ELSEVIER

European Journal of Operational Research 120 (2000) 277–288

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/orms

Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan

Giuseppe Lancia *

Dip. Elettronica e Informatica, University of Padova, 35100 Padova, Italy

Received 1 October 1997; accepted 1 April 1998

Abstract

In this paper we deal with the problem of assigning a set of n jobs, with release dates and tails, to either one of two unrelated parallel machines and scheduling each machine so that the makespan is minimized. This problem will be denoted by $R2|r_i, q_i|C_{\max}$. The model generalizes the problem on one machine $1|r_i, q_i|C_{\max}$, for which a very efficient algorithm exists. In this paper we describe a Branch and Bound procedure for solving the two machine problem which is partly based on Carlier's algorithm for the $1|r_i, q_i|C_{\max}$. An $O(n \log n)$ heuristic procedure for generating feasible solutions is given. Computational results are reported.

Keywords: Scheduling theory; Branch and bound; Parallel machines; Makespan

1. Introduction

Real-life planning of production consists of various steps, among which assigning the machines to the jobs, deciding the size of batches on each machine, and scheduling the starting times of the jobs. Although all these aspects are equally important and should all be considered in the same model in order to decide the best plan, there are few works in the literature which address the general problem as stated above. Instead, most works study the stand alone problems of optimization of assignment (see, among others, [6,1]), or batch-sizing (e.g. [4]), or scheduling (see for a survey [7,8,10]) *after* the assignment has been decided. A typical example of this last class is the Job Shop scheduling problem.

In general, parallel machines problems require at least two types of decisions: assignment and scheduling (i.e. timing). In this paper we address a problem in which two unrelated parallel machines, denoted by M_1 and M_2, are available to process a set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of n jobs. Both the assignment of jobs to machines and the schedule have to be determined.

For each job J, there is a *release date* r_J, on which it becomes available for processing and a *tail* q_J, representing a period of time that the job has to spend on an infinite-capacity machine after its processing on either machine M_1 or M_2. The execution of job J on machine M_i requires a *pro-*

* E-mail: lancia@dei.unipd.it

0377-2217/00/$ – see front matter
PII: S0377-2217(99)00156-3

cessing time p_{iJ}, which we assume to be a positive integer.

We call *assignment* a binary n-dimensional vector $\mathbf{x}$ deciding for each job the machine allocated to its execution (i.e. $x_J = 1$ iff J is processed on machine M_1). Once an assignment is given, we want to determine starting times $\mathbf{t} = (t_{J_1}, \ldots, t_{J_n})$ for all the jobs, in such a way that: (i) no job starts to be processed before its release date, (ii) each machine is working on at most one job at a time, and (iii) preemption is not allowed. Any such values for the starting times are called a feasible *schedule*.

If $\mathbf{x}$ is an assignment and $\mathbf{t}$ is a corresponding feasible schedule, we call the pair $(\mathbf{x}, \mathbf{t})$ a *plan*. A plan $(\mathbf{x}, \mathbf{t})$ determines for each job J a *completion time* $C_J := t_J + p_{1J}x_J + p_{2J}(1 - x_J) + q_J$. The cost of this plan is the maximum completion time, or *makespan*, $C_{\max} := \max_{J \in \mathscr{J}} C_J$. The problem consists in finding a plan for which the makespan is minimized. Using a notation which is standard in scheduling (see e.g. [8]), the problem is denoted as $R2|r_i, q_i|C_{\max}$.

The problem can alternatively be stated as a lateness minimization problem when, instead of tails, we have for each job J a *due date* d_J, specifying the time when it should be completed. Then, for a plan $(\mathbf{x}, \mathbf{t})$ we define the *lateness* of each job J as $L_J := t_J + p_{1J}x_J + p_{2J}(1 - x_J) - d_J$. The objective function is to find a plan which minimizes the maximum lateness. The equivalence of the two formulations is evident whenever we set a correspondence between due dates and tails through the relations $q_J = D - d_J$ (here, $D := \max_{J \in \mathscr{J}} d_J$ if we are passing from a lateness to a makespan minimization and $D := \max_{J \in \mathscr{J}} q_J$ otherwise).

The problem $R2|r_i, q_i|C_{\max}$ is strongly NP-Hard, since it is a generalization of the strongly NP-Hard one machine problem $1|r_i, q_i|C_{\max}$ (denoted by OMP in the sequel). In fact, the problem on one machine can be modeled as a problem on two machines in which all the jobs have an infinite (i.e. suitably large) processing time on machine M_2.

Although the OMP is strongly NP-hard [5], there exist very efficient branch and bound procedures for its solution [2,9,11]. In this paper, taking as a starting point the OMP algorithm by Carlier [2] we devise a branch and bound strategy for the solution of $R2|r_i, q_i|C_{\max}$. As Carlier's procedure is strongly based on a heuristic algorithm, due to Schrage, which schedules the jobs according to the Longest Tail Heuristic (LTH) dispatching rule, we will show how to generalize this algorithm so that not only it schedules the jobs but also determines an assignment of the jobs to the machines. This procedure is fully described in Section 4.

The remainder of the paper is organized as follows. In Section 2 we give a top-level description of the branch and bound procedure. Section 3 introduces the heuristic used for generating feasible solutions and provides an example. In Section 4 we describe valid lower bounds to the problem, while in Section 5 we discuss the branching rules used in the branch and bound. Section 6 lists some computational results and a few final remarks.

2. Overview

We start by noting that, given an assignment $\mathbf{x}$, finding the best schedule is fundamentally a sequencing problem. In fact, each schedule determines a permutation $\pi = (\pi(1), \ldots, \pi(n))$ of the jobs indexes such that for $i = 1, \ldots, n-1$, job $J_{\pi(i)}$ is scheduled before $J_{\pi(i+1)}$. Conversely, since the objective function is nondecreasing in the completion times, given a permutation π we can schedule the jobs in an optimal way for π by having each job $J_{\pi(i)}$ started on its machine as soon as all the jobs $J \in \{J_{\pi(1)}, \ldots, J_{\pi(i-1)}\}$ with $x_J = x_{J_{\pi(i)}}$ are completed. In view of this fact, we can consider as feasible solutions to the problem the pairs $(\mathbf{x}, \pi)$, where $\mathbf{x}$ specifies the assignment and π gives the execution sequence. Under this definition, adopted for simplicity of notation, different permutations can give rise to the same schedule. Conceptually, this is not a problem, since they are just different solutions with the same objective function value. Anyway, the algorithm is such that it never regards as distinct two permutations if they contain identical sub-permutations of the jobs on machine M_1 and M_2.

