Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Int. J. Production Economics 100 (2006) 168–179

international journal of production economics

www.elsevier.com/locate/ijpe

New exact method to solve the $Pm/r_j/\sum C_j$ schedule problem

F. Yalaoui*, C. Chu

ISTIT-OSI, CNRS FRE 2732, Lab. Ind. Systems Optimization, University of Technology of Troyes (UTT), 12, rue Marie Curie, BP 2060, 10010 Troyes, Cedex, France

Received 19 May 2004; accepted 1 November 2004
Available online 16 December 2004

Abstract

This paper addresses a parallel machine scheduling problem (PMSP) with release dates to minimize sum of completion time. This problem presents numerous potential applications in real life. An efficient exact branch and bound method is developed using theoretical properties. By allowing job splitting or by relaxing release date constraints, a polynomial lower bounding scheme is proposed. The method was tested on more than 2000 randomly generated medium-sized instances. We have solved medium-sized instances in a reasonable amount of time. Our method is the first attempt to solve exactly the considered problem. It opens an interesting way in PMSP with sum of completion time criterion.

Keywords: Parallel machine scheduling; Total flow time; Sum of completion time; Release dates; Job splitting; Lower bound; Branch and bound

1. Introduction

Since many years, various research works have been carried out to deal with parallel machine scheduling problems (PMSP). This interest comes from their numerous potential applications in real life. The PMSP with release date constraints has been shown to be NP-hard (Lenstra et al., 1977), whatever the criterion considered. For the sum of completion time criterion, Du et al. (1991) proved that the problem is NP-hard even if job preemption is allowed. But, they proved that if the processing times of all jobs are identical and the number of machines does not exceed two, the problem is solvable in polynomial time. Two important observations are made about the existing literature. First, most of the published papers with branch and bound algorithms consider the total *weighted* flow time criterion with identical machines and equal job release dates. See for instance, Elmaghraby and Park (1974), Barnes and Brennan (1977), Sarin et al. (1988), Belouadeh and Potts (1994) and Azizoglu and Kirca (1999).

*Corresponding author. Tel.: +33 325715626; fax: +33 32571 5649.
E-mail addresses: yalaoui@utt.fr (F. Yalaoui), Chengbin.Chu@utt.fr (C. Chu).

0925-5273/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2004.11.002

Second, important efforts have been made to develop approximation schemes or to obtain worst-case performance ratios approximation algorithms (Leonardi and Raz, 1997; Phillips et al., 1995; Hall, 1997; Phillips et al., 1998).

For more information about general scheduling problems, see the survey papers of Lawler et al. (1993), Cheng and Sin (1990), Hoogeveen et al. (1997), Cai et al. (1998) and Chen et al. (1998), Liao and Lin (2003), and Lin and Jeng (2004).

This paper addresses, using an exact method, the PMSP with release dates to minimize sum of completion time. This criterion is equivalent to minimize sum of or mean flow time, total or mean waiting time. To construct efficient branch and bound algorithm, we develop theoretical properties, upper and lower bounds. A lower bounding scheme is proposed by allowing job splitting or by relaxing release dates constraints. The relaxed problems $Pm/r_j, a_m, \text{Split}/\sum C_j$ and $Pm/r_j < a_{\min}, a_m/\sum C$ are proved to be polynomially solvable. Our method has been tested on more than 2000 randomly generated medium-sized problems.

This paper is organized as follows. Section 2 establishes a set of theoretical properties. Section 3 describes the structure of the branch and bound method. Section 4 reports the numerical experiments and analyzes the computational results. Section 5 concludes the paper.

2. Theoretical properties

The problem considered in this paper is to schedule N jobs indexed from 1 to N on M identical machines indexed from 1 to M. Each job i $(i = 1, 2, \ldots, N)$ has a release date r_i and a processing time p_i. The criterion is to minimize the sum of completion time. Since the objective function is non-decreasing with job completion times, the criterion is regular. Therefore, only semi-active and active schedules need to be considered. In a semi-active schedule, no job can be completed earlier without changing the job sequence while in an active schedule; no job can be completed earlier without delaying at least one of the other jobs.

For our problem, any semi-active schedule corresponds to a unique permutation on the set $\{1, 2, \ldots, N\}$. For any given permutation, a corresponding semi-active schedule can be constructed by assigning the first unscheduled job in the permutation to the earliest available machine with ties broken by selecting the machine with the smallest index. Conversely, for any given semi-active schedule, a permutation can be obtained by ordering the jobs with respect to non-decreasing starting times with ties broken in non-decreasing order of machine indexes. As a consequence, a partial schedule is a permutation of a subset of $\{1, 2, \ldots, N\}$.

In the remainder of this paper, the word "schedule" indifferently refers to a partial schedule or a complete schedule, except where otherwise noted. The following notation will be used.

