}$, where $L(\mathrm{H2})$ denotes the maximum lateness produced by H2. For the case in which the exact solution could not be obtained, the error was evaluated as $(L(\mathrm{H2}) - L^0)/C(\mathrm{H2})$, where L^0 is the value of the lower bound at the root node of the branch decision tree and $C(\mathrm{H2})$ the completion time produced by H2. (The choice of dividing the absolute error by the completion time, instead of the optimal lateness is due to the possibility of having zero (or even negative) optimal lateness.)

The table shows that for problems with one resource the exact solution can be determined efficiently, also for large values of n. Higher values of s clearly increase the difficulty. The approximations obtained with H2 were always satisfactory. Also the absolute error was always very small, never exceeding few units: the average absolute error was between 0 and 0.2, and independent of s or n. Also for the approximate algorithm the running times increase with the value of s, but of course, not as dramatically as for the branch and bound algorithm. The variance was moderate, with the maximum running time never exceeding the average time by more than 20 percent.

Acknowledgement

This work was supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica (Italy) and by Ministerstwo Edukacji Narodowej (Poland). Thanks are due to Guido Ognibeni for his assistance with programming.

References

[1] J. Błazewicz, Simple algorithms for multiprocessor scheduling to meet deadlines, Inform. Process. Lett. 6 (1977) 162–164.
[2] J. Błazewicz, J. Barcelo, W. Kubiak and H. Rock, Scheduling tasks on two processors with deadlines and additional resources, European J. Oper. Res. 26 (1986) 364–370.
[3] J. Błazewicz, W. Cellary, R. Słowinski and J. Weglarz, Scheduling under resource constraints: deterministic models, in: Annals of Operations Research 7 (Baltzer, Basel, 1986).
[4] J. Błazewicz and K. Ecker, A linear time algorithm for restricted bin packing and scheduling problems, Oper. Res. Lett. 2 (1983) 80–83.
[5] J. Błazewicz, W. Kubiak and J. Szwarcfiter, Minimizing mean flow time under resource constraints, Acta Inform. 24 (1987) 513–524.
[6] J. Błazewicz, J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Appl. Math. 5 (1983) 11–24.
[7] E.G. Coffman Jr, ed., Computer & Job/Shop Scheduling Theory (Wiley, New York, 1976).
[8] D. de Werra, Preemptive scheduling, linear programming and network flows, SIAM J. Algebraic Discrete Math. 5 (1984) 11–20.
[9] M.R. Garey and D.S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. Comput. 4 (1975) 397–411.
[10] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, in: Annals of Discrete Mathematics 5 (North-Holland, Amsterdam, 1979) 287–326.
[11] J.Y-T. Leung, Bounds on list scheduling of UET tasks with restricted resource constraints, Inform. Process. Lett. 9 (1979) 167–170.
[12] E.L. Lloyd, Coffman-Graham scheduling of UET task systems with 0-1 resources, Inform. Process. Lett. 12 (1981) 40–45.
[13] E.L. Lloyd, Concurrent task systems, Oper. Res. 29 (1981) 189–201.
[14] R. Möhring and F.J. Rademacher, The order theoretic approach to scheduling: Deterministic case, in: R. Słowinski and J. Weglarz, eds., Advances in Project Scheduling (Elsevier, Amsterdam, 1989) 29–66.
[15] H. Röck, Some new results in flow shop scheduling, Z. Oper. Res. 28 (1984) 1–16.
[16] R. Słowinski, Scheduling preemptible tasks on unrelated processors with additional resources to minimize schedule length, in: G. Bracchi and P.C. Lockemann, eds., Lecture Notes in Computer Science 65 (Springer, New York, 1978) 536–547.
[17] J. Weglarz, J. Błazewicz, W. Cellary and R. Słowinski, An automatic revised simplex method for constrained resource network scheduling, ACM Trans. Math. Software 3 (1977) 295–300.