We propose the solution of $R2|r_i, q_i|C_{\max}$ by a branch and bound strategy. Let $\mathscr{P}^0$ be the original

problem, associated to the root of the search tree. By the previous remark, the set of solutions of $\mathscr{P}^0$ is given by $\{0,1\}^n \times S_n$, where S_n is the set of all permutations of $\mathscr{J}$. In general, the set of feasible solutions of a problem $\mathscr{P}$ at any node of the branch and bound tree, is $B(\mathscr{P}) \times S'(\mathscr{P})$, with $B(\mathscr{P}) \subseteq \{0,1\}^n$ and $S'(\mathscr{P}) \subseteq S_n$.

The branching rule used to partition the set of feasible solutions associated to a node of the search tree will be of two possible types, which we call *assignment branching* and *sequence branching*. As expected, the assignment branching fixes for one or more jobs the machine on which they must be processed, while the sequence branching constrains the order in which the jobs can appear in the sequence on the machines. The two branching rules are described in detail in Section 5.

The second major ingredient of a branch and bound procedure is the lower bound $\mathrm{lb}(\mathscr{P})$, computed at each node. The bound we use is given by the linear programming relaxation of a partitioning problem, solved via knapsack techniques, and described in Section 4.

Finally, we give a heuristic procedure to obtain feasible solutions. This procedure, described in Section 3, is greedy but returns near optimal solutions most of the times.

To summarize, the branch and bound works as follows. We keep a list of open problems, ordered by increasing value of lower bounds. Let U be the value of the current best solution. At each step we remove from the list the first element, say problem $\mathscr{P}$, solve it with the heuristic procedure and possibly update the incumbent U. Then, if the solution found for $\mathscr{P}$ is not optimal, we try to apply the sequence branching. If this is not possible, we perform an assignment branching. Then we fathom all problems whose lower bound is not smaller than U. We iterate until the list of problems is empty.

3. A heuristic procedure

In this section we describe an $O(n \log n)$ heuristic procedure for generating a feasible solution to $R2|r_i, q_i|C_{\max}$. We denote this solution as $h_J \in \{0,1\}$ (the assignment) and τ_J (the schedule).

For the sake of generality, we assume that two (possibly empty) disjoint subsets of jobs $\mathscr{J}_1, \mathscr{J}_2 \subseteq \mathscr{J}$ can be specified, corresponding to jobs that have to be assigned to machine M_1 and machine M_2, respectively. This feature will be useful in those stages of the branch and bound procedure, to be described later, when the values of some of the x_J variables have been fixed.

If $k \in \{1,2\}$ indicates a machine, let us denote by $\bar{k}$ the other one. Let $\mathscr{T}$ be the set of jobs still to be scheduled, so that at the beginning $\mathscr{T} := \mathscr{J}$. The heuristic procedure keeps at each step two counters, z_1 and z_2, representing the time when machine M_1 and M_2 are available for processing their next job, and two sets $\mathscr{A}_k \subseteq \mathscr{T} \setminus \mathscr{J}_{\bar{k}}$, for $k = 1,2$, of those jobs that can be processed on the corresponding machine M_k, at time z_k (i.e. $r_J \leqslant z_k \quad \forall J \in \mathscr{A}_k$). At the beginning we set $z_k := \min\{r_J \mid J \in \mathscr{T} \setminus \mathscr{J}_{\bar{k}}\}$ for $k = 1,2$.

The general step is as follows: for $k = 1,2$, let $I_k \in \mathscr{A}_k$ be a job such that $q_{I_k} = \max_{J \in \mathscr{A}_k} q_J$; then we will schedule I_1 on machine M_1 if $z_1 + p_{1I_1} + q_{I_1} \leqslant z_2 + p_{2I_2} + q_{I_2}$ and I_2 on machine M_2 otherwise. The idea is to choose for each machine the most critical (LTH rule) among the available jobs, then compare these two jobs and schedule the one that gives the smallest contribution to the objective function. Note that it is possible for the same job to be selected for scheduling on both machines (i.e. $I_1 = I_2$).

Suppose we schedule job I_k on machine M_k; then we set $\tau_{I_k} := z_k$, $h_{I_k} := 2 - k$, remove I_k from further consideration by setting $\mathscr{T} := \mathscr{T} \setminus \{I_k\}$ and update the situation as

$$z_k := \max\{z_k + p_{kI_k}, \min\{r_J \mid J \in \mathscr{T} \setminus \mathscr{J}_{\bar{k}}\}\},$$

$$A_k := \{J \in \mathscr{T} \setminus \mathscr{J}_{\bar{k}} \mid r_J \leqslant z_k\},$$

$$z_{\bar{k}} := \max\{z_{\bar{k}},\ \min\{r_J | J \in \mathscr{T} \setminus \mathscr{J}_k\}\},$$

$$A_{\bar{k}} := \{J \in \mathscr{T} \setminus \mathscr{J}_k \mid r_J \leqslant z_{\bar{k}}\},$$

thus preparing for the next step. Note that at some point the set of unscheduled jobs that could possibly be assigned to a certain machine might become empty, so that the schedule will continue on the other machine only. The procedure terminates when all the jobs in $\mathscr{J}$ have been scheduled.

Proposition 1. *At the end of the procedure, let $\mathscr{V}_k$ be the set of jobs that have been assigned to machine M_k, for $k=1,2$. Then*

1. $\mathscr{J}_k \subseteq \mathscr{V}_k$.
2. *Consider a OMP with set of jobs $\mathscr{V}_k$ and processing times $p_J := p_{kJ}$. Then Schrage's procedure applied to this OMP yields exactly the same starting times τ_J, $J \in \mathscr{V}_k$.*

Proof.

1. Follows from the fact that at each step $A_{\bar{k}} \cap \mathscr{J}_k = \emptyset$, so that no jobs of $\mathscr{J}_k$ can be assigned to $M_{\bar{k}}$, and, at the end of the procedure, all jobs have been assigned.
2. When the procedure assigns a job to machine M_k, it is the one with the largest tail among those available on machine M_k at the moment. Further, it is scheduled either at its release date or immediately following the last one assigned to machine M_k so far. But these are exactly the rules according to which a LTH solution for the OMP relative to machine M_k on jobs $\mathscr{V}_k$ is obtained. □

We can summarize this last result by saying that the heuristic procedure determines an assignment $\mathscr{V}_1$, $\mathscr{V}_2$ of jobs to machines and two Schrage schedules for jobs in $\mathscr{V}_1$ and $\mathscr{V}_2$, respectively.