$R_i(t) = \max(t, r_i)$	: the earliest starting time of job i at time t;
$E_i(t) = \max(t, r_i) + p_i$	: the earliest completion time of job i at time t;
$J(K)$	: set of jobs scheduled in schedule K;
$J_m(K)$	: set of jobs scheduled on machine m in schedule K;
$C_i(K)$	: completion time of job i in schedule K;
$\Delta_i(K)$	: completion time of the job immediately preceding job i in schedule K. If job i is the first job in K, $\Delta_i(K)$ is set to 0, without loss of generality;
$F(K)$	: sum of completion time of jobs scheduled in schedule K;
$FT_m(K)$	: sum of completion time of jobs scheduled on machine m in schedule K;
$\varphi_m(K)$	: completion time of the last scheduled job on machine m in schedule K called *finishing time* of machine m in schedule K;
$\mu(K)$	: the smallest indexed machine available the earliest in schedule K;
$K \circ \{i\}$	: new schedule obtained by adding job i after K (i.e. by adding job i on μ (K));
$\Omega(K,i)$	: the schedule composed of $K \circ i$, completed by the partial optimal schedule of remaining jobs.

When there is no ambiguity, $J(K)$, $J_m(K)$, $C_i(K)$, $\Delta_i(K)$, $F(K)$, $\varphi_m(K)$, and $\mu(K)$ are simplified into J, J_m, C_i, Δ_i, F, φ_m, and μ, respectively.

We will use for this problem some theoretical properties and observations presented in Yalaoui and Chu (2002).

Observation 1. (Yalaoui and Chu (2002)): *Any partial schedule K with strictly more than $N-M+1$ jobs scheduled on a machine can be discarded. In any optimal schedule resulting from K, the number of additional jobs on any machine m is at most $U_m(K) = \min[N - M + 1 - |(J_m)|, N - |(J(K))|]$. In the remainder, we denote by $U(K)$ the maximum over all $U_m(K)'s$; i.e., $U(K) = \max_{\{1 \leqslant m \leqslant M\}} U_m(K)$.*

We present in the remaining part of the section, the dominance properties used in the branch and bound algorithm. A dominance property allows the identification of the dominated sets of solutions. For each solution S in a dominated set, there is at least one solution outside the set that is at least as good as S. Since a partial solution can lead to a set of complete solutions, dominance properties allow identifying non-promising partial solutions.

The main idea of the following properties is to identify the conditions which permit to stop the construction of a complete schedule using information about the current partial schedule and the task to be processed. Note also that except Property 3, the other Properties are proved using the same scheme based on the construction of a schedule better than the given one. We give for the some cases a representation of the improved schedule. Figs. 1–3, are representations of the schedule S and $\Omega(K,i)$, presented in the different properties.

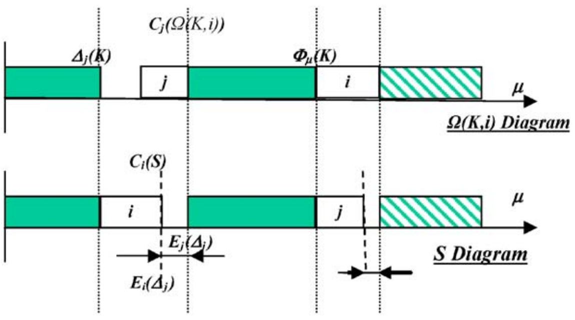

Fig. 1. Property 2.

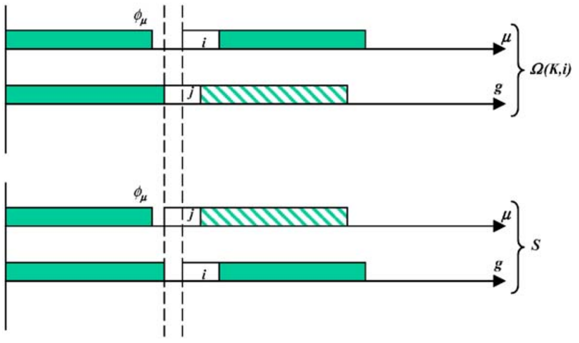

Fig. 2. Property 3.

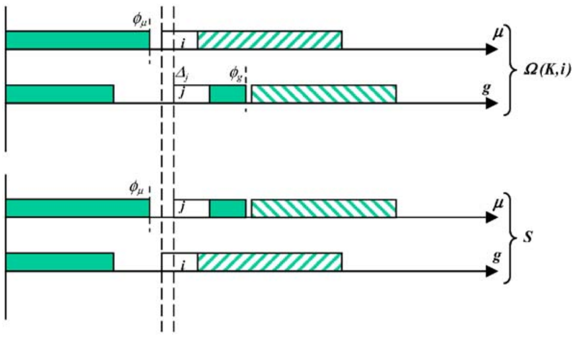

Fig. 3. Property 6.

These dominance properties can be categorized into three: those comparing an unscheduled job with a job in the partial scheduled (Properties 2 and 3) labeled as Group 1; those comparing unscheduled job among themselves (Properties 4–6) labeled as Group 2; and those using "dynamic programming properties" (Property 7), labeled as Group 3.