The procedure described can be coded with complexity $O(n \log n)$ as follows: first, we sort the jobs by their r_J values, with cost $O(n \log n)$. Then we implement $\mathscr{A}_1$ and $\mathscr{A}_2$ as max-heaps, ordered according to q_J values. For deciding which job is available at a certain time z_k, we simply scan the remaining jobs by increasing values of r_J inserting in A_k those for which $r_J \leqslant z_k$; insertion in a heap costs $O(\log n)$. Then, choosing from each heap the best element has cost $O(1)$ and, after a job is scheduled, we remove it from the heap(s) where it belongs, again with cost $O(\log n)$. Since each element enters a heap and is deleted at most once, the total cost of managing the heaps is $O(n \log n)$. It follows that the overall complexity of the procedure is $O(n \log n)$.

3.1. An example

Consider the following instance of $R2|r_i, q_i|C_{\max}$, with $\mathscr{J}_1 = \emptyset$, $\mathscr{J}_2 = \{J_3\}$:

job J	J_1	J_2	J_3	J_4	J_5	J_6	J_7
p_{1J}	8	3	8	15	5	10	5
p_{2J}	10	4	9	10	2	10	4
r_J	2	2	8	12	15	18	29
q_J	10	12	9	2	7	3	1

The corresponding heuristic solution, of value 36, is given in Fig. 1. Note that the heuristic procedure applied to this instance with $\mathscr{J}_1 = \mathscr{J}_2 = \emptyset$ (i.e. at the root node of the branch and bound tree), produces a solution of value 34. This solu-

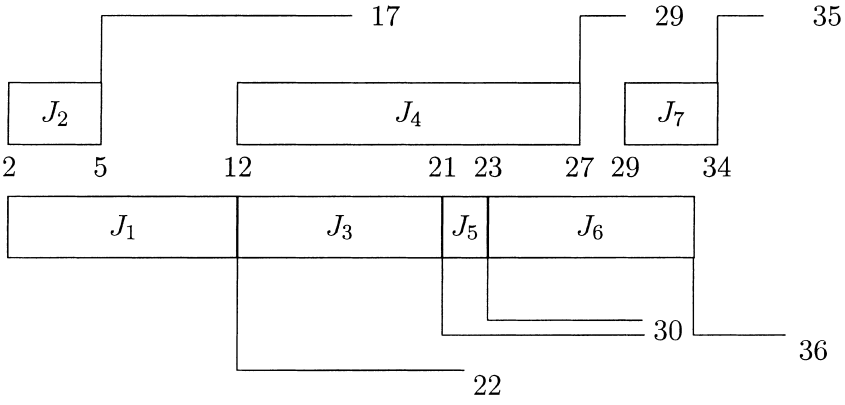

Fig. 1. Gantt chart for the example.

tion turns out to be optimal, since $r_{J_7} + \min\{p_{1J_7}, p_{2J_7}\} + q_{J_7} = 34$ is clearly a lower bound.

4. Lower bounds to $R2|r_i, q_i|C_{\max}$

Let $C^*(\mathscr{J}, \mathscr{J}_1, \mathscr{J}_2)$ be the optimal value to the problem $R2|r_i, q_i|C_{\max}$ in which $\mathscr{J}$ is the set of jobs, $\mathscr{J}_1 \subseteq \mathscr{J}$ is a subset of jobs that must be assigned to machine M_1, and equivalently for $\mathscr{J}_2$. Our goal is to determine $C^*(\mathscr{J}, \emptyset, \emptyset)$. For any $\mathscr{I} \subseteq \mathscr{J}$, consider the following problem:

$$
\begin{aligned}
&v(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2) := \min z_{\mathscr{I}},\\
&\sum_{J\in\mathscr{I}} p_{1J}x_J \leqslant z_{\mathscr{I}},\\
&\sum_{J\in\mathscr{I}} p_{2J}(1-x_J) \leqslant z_{\mathscr{I}},\\
&x_J = 1, \quad J \in \mathscr{I}\cap\mathscr{J}_1,\\
&x_J = 0, \quad J \in \mathscr{I}\cap\mathscr{J}_2,\\
&x_J \in \{0,1\}, \quad J \in \mathscr{I}.
\end{aligned}
$$

Then we have

Proposition 2. *For every nonempty set* $\mathscr{I} \subseteq \mathscr{J}$,

$$
\begin{aligned}
l(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2) &:= \min_{J\in\mathscr{I}} r_J + v(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2) + \min_{J\in\mathscr{I}} q_J\\
&\leqslant C^*(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2).
\end{aligned}
$$

Proof. Consider an optimal solution of value $C^*(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2)$, and let s_J and $e_J := s_J + p_{1J}x_J + p_{2J}(1-x_J)$ be the corresponding starting and ending times of job processings. Let F_1 be the first job of $\mathscr{I}$ appearing in the sequence on machine M_1 and L_1 be the last; analogously define F_2 and L_2 (if no job of $\mathscr{I}$ appears on machine M_k, F_k and L_k are not defined and each future reference to machine M_k has to be ignored). Then, for $k = 1, 2$ we have $r_{F_k} + (e_{L_k} - s_{F_k}) + q_{L_k} \leqslant C^*(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2)$ and hence also

$$
\begin{aligned}
&\min_{J\in\mathscr{I}} r_J + \max\{e_{L_1} - s_{F_1}, e_{L_2} - s_{F_2}\}\\
&\quad + \min_{J\in\mathscr{I}} q_J \leqslant C^*(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2).
\end{aligned}
$$

But $v(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2) \leqslant \max\{e_{L_1} - s_{F_1}, e_{L_2} - s_{F_2}\}$, so the claim follows. □

Example 3.1 (continued)**.** Consider again the instance of Example 3.1. Here we have $v(\mathscr{J}, \emptyset, \{J_3\}) = 25$, obtained by putting jobs $\{J_1, J_2, J_6\}$ on machine M_1 and $\{J_3, J_4, J_5, J_7\}$ on machine M_2; the corresponding lower bound is $l(\mathscr{J}, \emptyset, \{J_3\}) = 2 + 25 + 1 = 28$.

By means of Proposition 2 we can derive a lower bound to our problem. Unfortunately, computing $v(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2)$ is in general NP-Hard. To see this, we notice that the NP-complete problem of determining if a set S of objects can be partitioned into two subsets of equal size, can be solved by finding $v(S, \emptyset, \emptyset)$ – where the size of each object is considered as its processing time on both the machines – and testing if it equals half the sum of the sizes. Although NP-Hard, the problem of finding $v(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2)$ is among the relatively easily solvable ones. For instance, we can transform it into a pair of knapsack problems as follows.

For simplicity, we will consider the case when $\mathscr{J}_1 = \mathscr{J}_2 = \emptyset$; this is done without loss of generality, since the presence of different $\mathscr{J}_1, \mathscr{J}_2$ amounts to simply changing some constants in the constraints and the objective function. Define binary variables $w_J = 1$ iff J goes on machine M_2 and consider the problems

$$
\begin{aligned}
(K1)\quad v_1 := \min \quad & \sum_{J\in\mathscr{I}} p_{1J}(1-w_J),\\
& \sum_{J\in\mathscr{I}} p_{2J}w_J \leqslant \sum_{J\in\mathscr{I}} p_{1J}(1-w_J),\\
& w_J \in \{0,1\},
\end{aligned}
$$

and

$$
\begin{aligned}
(K2)\quad v_2 := \min \quad & \sum_{J\in\mathscr{I}} p_{2J}(1-x_J),\\
& \sum_{J\in\mathscr{I}} p_{1J}x_J \leqslant \sum_{J\in\mathscr{I}} p_{2J}(1-x_J),\\
& x_J \in \{0,1\}.
\end{aligned}
$$

Then we have

Proposition 3. $v(\mathscr{I}, \emptyset, \emptyset) = \min\{v_1, v_2\}$.

Proof. Partition the set of all possible solutions of the original problem into those in which machine M_1 is loaded more than machine M_2 and vice versa.

Then it is easy to see that for $k = 1, 2$, v_k is the optimal value of the solutions in which machine M_k is loaded more than machine $M_{\bar{k}}$. □

The problems stated are in fact knapsack problems; by removing the constants and changing the objective from minimization to maximization, we can reformulate $(K1)$ (and analogously $(K2)$) as

$$\max \quad \sum_{J\in\mathscr{J}} p_{1J} w_J,$$
$$\sum_{J\in\mathscr{J}} (p_{1j} + p_{2J}) w_J \leqslant \sum_{J\in\mathscr{J}} p_{1J},$$
$$w_J \in \{0, 1\}.$$

Knapsack problems can be solved very efficiently by means of some existing procedures ([12]); however, for our purposes, good lower bounds to the optimal solution value suffice. One way of obtaining lower bounds is to allow fractional solutions. For $j = 1, \ldots, n$, define $\alpha_j := p_{1J_j}/(p_{1J_j} + p_{2J_j})$; then we can find a lower bound to v_1 by considering objects in order of nonincreasing α_j, taking those that fit the knapsack and a fraction of the first one which does not fit. That is, we assign to machine M_2 the jobs with the highest α_J and to machine M_1 those with the smallest α_J, and possibly split a job on the two machines. Allowing fractional values, the procedure will end with machine M_1 and machine M_2 equally loaded. Hence, there is no need to compute a similar lower bound for v_2. Also, note that this lower bound coincides with the linear programming relaxation of the problem.

The complexity of finding lower bounds this way within a branch and bound procedure is $O(n)$. In fact, although sorting the α's has cost $O(n \log\ n)$, we need to do it only once, and then we can use the same sorting for all the subsets $\mathscr{J}$ on which we want to compute the lower bound. Finally, we recall that the same value for the lower bound can also be obtained by applying the method described by Potts ([13]). This method does not require sorting, and achieves complexity $O(n)$ by using median techniques.

Example 3.1 (continued)**.** Allowing fractional solutions, we get a lower bound on $v(\mathscr{J}, \emptyset, \{J_3\})$ equal to $232/9$, which, by integrality of the data, we can round up to 24; then, the corresponding lower bound to the original problem is 27.

It is interesting to note that the same LP bound is found by solving a Lagrangian dual problem, and again the solution requires a sorting of the α_j values. The symmetry of the problem is such that dualizing the constraint relative to machine M_1 or M_2 yields the same solution; so, consider the Lagrangian function, for $\lambda \geqslant 0$

$$L(\lambda) = \min \quad z_{\mathscr{J}} + \lambda \left(\sum_{J\in\mathscr{J}} p_{2J}(1 - x_J) - z_{\mathscr{J}} \right),$$
$$\sum_{J\in\mathscr{J}} p_{1J} x_J \leqslant z_{\mathscr{J}},$$
$$x_J \in \{0, 1\}.$$

Computing the Lagrangian function requires then to solve the problem

$$L'(\lambda) = \max \quad \sum_{J\in\mathscr{J}} \lambda p_{2J} x_J + (\lambda - 1) z_{\mathscr{J}},$$
$$\sum_{J\in\mathscr{J}} p_{1J} x_J \leqslant z_{\mathscr{J}},$$
$$x_J \in \{0, 1\},$$

where $L(\lambda) = \lambda \sum_{J\in\mathscr{J}} p_{2J} - L'(\lambda)$. For $\lambda \geqslant 1$ the problem is trivially solved and $x_J = 1$ for all J. Consider now $\lambda \in [0, 1)$; since in this case the coefficient of $z_{\mathscr{J}}$ in L' is negative, equality $\sum_{J\in\mathscr{J}} p_{1J} x_J = z_{\mathscr{J}}$ will hold and so the objective function becomes

$$\max \quad \sum_{J\in\mathscr{J}} (\lambda p_{2J} + (\lambda - 1) p_{1J}) x_J.$$

To maximize it, we check if $\lambda p_{2J} + (\lambda - 1) p_{1J} > 0$, in which case we set $x_J = 1$, else we set $x_J = 0$. Without loss of generality assume the jobs are numbered so that $\alpha_1 \leqslant \alpha_2 \leqslant \ldots \leqslant \alpha_n$ and let $\alpha_0 = 0, \alpha_{n+1} = 1$. Then, for $\lambda \in [\alpha_k, \alpha_{k+1})$ the variables $x_{J_1}, \ldots, x_{J_k}$ are set to 1, while $x_{J_{k+1}}, \ldots, x_{J_n}$ are set to 0. Hence, the value of L' in the interval $[\alpha_k, \alpha_{k+1})$ is $L'(\lambda) = \lambda \sum_{j=1}^{k} (p_{1J_j} + p_{2J_j}) - \sum_{j=1}^{k} p_{1J_j}$ from which

$$L(\lambda) = \lambda \left(\sum_{j=k+1}^{n} p_{2J_j} - \sum_{j=1}^{k} p_{1J_j} \right) + \sum_{j=1}^{k} p_{1J_j}.$$

The dual problem requires to maximize $L(\lambda)$ for $\lambda \geqslant 0$. We see that $L(\lambda)$ is a piecewise linear function and with slope decreasing from a positive to a negative value. Hence to find the λ which maximizes $L(\lambda)$ we only need to find a k such that

$$\sum_{j=k+1}^{n} p_{2J_j} \geqslant \sum_{j=1}^{k} p_{1J_j} \wedge \sum_{j=k+2}^{n} p_{2J_j} \leqslant \sum_{j=1}^{k+1} p_{1J_j}.$$

Then the optimal solution to the dual problem will be $\lambda = \alpha_{k+1}$. Again, note that, given a sorting of the jobs by their α_j values, the dual problem can be solved with complexity $O(n)$.