Property 2. *For any partial schedule K and for any job not scheduled in K, if there is a job j scheduled in K on machine μ, the machine available the earliest, such that $E_i(\Delta_j) \leqslant E_j(\Delta_j)$ and $E_i(\Delta_j) - E_j(\Delta_j) \leqslant (p_i - p_j) \times U_\mu(K)$ then $\Omega(K,i)$ is dominated.*

Proof. Consider schedule $\Omega(K,i)$. Note that by definition, in this schedule job i is scheduled on machine μ. Construct another schedule S identical to $\Omega(K,i)$ but the positions of jobs i and j are interchanged (see Fig. 1). Due to the assumptions, the resulting schedule S is feasible. We examine two cases: In the first case, $p_i > p_j$, and in the second $p_i \leqslant p_j$.

If $p_i > p_j$, we must have $R_j(\Delta_j) > R_i(\Delta_j)$ since $E_i(\Delta_j) = R_i(\Delta_j) + p_i \leqslant E_j(\Delta_j) = R_j(\Delta_j) + p_j$. Then we can conclude that $R_i(\Delta_j) < r_j$. We have $C_i(S) + C_j(S) \leqslant C_i(\Omega(K,i)) + C_j(\Omega(K,i))$. The other jobs scheduled after job j in $\Omega(K,i)$ can be scheduled earlier. Then $\Omega(K,i)$ is dominated (See Fig. 1).

If $p_i \leqslant p_j$, the jobs between jobs j and i can be completed earlier and the jobs after job i on machine μ in $\Omega(K,i)$ are delayed. The completion time of any job is increased by at most $(p_j - p_i)$. According to Corollary 1, the number of these jobs is at most $U_\mu(K) - 1$. In addition, we have

$$C_j(S) - C_i(\Omega(K,i)) \leqslant (p_j - p_i)$$

and

$$C_i(S) - C_j(\Omega(K,i)) = E_i(\Delta_j) - E_j(\Delta_j).$$

From the assumptions of the property, we have:

$$F(S) - F(\Omega(K,i)) \leqslant (p_j - p_i) \times U_\mu(K) + E_i(\Delta_j) - E_j(\Delta_j) \leqslant 0.$$

In other words, $\Omega(K,i)$ is dominated. □

Property 3. *For any partial schedule K and any job i not scheduled in K, schedule $\Omega(K,i)$ is dominated if there is a job j scheduled on machine $g \neq \mu$ in K such that $\Delta_j \leqslant \varphi_\mu$ and $R_i(\varphi_\mu) < r_j$.*

Proof. Construct a semi-active schedule S by interchanging the jobs scheduled on machines g after time Δ_j with jobs scheduled on machine μ after time φ_μ. The completion time of each of these jobs is decreased or remains the same. Therefore, $\Omega(K,i)$ is dominated. □

Property 4. *For any partial schedule K and for any job i not scheduled in K, if there is another job j not scheduled in K such that $E_i(\varphi_\mu) \geqslant E_j(\varphi_\mu)$ and $E_i(\varphi_\mu) - E_j(\varphi_\mu) \geqslant (p_i - p_j) \times (U(K) - 1)$ then $\Omega(K, i)$ is dominated.*

Proof. Two cases must be examined.

If jobs i and j are both scheduled on machine μ in $\Omega(K,i)$, the same argument as in the single machine counterpart can be applied and we can conclude that $\Omega(K,i)$ is dominated (see Bianco and Ricciardelli, 1982).

If job j is scheduled on another machine, say machine g, two subcases need to be examined. In the first subcase, $C_j(\Omega(K,i)) - p_j \leqslant R_i(\varphi_\mu)$, and in the second, $C_j(\Omega(K,i)) - p_j > R_i(\varphi_\mu)$.

In the first subcase, i.e., $C_j(\Omega(K,i)) - p_j \leqslant R_i(\varphi_\mu)$, we have $\varphi_\mu \leqslant \varphi_g \leqslant C_j(\Omega(K,i)) - p_j \leqslant R_i(\varphi_\mu)$. Construct another scheduling S identical to $\Omega(K,i)$ except that jobs initially scheduled on machine g after job j (including job j) are scheduled on machine μ and jobs initially scheduled on machine μ after job i (including job i) are scheduled on machine g. Due to the fact that $\varphi_\mu \leqslant \varphi_g \leqslant C_j(\Omega(K,i)) - p_j \leqslant R_i(\varphi_\mu)$, the completion time of no job is increased. Therefore $F(S) \leqslant F(\Omega(K,i))$.

In the second subcase, i.e. $C_j(\Omega(K,i)) - p_j > R_i(\varphi_\mu)$, construct another schedule S identical to $\Omega(K,i)$ except that jobs i and j are interchanged. With the assumptions of the property, we obtain

$$FT_\mu(S) \leqslant FT_\mu(\Omega(K,i)) + E_j(\varphi_\mu) - E_i(\varphi_\mu)$$

with $FT_\mu(S)$ and $FT_\mu(\Omega(K,i))$ the sum of completion time of jobs scheduled on machine μ respectively in schedules S and $\Omega(K,i)$. Each of the jobs scheduled after job j on machine g in $\Omega(K,i)$ are delayed by at most $(p_i - p_j)$ if $p_i \geqslant p_j$ and can be completed earlier if $p_i < p_j$. In any case, the completion time of each of those jobs is increased by at most max $(p_i - p_j, 0)$. The number of these jobs is at most $U(K) - 2$. Considering further the fact that $C_i(S) - C_j(\Omega(K,i)) \leqslant \max(p_i - p_j, 0)$, we obtain $FT_g(S) \leqslant FT_g(\Omega(K,i)) + \max(p_i - p_j, 0) \times (U(K) - 1)$.