Although one may expect that $\mathscr{I} \subseteq \mathscr{I}'$ implies $l(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2) \leqslant l(\mathscr{I}', \mathscr{J}_1, \mathscr{J}_2)$, it is easily seen that this is not true; in fact adding a job to $\mathscr{I}$ will certainly increase $v(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2)$ but might as well decrease $\min_{J\in\mathscr{I}} r_J + \min_{J\in\mathscr{I}} q_J$.

So, in general, we could improve the lower bound relative to a set $\mathscr{I}$ by considering some subsets of $\mathscr{I}$. For the singleton subsets, the lower bound can be immediately computed as

$$l(\{J\}, \mathscr{J}_1, \mathscr{J}_2) = r_J + P + q_J$$

where $P = p_{1J}$ if $J \in \mathscr{J}_1$, $P = p_{2J}$ if $J \in \mathscr{J}_2$ and $P = \min\{p_{1J}, p_{2J}\}$ if $J \notin \mathscr{J}_1 \cup \mathscr{J}_2$.

It follows that $\tilde{l}(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2) := \max\{l(\mathscr{I}, \mathscr{J}_1, \mathscr{J}_2), \max_{J\in\mathscr{I}} l(\{J\}, \mathscr{J}_1, \mathscr{J}_2)\}$ is a valid lower bound, which we use for the branch and bound procedure.

Example 3.1 (continued)**.** In example 3.1 we have $\max_{J\in\mathscr{J}} l(\{J\}, \emptyset, \{J_3\}) = r_{J_7} + p_{2J_7} + q_{J_7} = 34$. Hence $\tilde{l}(\mathscr{J}, \emptyset, \{J_3\}) = 34$.

5. The branching rule

In this section we describe the branching rule used to partition the set of feasible solutions of a subproblem $\mathscr{P}$ associated to a node of the search tree. The branching will be of two possible types, which we call *assignment branching* and *sequence branching*. The assignment branching fixes for one or more jobs the machine on which they must be executed, while the sequence branching constrains the order in which the jobs can appear in the sequence on the machines.

As described in Section 2, the set of feasible solutions of a problem $\mathscr{P}$ at any node of the branch and bound tree, is $B(\mathscr{P}) \times S'(\mathscr{P})$, with $B(\mathscr{P}) \subseteq \{0,1\}^n$ and $S'(\mathscr{P}) \subseteq S_n$. $B(\mathscr{P})$ is obtained from $\{0,1\}^n$ by specifying two disjoint subsets of $\mathscr{J}$, say $\mathscr{J}_1(\mathscr{P})$ and $\mathscr{J}_2(\mathscr{P})$ and fixing the components of $\mathbf{x}$ in $\mathscr{J}_1(\mathscr{P})$ to 1 and those in $\mathscr{J}_2(\mathscr{P})$ to 0. That is,

$$B(\mathscr{P}) := \{\mathbf{x} \in \{0,1\}^n : x_J = 1, J \in \mathscr{J}_1(\mathscr{P}), \; x_J = 0, J \in \mathscr{J}_2(\mathscr{P})\}.$$

As far as $S'(\mathscr{P})$ is concerned, problem $\mathscr{P}$ will have associated an acyclic directed graph with vertex set $\mathscr{J}$ and arc set $E(\mathscr{P})$, where each arc $(J_u, J_v) \in E(\mathscr{P})$ connects jobs that are both in $\mathscr{J}_1(\mathscr{P})$ or $\mathscr{J}_2(\mathscr{P})$, and represents the constraint 'job J_u must be sequenced before job J_v'. Note that the problem $\mathscr{P}$ is completely identified by the triple $(\mathscr{J}_1(\mathscr{P}), \mathscr{J}_2(\mathscr{P}), E(\mathscr{P}))$. For simplicity, from now on, we will omit the dependence of $B(\mathscr{P})$, $S'(\mathscr{P})$, $\mathscr{J}_1(\mathscr{P})$, $\mathscr{J}_2(\mathscr{P})$, and $E(\mathscr{P})$ on $\mathscr{P}$ whenever there is no risk of confusion.

It is possible to force the heuristic feasible solutions to sequence the jobs according to the restrictions given by the arcs in E. Note that if (I,J) are a pair of jobs connected by an arc of E, and they are fixed on machine M_k, then the starting time of J in any feasible solution cannot be smaller than $r_I + p_{kI}$, so that we can redefine the release date of J as $\max\{r_J, r_I + p_{kI}\}$. Analogously, we can reset the tail of I as $\max\{q_I, p_{kJ} + q_J\}$. The idea, as in Carlier [2], is then to keep for each problem current values of r's and q's, such that for each $J \in \mathscr{J}$ we have

$$\begin{aligned} r_J &= \max\left\{r_J^0, \max_{(I,J)\in E}\{r_I + p_{kI}\}\right\}, \\ q_J &= \max\left\{q_J^0, \max_{(J,I)\in E}\{p_{kI} + q_I\}\right\}, \end{aligned} \tag{5.1}$$

where r^0, q^0 are the original values of release dates and tails.

When these assumptions are met, if $(I,J) \in E$ then $r_I < r_J$ and $q_I > q_J$, so that the heuristic procedure will schedule job I before job J. For the problem at the root of the search tree, the r's and q's are the original ones; in Section 5.2 we describe

how to update the r's and q's values whenever E changes so as to guarantee Eq. (5.1).

To solve a problem $\mathscr{P}$, we start by running the heuristic described in Section 4, this way obtaining a feasible solution, with assignment $\{h_J | J \in \mathscr{J}\}$ and starting times $\{\tau_J | J \in \mathscr{J}\}$, in which jobs in $\mathscr{J}_1$ appear in the sequence on machine M_1 and those in $\mathscr{J}_2$ have been scheduled on machine M_2. For $k = 1, 2$, let $C(M_k)$ be the completion time of machine M_k, i.e. the maximum completion time of the jobs that have been scheduled on machine M_k by the heuristic. We define as *critical machine* machine M_1 if $C(M_1) \geqslant C(M_2)$ and M_2 otherwise. The critical machine will be indexed by γ. A *critical sequence* for machine M_k, $k = 1, 2$, is defined as a sequence of jobs $(J(1), \ldots, J(p))$ which have all been assigned to M_k, such that $\tau_{J(1)} = r_{J(1)}$, $\tau_{J(i+1)} = \tau_{J(i)} + p_{kJ(i)}$, for $i = 1 \ldots p-1$ and $\tau_{J(p)} + p_{kJ(p)} + q_{J(p)} = C(M_k)$. Let $\mathscr{C}_k$ be the longest (i.e. with the largest number of jobs) critical sequence on machine M_k, $k = 1, 2$. Without loss of generality, renumber the jobs so that $\mathscr{C}_\gamma = (J_1, J_2, \ldots, J_p)$; in the following, unless otherwise specified, by critical sequence we refer to $\mathscr{C}_\gamma$.

If all the jobs of the critical sequence have been fixed (i.e. $\mathscr{C}_\gamma \subseteq \mathscr{J}_\gamma$) and $\min_{J \in \mathscr{C}_\gamma} q_J = q_{J_p}$, then the heuristic solution is optimal for $\mathscr{P}$, since any other solution will list the jobs of $\mathscr{C}_\gamma$ on machine M_γ in some order, and the makespan will be at least $\min_{J \in \mathscr{C}_\gamma} r_J + \sum_{J \in \mathscr{C}_\gamma} p_{\gamma J} + \min_{J \in \mathscr{C}_\gamma} q_J = C(M_\gamma)$. Hence, in this case no branching is needed for the node corresponding to the problem $\mathscr{P}$. Otherwise, either the sequence could be changed to improve the solution value, or there is at least one job on the critical sequence which does not necessarily appear on machine M_γ in an optimal solution. Then, we may improve the current solution by assigning this job to the other machine. To choose between these two possibilities, we look for a *critical job* J_c which is informally defined as the last job of the critical sequence which is either free or has a tail 'too small'. Let $c \in \{1, \ldots p\}$ be such that

$$J_{c+1}, J_{c+2}, \ldots, J_p \in \mathscr{J}_\gamma \quad \wedge \quad q_{J_{c+1}}, q_{J_{c+2}}, \ldots, q_{J_{p-1}} \geqslant q_{J_p}$$

and

$$J_c \notin \mathscr{J}_\gamma \vee (J_c \in \mathscr{J}_\gamma \wedge q_{J_c} < q_{J_p}).$$

5.1. Assignment branching

Suppose $J_c \notin \mathscr{J}_\gamma$. Let $\mathscr{F}_\gamma := \mathscr{C}_\gamma \setminus \mathscr{J}_\gamma$ be the set of all jobs in the critical sequence which have not been fixed on machine M_γ and could possibly be put on machine $M_{\bar{\gamma}}$. Analogously, define $\mathscr{F}_{\bar{\gamma}}$. Assume $\mathscr{F}_\gamma$ has been ordered as $\{I_1, I_2, \ldots, I_{|F_\gamma|}\}$. We derive $|F_\gamma| + 1$ subproblems of $\mathscr{P}$ by fixing the jobs of $\mathscr{F}_\gamma$ on either machine M_1 or machine M_2 as follows. For $k = 1, \ldots, |\mathscr{F}_\gamma| + 1$, we fix the jobs $I_1, I_2, \ldots, I_{k-1}$ on machine M_γ and I_k, if $k \leqslant |\mathscr{F}_\gamma|$, on machine $M_{\bar{\gamma}}$.

For $k = 1, \ldots, |F_\gamma| + 1$ define $\mathscr{J}_{\gamma k} = \mathscr{J}_\gamma \cup \{I_1, \ldots, I_{k-1}\}$ and $\mathscr{J}_{\bar{\gamma} k} = \mathscr{J}_{\bar{\gamma}} \cup \{I_k\}$, where we consider $\{I_{|\mathscr{F}_\gamma|+1}\} = \emptyset$. Further, let $B_k = \{\mathbf{x} \in \{0,1\}^n : x_J = 1, J \in \mathscr{J}_{1k}, x_J = 0, J \in \mathscr{J}_{2k}\}$. Then we have

$$B = B_1 \cup \cdots \cup B_{|\mathscr{F}_\gamma|+1} \quad \text{and} \quad B_k \cap B_h = \emptyset, \quad k \neq h \in \{1, \ldots, |\mathscr{F}_\gamma| + 1\}.$$

We have established the following result.

Proposition 4. *The assignment branching partitions B into $|\mathscr{F}_\gamma| + 1$ disjoint subsets B_k.*

When an assignment branching is performed in the branch and bound strategy, for each $k \in \{1, \ldots, |\mathscr{F}_\gamma| + 1\}$ we generate a new problem $\mathscr{P}_k$, child of $\mathscr{P}$, with feasible set $B_k \times S'(\mathscr{P})$. For this problem we compute two lower bounds, $l_1 = \tilde{l}(\mathscr{F}_\gamma, \mathscr{J}_{1k}, \mathscr{J}_{2k})$ and $l_2 = \tilde{l}(\mathscr{F}_1 \cup \mathscr{F}_2, \mathscr{J}_{1k}, \mathscr{J}_{2k})$.

We set $\mathrm{lb}(\mathscr{P}_k)$, the lower bound for $\mathscr{P}_k$ as $\mathrm{lb}(\mathscr{P}_k) := \max\{l_1, l_2, \mathrm{lb}(\mathscr{P})\}$. If $\mathrm{lb}(\mathscr{P}_k)$ is not smaller than the value of the best feasible solution found so far, we immediately discard problem $\mathscr{P}_k$, otherwise we insert it in the list of open problems.

The assignment branching leaves open the question as how to order $\mathscr{F}_\gamma$ By empirical testing we have found that a good strategy is to sort the jobs in $\mathscr{F}_\gamma$ by nonincreasing processing times. Actually, since the p_{iJ}'s are fixed throughout the branch and bound, we do not need to sort $\mathscr{F}_\gamma$ each time; we instead compute two lists (one for each machine) of all the jobs, sorted by processing times, at the very beginning and then scan these lists when considering the elements of $\mathscr{F}_\gamma$.

5.2. Sequence branching

Suppose now $J_c \in \mathscr{J}_\gamma$ and $q_{J_c} < q_{J_p}$. Define $\mathscr{G} := \{J_{c+1}, \ldots, J_p\}$. We want to show that to improve the makespan on machine M_γ we must schedule job J_c either before all jobs in $\mathscr{G}$ or after all of them.