Together with the relation $FT_\mu(S) \leqslant FT_\mu(\Omega(K,i)) + E_j(\varphi_\mu) - E_i(\varphi_\mu)$ and the conditions of the property, we can conclude that schedule $\Omega(K,i)$ is dominated. □

Property 5. *For any partial schedule K and any job i not scheduled in K, schedule $\Omega(K,i)$ is dominated if there is another unscheduled job j such that*

$$\left[E_j(\varphi_\mu) - R_i(\varphi_\mu)\right] \times \left[U_\mu(K) - 1\right] \leqslant \min\left[R_j(\psi) - R_j(\varphi_\mu); \min_{\{k \notin J(K)\}} (p_k)\right],$$

where ψ is the second smallest machine finishing time in K.

Proof. First note that if $E_j(\varphi_\mu) - R_i(\varphi_\mu) \leqslant 0$, $\Omega(K, i)$ is obviously dominated due to the fact that it is not an active schedule. Therefore, we assume that $E_j(\varphi_\mu) - R_i(\varphi_\mu) > 0$. Two cases must be examined.

If jobs i and j are both scheduled on machine μ in $\Omega(K, i)$, we have

$$E_j(\varphi_\mu) - R_i(\varphi_\mu) \leqslant [E_j(\varphi_\mu) - R_i(\varphi_\mu)] \times [U_\mu(K-1)] \leqslant \min\left[R_j(\psi) - R_j(\varphi_\mu); \min_{\{k \notin J(K)\}}(p_k)\right] \leqslant \min_{\{k \notin J(K)\}}(p_k).$$

According to Chu (1995), the partial schedule on machine μ is dominated, and $\Omega(K, i)$ is also dominated.

If job j is scheduled on machine $g \neq \mu$, construct another schedule S identical to $\Omega(K, i)$ except that job j is removed from machine g and inserted before job i on machine μ. We have $C_j(S) - C_j(\Omega(K, i)) \leqslant R_j(\varphi_\mu) - R_j(\psi)$. For any job scheduled after job i (including job i itself), the completion time is increased by at most $E_j(\varphi_\mu) - R_i(\varphi_\mu)$. The number of these jobs is at most $U_\mu(K)$–1.

As a consequence, $F(S) - F(\Omega(K, i)) \leqslant R_j(\varphi_\mu) - R_j(\psi) + [E_j(\varphi_\mu) - R_i(\varphi_\mu)] \times [U_\mu(K) - 1] \leqslant 0$. Therefore, schedule $\Omega(K, i)$ is dominated. □

Property 6. *For any partial schedule K and any job i not scheduled in K, schedule $\Omega(K, i)$ is dominated if there is another unscheduled job j such that $E_i(\varphi_\mu) \leqslant E_j(\varphi_\mu)$ and $(p_i - p_j) \leqslant [E_i(\varphi_\mu) - E_j(\varphi_\mu)] \times [U_\mu(K) - 1]$.*

Proof. The assumption of the property implies that $p_i \leqslant p_j$ and $R_i(\varphi_\mu) \geqslant R_j(\varphi_\mu)$. Two cases must be examined.

If jobs i and j are both scheduled on machine μ then we can use the same argument as in the single machine problem (for the proof see Chu (1992)) to conclude that $\Omega(K, i)$ is dominated.

If job j is scheduled on another machine, say machine g, two subcases must be examined. In the first subcase, $C_j(\Omega(K, i)) - p_j \leqslant R_i(\varphi_\mu)$, and in the second, $C_j(\Omega(K, i)) - p_j > R_i(\varphi_\mu)$.

In the first subcase; i.e., $C_j(\Omega(K, i)) - p_j \leqslant R_i(\varphi_\mu)$, construct another scheduling S identical to $\Omega(K, i)$ except that the jobs initially scheduled on machine g after job j (including job j itself) are scheduled on machine μ and the jobs initially scheduled on machine μ after job i (including job i itself) are scheduled on machine g. Due to the fact that $C_j(\Omega(K, i)) - p_j \geqslant \varphi_g \geqslant \varphi_\mu$, the completion time of no job is increased. Therefore, $F(S) \leqslant F(\Omega(K, i))$.

In the second subcase; i.e., $C_j(\Omega(K, i)) - p_j > R_i(\varphi_\mu)$, construct another schedule S identical to $\Omega(K, i)$ except that jobs i and j are interchanged. With the assumptions of the property, we obtain $FT_g(S) \leqslant FT_g(\Omega(K, i)) + (p_i - p_j)$. Each of the jobs scheduled after job i on machine μ in $\Omega(K, i)$ is delayed by at most $E_j(\varphi_\mu) - E_i(\varphi_\mu)$. The number of these jobs is at most $[U_\mu(K) - 2]$. Considering further the fact that

$$C_j(S) - C_i(\Omega(K, i)) = E_j(\varphi_\mu) - E_i(\varphi_\mu),$$

we obtain

$$FT_\mu(S) \leqslant FT_\mu(\Omega(K, i)) + (E_j(\varphi_\mu) - E_i(\varphi_\mu)) \times [U_\mu(K) - 1].$$

Together with the relation

$$FT_g(S) \leqslant FT_g(\Omega(K, i)) + (p_i - p_j)$$

and the conditions of the property, we can conclude that schedule $\Omega(K, i)$ is dominated. □

Property 7. *For any partial schedule K, and any job i not scheduled in K, if there is another partial schedule K′ such that*

$$J(K') = J(K) \cup \{i\}, F(K') \leqslant F(K \circ \{i\}) \quad \text{and} \quad [N - |(J(K))| - 1] \times \delta \leqslant F(K \circ \{i\}) - F(K')$$

then $\Omega(K, i)$ is dominated, where $\delta = \max_{\{k=1,\ldots,M\}}(\varphi'k - \varphi k)$ with φ_k and φ'_k being the kth smallest machine finishing time in partial schedules $K \circ \{i\}$ and K′, respectively.

Proof. Construct another schedule S from partial schedule K', a partial schedule obtained by any method, in the following way. For any k such that $1 \leqslant k \leqslant M$, identify from $K \circ \{i\}$ the machine finishing at the kth position, denoted g_k. The jobs scheduled after φ_k on machine g_k in $K \circ \{i\}$ are scheduled after φ'_k in the same order on the machine finishing at the kth position in partial schedule K', denoted as machine g'_k.

If $\delta \leqslant 0$, it is obvious that $F(S) \leqslant F(\Omega(K,i))$ from the conditions stated in the property. Therefore, $\Omega(K,i)$ is dominated.

If $\delta > 0$, let Q_k ($k = 1, 2, \ldots, M$) be the number of jobs scheduled on machine g'_k after φ'_k in S. Compared with $\Omega(K,i)$, the completion time of each of these jobs is increased by at most $(\varphi'_k - \varphi_k)$ if $(\varphi'_k - \varphi_k) > 0$, and is not increased if $(\varphi'_k - \varphi_k) \leqslant 0$. Let $L = \{k/1 \leqslant k \leqslant M \text{ and } \varphi'_k \leqslant \varphi_k\}$. Therefore,

$$\begin{aligned} &F(S) - F(\Omega(K,i)) \\ &\quad \leqslant F(K') - F(K \circ \{i\}) + \sum_{k \in L} (\varphi'_k - \varphi_k) \times Q_k \\ &\quad \leqslant F(K') - F(K \circ \{i\}) + \delta \times \sum_{k \in L} Q_k. \end{aligned}$$

Due to the fact that $\sum_{k \in L} Q_k \leqslant N - |J(K)| - 1$, $\Omega(K,i)$ is dominated. $\square$

3. Lower bounds

First, we study properties of the shortest processing time (SPT) rule. Conway et al. (1967) showed that the problem, denoted P, of minimizing the sum of completion time on identical parallel machines such that all machines and jobs are available at time 0 is solvable in $O(N \log N)$ time using the generalized SPT rule. Assume, without loss of generality, that the jobs are renumbered in the SPT order i.e, $p_1 \leqslant p_2 \leqslant \ldots \leqslant p_N$. The generalized SPT rule can be described as follows. The jobs are scheduled in the order of their indexes. After jobs 1 to $i-1$ are scheduled, job i is assigned to the machine which becomes available at the earliest. The machine where job 1 is scheduled has the largest number of jobs scheduled. Let $q = N \bmod M$. We see that if $q = 0$ (meaning that N is an integer multiple of M), then $n = \lfloor N/M \rfloor$ (n integer) jobs are scheduled on each machine. If $q > 0$, the number of jobs scheduled on machine m is $\lfloor N/M \rfloor + 1$ for $m = 1, 2, \ldots, q$ and $\lfloor N/M \rfloor$ for $m = q+1, \ldots, M$.

This result can be generalized to the case where machines are not all available at the same time.

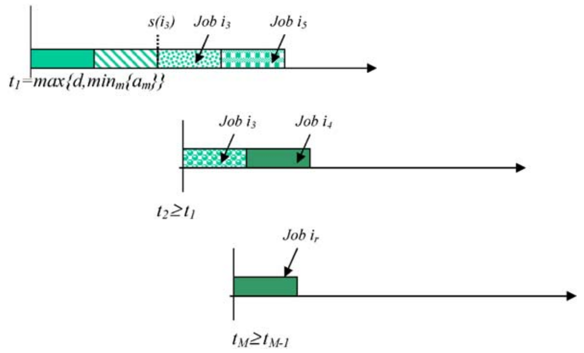

Fig. 4. Proposition 1.

This generalization is given in the following proposition.