Denote by $C^*(M_\gamma)$ the best makespan possible on machine M_γ in any solution of $\mathscr{P}$. Clearly $C^*(M_\gamma) \leqslant C(M_\gamma)$. For $\mathscr{A} \subseteq \mathscr{J}_\gamma$, define

$$h(\mathscr{A}) := \min_{J\in\mathscr{A}} r_J + \sum_{J\in\mathscr{A}} p_{\gamma J} + \min_{J\in\mathscr{A}} q_J.$$

Note that if for $\mathscr{A} \subseteq \mathscr{J}_\gamma$ and $J \in \mathscr{J}_\gamma \setminus \mathscr{A}$, we have $h(\mathscr{A}) + p_{\gamma J} > C^*(M_\gamma)$, then in an optimal sequence for machine M_γ, either J precedes all jobs in $\mathscr{A}$ or follows all of them. In fact, otherwise the sequence would contain a subsequence of jobs starting with a job J' of $\mathscr{A}$, going through J and finishing with a job J'' of $\mathscr{A}$; its cost would be at least

$$\begin{aligned} &r_{J'} + \sum_{I\in\mathscr{A}} p_{\gamma I} + p_{\gamma J} + q_{J''} \\ &\quad \geqslant \min_{I\in\mathscr{A}} r_I + \sum_{I\in\mathscr{A}} p_{\gamma I} + \min_{I\in\mathscr{A}} q_I + p_{\gamma J} \\ &\quad = h(\mathscr{A}) + p_{\gamma J} > C^*(M_\gamma). \end{aligned}$$

Following the proof given in [2], which applies since, by Proposition 1, the schedule for machine M_γ is a Schrage one, we have

$$\min_{J\in\mathscr{G}} r_J > r_{J_1} + p_{\gamma J_1} + \cdots + p_{\gamma J_{c-1}}.$$

Then we obtain

$$\begin{aligned} h(\mathscr{G}) &= \min_{J\in\mathscr{G}} r_J + \sum_{J\in\mathscr{G}} p_{\gamma J} + \min_{J\in\mathscr{G}} q_J \\ &= \min_{J\in\mathscr{G}} r_J + p_{\gamma J_c} + \sum_{i=c+1,\ldots,p} p_{\gamma J_i} + \min_{J\in\mathscr{G}} q_J - p_{\gamma J_c} \\ &> r_{J_1} + p_{\gamma J_1} + \cdots + p_{\gamma J_{c-1}} + p_{\gamma J_c} \\ &\quad + \cdots + p_{\gamma J_p} + q_{J_p} - p_{\gamma J_c} \\ &= C(M_\gamma) - p_{\gamma J_c}. \end{aligned}$$

Therefore $h(\mathscr{G}) + p_{\gamma J_c} > C(M_\gamma) \geqslant C^*(M_\gamma)$, and we conclude that in an optimal solution for machine M_γ either J_c follows all jobs in $\mathscr{G}$ or precedes all of them. Since in order to improve the solution for problem $\mathscr{P}$ we must improve the makespan on machine M_γ, we can split the set of feasible solution to $\mathscr{P}$ by imposing the precedence constraints $J_c \prec \mathscr{G}$ and $\mathscr{G} \prec J_c$. Hence we generate two new subproblems of $\mathscr{P}$, say $\mathscr{P}_1$ and $\mathscr{P}_2$. In the first we set

$$E(\mathscr{P}_1) := E(\mathscr{P}) \cup \{(J_c, J_j), j = c+1, \ldots, p\}$$

and in the second

$$E(\mathscr{P}_2) := E(\mathscr{P}) \cup \{(J_j, J_c), j = c+1, \ldots, p\}.$$

Then we update the values for release dates and tails to associate to each subproblem $\mathscr{P}_i$ so that 5.1 holds. Let $E^-(\mathscr{P}_i)$ be the inverse graph of $E(\mathscr{P}_i)$ (i.e. $(u,v) \in E^-(\mathscr{P}_i)$ iff $(v,u) \in E(\mathscr{P}_i)$) and consider the graphs $G_i := (\mathscr{J}_\gamma, E(\mathscr{P}_i) \cup \{(s,J), J \in \mathscr{J}_\gamma\})$ and $G_i^- := (\mathscr{J}_\gamma, E^-(\mathscr{P}_i) \cup \{(s,J), J \in \mathscr{J}_\gamma\})$ defined by adding a source s connected to each job in $\mathscr{J}_\gamma$. Let $\delta(I,J)$ be the length of an edge of G_i, defined as follows:

$$\delta(I,J) = \begin{cases} r_J^0 & \text{if } I = s, \\ p_{\gamma I} & \text{otherwise.} \end{cases}$$

Analogously, lengths $\delta'(I,J)$ for edges in G_i^- are defined as

$$\delta'(I,J) = \begin{cases} q_J^0 & \text{if } I = s, \\ p_{\gamma I} & \text{otherwise.} \end{cases}$$

Then it is seen that Eq. (5.1) holds if and only if for each $J \in \mathscr{J}_\gamma$, r_J is the length of a longest $s-J$ path in G_i and q_J is the length of a longest $s-J$ path in G_i^-. Hence, we can compute the r_J and q_J with a longest path algorithm, that, being the graph acyclic, runs in time linear in the number of edges.

Finally, we compute lower bounds for $\mathscr{P}_i, i = 1,2$ as in [2]:

$$\begin{aligned} &\mathrm{lb}(\mathscr{P}_i) \\ &\quad = \max\{\tilde{l}(\mathscr{G}, \mathscr{G}, \emptyset), \tilde{l}(J_c \cup \mathscr{G}, J_c \cup \mathscr{G}, \emptyset), \mathrm{lb}(\mathscr{P})\}. \end{aligned}$$

6. Computational results

The algorithm has been coded in Pascal, and run on a Sun Workstation. To test the procedure, we first generated 180 random instances, of respectively, 10, 50, 100, 150, 200 and 250 jobs. For

each instance on n jobs, we drew the values of processing times on the two machines with the uniform distribution in $1,\ldots,50$, and release dates and tails with the uniform distribution in $1,\ldots,nk$, where k is a parameter of values $5,10,15,\ldots,100,110,\ldots,200$. This way of generating instances is the same as the one used in [2] for the OMP.

The results, reported in Tables 1 and 2, are somewhat consistent with the results of Carlier's procedure for the OMP. In this latter, most of the times Schrage's heuristic produces an optimal solution and no branching is needed. Analogously, for $R2|r_i,q_i|C_{\max}$, in many cases the heuristic solution was optimal. The tougher instances occur for $k=5$ (Table 1). This can be explained for two reasons. First, for a small value of k the data describing the jobs are very similar among them, making the assignment decision difficult. Second, when k is large the release dates and tails dominate the processing times, making the sequencing decision easier. This behavior can be expected also for Carlier's procedure, even if in $R2|r_i,q_i|C_{\max}$ the increase in cost incurred for these instances is higher. The entries in Table 1 have the following meaning. Each row refers to a problem on n jobs. The value in column H is the number of times the heuristic procedure has been called, i.e. how many times a problem has been considered for branching; columns AB and SB give the total number of assignment and sequence branching performed. The final column gives the system CPU time, expressed in hundredths of a second.

In Table 2 we have grouped the remaining instances, and each row corresponds to a set of problems, generated for $k=10,\ldots,200$ with fixed n. Now columns 'avg' report the average of values over all the instances of a row, while columns 'max' report the max values. For each set of instances, the minimum values where 1 for H and 0 for both assignment and sequence branchings. Again, the more difficult problems corresponded to $k=10$, but they were not as difficult as for $k=5$, which is why we decided to consider those instances separately. Overall, we see that the maximum running time is in the order of ten seconds, while most of the problems were solved in one hundredth of a second, at the root of the branch and bound tree.