Proposition 1. *The generalized* SPT *rule minimizes sum of completion time on identical parallel machines with unequal machine availability dates, if all jobs have identical release dates.*

Proof. Let d denote the common release date of all the jobs. Let a_m denote the time of availability of machine m; $m = 1, \ldots, M$. Let $t_1 = \min\{d, \min_m\{a_m\}\}$ which denotes the earliest start time of processing of any job. Observe that for any $t \geqslant t_1$ all the jobs are "ready". Renumber the machines and rank the machine in non-decreasing order of their availability, see Fig. 4, so that

$$t_1 \leqslant a_2 = t_2 \leqslant \ldots \leqslant a_M = t_M.$$

Since all the jobs are available for processing, then there exists an optimal schedule that the jobs are arranged in SPT on any machine. It remains only to demonstrate that the *generalized* SPT rule yields the optimum. This is demonstrated by a simple exchange of the jobs. Consider Fig. 4 which GSPT let to job i_3 being on machine 1 and job i_4 on machine 2. (In the general setting we would have job i_r and i_{r+1}; but there is no need to confuse the issue with complex notation). By construction, $p(i_3) \leqslant p(i_4) \leqslant p(i_5)$. Denote the start of job i_3 by $s(i_3)$. If jobs i_3 and i_4 were switched we would have (denoting the completion time of job i_j by $C(i_j)$

before the exchange, and by $C'(i_j)$ after the exchange),

$$\begin{aligned}(\text{sum}) &= C(i_3) + C(i_4) + C(i_5)\\ &= [s(i_3) + p(i_3) + s(i_3) + p(i_3) + p(i_5)] + [t_2 + p(i_4)]\\ &= [2s(i_3) + 2p(i_3) + p(i_5)] + [t_2 + p(i_4)].\end{aligned}$$

The sum of the completion times after the exchange is given by

$$\begin{aligned}(\text{sum})' &= C'(i_3) + C'(i_4) + C'(i_5)\\ &= [s(i_3) + p(i_4) + s(i_3) + p(i_4) + p(i_5)] + [t_2 + p(i_3)]\\ &= [2s(i_3) + 2p(i_4) + p(i_5)] + [t_2 + p(i_3)].\end{aligned}$$

Comparing these two sums, it is clear that $(\text{sum}) \geqslant (\text{sum})'$ by virtue of the fact that $p(i_4) \geqslant p(i_3)$. Therefore i_3, i_4 and i_5 should be in GSPT. The argument can be repeated for any pair of machines in order of their availability times, and the assertion is demonstrated. □

To find a lower bound, we relax the problem. The usual relaxation scheme is to allow job preemption. However, the relaxed problem remains NP-hard (Du et al., 1991). We then consider two different relaxations: the first one is based on job splitting and the second one is obtained by assuming that all release dates are equal.

The lower bound based on equal release dates is obtained by setting all release dates to 0, i.e. for all $i = 1, \ldots, N$, $r_i = 0$. This relaxed problem, as shown in Proposition 1, can be polynomially solved by the generalized SPT rule, even if the machines are not available at the same time.

Therefore, we focus on the lower bound based on job splitting. The considered relaxation is through job splitting. Job splitting not only allows the processing of a job to be interrupted and resumed later, but also allows a job to be processed simultaneously by more than one machine.

We show that this relaxed problem can be polynomially solved by the extended shortest remaining processing time (SRPT) rule, even if not all machines are available at the same time: At any time, a job with the SRPT among available jobs is simultaneously processed on all the available machines. The processing is interrupted if another job becomes available with a processing time strictly shorter than the remaining processing time of the job in process. The following property states that this schedule is optimal for the relaxed problem.

Property 8. *If job splitting is allowed, the identical PMSP to minimize sum of completion time with release dates constraints can be polynomially solved using the extended SRPT rule even if the machines are not all available at the same time. The complexity of the method is $O(N \log N)$.*

Proof. Let S^* be the schedule obtained by the extended SRPT rule. Let S^0 be any feasible schedule different from S^*. We show that $F(S^*) \leqslant F(S^0)$.

Let t^0 be the first time that S^0 differs from S^*. Note that it is impossible that at time t^0 some machines are idle in S^* while they are occupied in S^0, since in S^* as soon as machines become available, they start processing a job with the SRPT among those available.

Let m^0 be the smallest indexed machine performing a different activity on S^0 than in S^* at time t^0. Two cases are possible. In the first case, machine m^0 is processing a job i in S^* while it is idle in S^0. In the second case, m^0 is processing a job i at time t^0 in S^* while it is processing a different job j in S^0.

In the first case, another schedule S^1 can be constructed from S^0, by processing a part of job i in the idle interval, the completion time of job i is then decreased without increasing that of any other job.

In the second case, it is necessarily true that the remaining processing time of job i at time t^0 is shorter than that of job j. We can construct another schedule S^1 by reassigning the time slots dedicated either to job i or to job j in S^0. The reassignment is done as follows. The time slots are first assigned to job i until job i is completed. The remaining time slots are assigned to job j. Due to the fact that the remaining processing time of job i at time t^0 is shorter than that of job j, the sum of completion times of jobs i and j are reduced without increasing that of any other job.

Table 1
An example

Jobs	1	2	3	4	5	6	7
Ready times	0	1	4	3	2	4	8
Processing times	3	4	6	2	8	10	7
Cj	3	5	7	4	12	16	10

In either case, a new schedule S^1 can be obtained such that $F(S^1) \leqslant F(S^0)$.