Table 1
Instances for $k=5$

n	H	AB	SB	t
10	7	5	1	2
50	2846	2544	267	378
100	534	534	0	180
150	466	466	0	218
200	89	89	0	105
250	3379	3332	46	980

Table 2
Instances for $k=10,15,\ldots,200$

n	H_{avg}	$H_{\max}$	AB_{avg}	$AB_{\max}$	SB_{avg}	$SB_{\max}$	t_{avg}
10	1.8	18	0.6	5	0.2	7	4
50	4.2	11	1.1	3	0.7	4	7
100	3.9	16	1.0	9	0.6	5	7
150	9.6	101	3.4	46	3.8	53	20
200	3.8	22	0.6	5	1.2	15	13
250	4.1	18	1.1	6	0.9	10	11

We then generated a new set of problems on $n=20,\ 40,\ 60,\ 80$ and 100 jobs, for $k=5,10,\ldots,95,100$. This time, to make the problems more difficult, we imposed that for each job the processing times on machine M_1 and M_2 cannot be too different. In particular, for each job J we generated a random number $P\in\{1,\ldots,100\}$. Then, we picked p_{1J} and p_{2j} uniformly in the interval $3/4P,\ldots,5/4P$. This way of choosing the processing times corresponds to having an estimate P of the duration of a job, and allowing a random variation, which can never exceed $1/4P$. For each pair (k,n) we then generated 5 instances. The results are reported in Table 3.

As expected, this set of problems turned out to be more difficult than the others, and particularly for $k\leqslant 15$. We limited the maximum number of times that the heuristic can be applied to 20,000 and the maximum CPU time to 1 min (the former limit was always reached first). Some problems where not solved within these bounds. In column u we report the number of problems unsolved (out of 5). For these unsolved problems, we where however able to obtain an estimate of the deviation from the optimum, by comparing the best lower and upper bounds. In column Δ we report the average coefficient by which the value found for an unsolved problem is larger than the true

Table 3
Instances with $p_{1j} \simeq p_{2j}$

k	n	H_{avg}	H_{max}	AB_{avg}	AB_{max}	SB_{avg}	SB_{max}	u	Δ	t_{avg}
	20	412	1463	359	1305	19	86	0	–	14
	40	3446	8883	2961	8144	179	684	0	–	250
5	60	4663	17994	4119	15839	308	1232	0	–	490
	80	14139	20000	13496	19443	25	96	3	0.004	2212
	100	12625	20000	11784	19784	12	52	2	0.004	2360
	20	1116	2425	635	1588	382	899	0	–	32
	40	13118	19772	11844	18729	380	1237	0	–	840
10	60	13662	20000	9859	13620	3519	9125	3	0.009	1402
	80	11157	20000	10288	18596	288	1251	2	0.025	1690
	100	7888	20000	7506	18649	0	0	3	0.035	1950
	20	167	722	76	339	70	296	0	–	15
	40	3440	14416	1011	3141	2287	10374	0	–	237
15	60	4161	20000	1116	5196	2304	11130	1	0.001	304
	80	2366	20000	420	3162	1932	16787	1	0.003	281
	100	525	2248	193	644	308	1517	0	–	77
	20	5	240	2	60	2	139	0	–	8
	40	4	67	1	29	2	35	0	–	8
$\geqslant 20$	60	5	68	2	32	1	34	0	–	10
	80	8	218	2	47	3	133	0	–	12
	100	5	80	2	25	2	60	0	–	12

optimum. We can see that this percentage rarely exceeds 1%. Considering that the running times are still in the order of some seconds, the branch and bound proves to be an effective heuristic for obtaining near-optimal solutions in those cases for which optimality was not proved.

6.1. Final remarks

Our tests show that when $p_{1J} \simeq p_{2J}$, for some problems the algorithm may run into several assignment branchings. In particular, when $p_{1J} = p_{2J}$ for all J, one should possibly use an algorithm specifically developed for *identical* rather than unrelated parallel machines ([3]).

The branch and bound procedure could be generalized to the case of $m > 2$ unrelated parallel machines. The heuristic procedure remains basically the same, and its complexity becomes $O(mn \log n)$. As for the branching rule, we can expect the assignment branching to be extremely inefficient, due to the very large number of possible ways of removing a job from a critical sequence to put it on some other machine. Therefore, in such more general case, a different approach should be possibly used. The generalization to the case when jobs have precedence constraints is on the other hand immediate, since the branch and bound already takes care of constraints of this type.

Acknowledgements

I wish to thank Federico Della Croce for helpful discussions on the subject, and a referee for many useful comments.

References

[1] P. Barcia, K. Jörnsten, Improved Lagrangean decomposition: An application to the generalized assignment problem, European Journal of Operational Research 46 (1990) 84–92.

[2] J. Carlier, The one-machine sequencing problem, European Journal of Operational Research 11 (1982) 42–47.

[3] J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize the makespan, European Journal of Operational Research 29 (1987) 298–306.
[4] E.G. Coffman, M. Yannakakis, M.J. Magazine, C. Santos, Batch sizing and job sequencing on a single machine, Annals of Operations Research 26 (1990) 135–147.
[5] M.R. Garey, D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[6] K. Jörnsten, M. Näsberg, A new Lagrangian relaxation approach to the generalized assignment problem, European Journal of Operational Research 27 (1986) 313–323.
[7] E.L. Lawler, Recent results in the theory of machine scheduling, in: A. Bachem, M. Grötschel, B. Korte (Eds.), Mathematical Programming, The State of the Art–Bonn 1982, Springer, Berlin, 1982, pp. 202–231.
[8] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, Sequencing and Scheduling: Algorithms and Complexity, Handbooks in OR and MS, 4, Elsevier, New York, 1993.
[9] B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan, Minimizing maximum lateness on one machine: Computational experience and some applications, Statistica Nederlandica 30 (1976) 25–41.
[10] J.K. Lenstra, A.H.G, Rinnooy Kan, Sequencing and Scheduling, in: M. O'h Eigeartaigh, J.K. Lenstra, A.H.G. Rinnooy Kan (Eds.), Combinatorial Optimization: Annotated Bibliographies, Wiley, New York, 1985.
[11] G.B. Mc Mahon, M. Florian, On scheduling with ready times and due dates to minimize maximum lateness, Operations Research 23 (1975) 475–482.
[12] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Wiley, Chirchester, 1990.
[13] C.N. Potts, Analysis of a linear programming heuristic for scheduling unrelated parallel machines, Discrete Applied Mathematics 10 (1985) 155–164.