The same process is repeated by comparing schedule S^* and S^1. Let t^1 be the first time that S^1 differs from S^*. With the same argument as for S^0, we will obtain another schedule S^2 such that $F(S^2) \leqslant F(S^1) \leqslant F(S^0)$. And so on and so forth, since either $t^k > t^{k-1}$ or $t^k = t^{k-1}$ but $m^k > m^{k-1}$ for any $k \geqslant 1$, a schedule identical to S^* will be obtained. We will obtain

$$F(S^*) \leqslant \cdots \leqslant F(S^2) \leqslant F(S^1) \leqslant F(S^0). \quad \square$$

This result can be also obtained intuitively as illustrated as follows. Let job i have a parameters p_i (processing time) and r_i (ready time or release date). Suppose that we have M machines with available time a_m; $m = 1, \ldots, M$. Then the problem reduces to the simple problem of "filling" the M machines with the available jobs, ordered first by the ready times and secondly by the time that the fits in the remaining machines (typically the job with the SRPT, as asserted before), with unavailability of the machine blocked out. One can construct a "tableau" similar to the transportation tableau, in which the rows are the machines and the columns are the jobs. The "filling" of the cells of such tableau with the objective of minimizing the sum of the completion times is a simple matter of filling a column with the shortest available job (or residue of a job), unless a job becomes ready that "fits" in the remaining rows. The following tables give an example.

Machines	1	2	3
Availability	0	2	5

The optimum is $\Sigma C_j = 57$. The completion time of each job is given in the fourth row of Table 1. Observe that at time $t = 2$, jobs 1, 2 are ready, with job 1 in progress with only one time unit of processing remaining. It is allowed to continue processing in period 3. In period 3 jobs 2, 4, 5 are ready, with job 4 having the smallest processing time. It is processed in period 3 and 4, etc.

4. Branch-and-bound algorithm

The branch-and-bound algorithm developed is inspired by the branch-and-bound algorithm proposed by Chu (1992) to solve a single machine scheduling problem. The algorithm is explained as follows. Each node in the search tree represents a partial schedule.

We first calculate an initial solution, which represents the first upper bound. It is given by the best solution found by two heuristics HPRTF and HAL. Both methods are based on priority rules. At each iteration, for the two methods, the job with the highest priority is assigned to the earliest available machine. In HAL, a job has the highest priority if it has the SPT among jobs available i.e. at each step the HAL choose the job with the smallest processing time. In HPRTF, a job has a higher priority if it has a smaller priority rule for the total flow time (PRTF) function value. For a job i at the time t the PRTF function PRTF(i,t) is defined as: $\mathrm{PRTF}(i,t) = 2\max(t, r_i) + p_i$. For the definition of the PRTF function and more information see Chu (1992).

The branching from a node consists of assigning an additional job on the machine that becomes available the earliest in the partial schedule represented by the node. The node to be explored next is the head of the list of unexplored nodes arranged in increasing order of their lower bounds, with ties broken according to non-increasing number of scheduled jobs. Before any new node is created, a series of dominance properties are checked. For each node of the search tree which cannot be eliminated by dominance properties, a lower bound is calculated. If the lower bound is greater than or equal to the sum of completion time of an already known complete schedule, this node is also eliminated.

Table 2
Computational results for $N = 10$ to 110 and $M = 2$ to 5 with LB_SRPT

M	Parameter	N														
		10	15	20	25	30	35	40	45	50	60	70	80	90	100	110
2	Nodes	5	10	30	122	199	620	983	1879	2202	1702	1135	1654	302	489	575
	Time	1	6	40	351	751	11957	18544	49438	106997	93342	69333	74949	2583	12485	18157
	Nosolv	—	—	—	—	—	—	2	3	5	10	11	11	14	14	16
3	Nodes	7	25	185	1423	1469	1406	2707	959	635						
	Time	2	22	730	29635	75326	70412	144354	16203	51577						
	Nosolv	—	—	—	—	3	7	13	16	20						
4	Nodes	8	29	264	1332	1135	1475	936	613							
	Time	3	31	1992	49798	31237	82299	46628	17758							
	Nosolv	—	—	—	2	6	13	15	18							
5	Nodes	10	22	151	544	1701	1754	1229	794							
	Time	6	27	949	20903	79540	82031	59562	42450							
	Nosolv	—	—	—	3	4	11	15	19							

Table 3
Computational results for $N = 10$ to 120 and $M = 2$ to 5 with LB_SRPT-SPT

M	Parameter	N															
		10	15	20	25	30	35	40	45	50	60	70	80	90	100	110	120
2	Nodes	4	8	23	82	143	448	789	1421	2101	1897	824	1325	773	528	554	564
	Time	1	7	44	268	678	7955	152240	47647	97867	91683	54184	64727	62272	31544	32539	33189
	Nosolv	—	—	—	—	—	—	1	2	3	5	10	11	13	13	14	14
3	Nodes	5	19	125	926	978	1241	2535	1979	2259							
	Time	2	21	500	24002	41624	64921	129888	95453	80457							
	Nosolv	—	—	—	—	1	3	9	13	18							
4	Nodes	5	17	154	961	1062	1690	946	1530								
	Time	2	20	1069	34617	35268	114828	43852	107872								
	Nosolv	—	—	—	—	1	4	12	16								
5	Nodes	5	12	87	436	192	1194	926	1556								
	Time	3	15	532	20079	84338	56440	51760	111307								
	Nosolv	—	—	—	—	1	4	12	15								

5. Computational results

This section describes the computational results. The branch-and-bound algorithm was implemented using a *C Compiler language* and tested on an *HP workstation J400* (*two processors and a memory of one Gigabyte*). To generate the test problems, we adapted the scheme proposed by Hariri and Potts (1983) for the single machine problem and used by subsequent works (Chu, 1992, 1995). The test problems contain $M = 2$, 3, 4, 5 or 10 machines and $N = 10$, 15, 20, 25, 30, …, 120 jobs. For each job, an integer processing time p_i is generated using a uniform distribution on $[1, 100]$ and an integer release date between 0 and $\lambda \times N \times 50.5/M$. Ten values of $\lambda \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.25, 1.50, 1.75, 2.0, 3.0\}$ are considered to generate 50 examples for each pair of (N; M) with 5 problems for each value of λ.

The computational results are described in the following tables where "Nodes", "Time" and "Nosolv" represent, respectively, the average number of nodes generated over the 50 instances, the average computation time of instances solved within 30 minutes out of 50 instances time in CPU milliseconds and the number of unsolved problems out of 50 in a class within a limit of 30 minutes (for each problem).

Two sets of results are obtained. The first one, represented by Table 2, gives the computational results when the lower bound is based on the extended SRPT rule only, denoted LB_SRPT.

The second one, represented by Table 3, gives the computational results when the lower bound is based on the best result using the extended SRPT rule or the generalized SPT rule, denoted LB_SRPT-SPT.

From the tables, we can see that the number of nodes, the computations times and the number of unsolved problems increase with the size of the problem (i.e. number of machines and number of jobs). This phenomenon is due to the NP-hardness of the problem. Note that the number of nodes generated and the computation time are calculated only for solved instances. This explains why some data increase with the size of the problem and why the some data are larger with LB_SRPT-SPT than with LB_SRPT alone.

The unsolved instances come from the set generated with $\lambda \in \{0.4, 0.6, 0.8, 1.0\}$. If λ is small, the generated problems are easy, because the release dates are almost the same. This class of problem, after small computation time, becomes equivalent to a problem with equal release dates and can be solved by the generalized SPT rule, which is a polynomial algorithm. For large values of λ, the release dates are very scattered, which implies a small number of active schedules. In this latter situation, it becomes easy to obtain an optimal solution from the set of active schedules.

Apart from these remarks, we can also make following observations. The unsolved problems are less than 13% of all test problems by using LB_SRPT as lower bound and less than 10% by using LB_SRPT-SPT as lower bound. The deviation of the average upper bound versus the average optimal solution is only about 3%.

We also analyzed the impact of the dominance properties on the performance of the branch and bound algorithm. The impact of the three groups of dominance properties can be seen through Tables 4 and 5. Table 4 gives the percentage of nodes eliminated for different numbers of machines for each group of properties, whereas Table 5 indicates the percentage of nodes eliminated for different numbers of jobs. The dominance conditions were not applied simultaneously but individually.

We note that the properties given by group 1 have a very significant influence when the number of machines is small (2 or 3). The effectiveness and the relevance of group 3 are visible when the number of machines is significant (5 and more).

Table 4
Impact of the dominance properties with the number of machines

Machine	Group 1 (%)	Group 2 (%)	Group 3 (%)
2	61.67	29.16	9.17
3	38.94	36.44	24.62
4	31.65	36.66	31.7
5	20.31	43.18	36.51
10	26.72	29.85	43.42

Table 5
Impact of the dominance properties with the number of jobs

Jobs	10	15	20	25	30	35	40	45
Group 1 (%)	10.24	23.41	32.31	45.4	48.38	48.71	45.15	42.17
Group 2 (%)	79.74	49.93	36	21.31	20.93	21.79	19.29	18.18
Group 3 (%)	10.02	23.66	31.7	33.28	30.68	29.5	35.56	39.65

When the number of machines is 4 the different dominance properties have an identical effect. We also noted that the breadth-first strategy definitely gives more interesting results in term of computing times and number of nodes. This is justified mainly by the nature of the dominance property given by Property 7.

For the analysis according to the number of jobs, we stopped with 45 jobs because it is the maximum for multi-machines case and it is the largest number of jobs that we have solved for $m \geqslant 3$ machines. We can see that the most influential properties are those of the group 2 for a number of job between 10 and 20 and of the groups 2 and 3 for remaining number of job.

We also carried out test problems with 10 machines and we could reach a number of jobs equal to 45 in reasonable computational time (171 356 ms CPU of average), with a small number of nodes (1972 on average) and only 14 problems unsolved out of 50.

A solution is obtained for instances with 10 machines and 45 jobs, 5 machines and 45 jobs problems and 3 machines and 110 jobs with LB_SRPT as lower bound and a solution is obtained for instances with 2 machines and 120 jobs when LB_SRPT-SPT is used as lower bound.

These results demonstrate the good quality of our upper bound generating scheme (using the HPRTF and HAL heuristics) and the efficiency of the lower bound based on the combination of the extended SRPT and the generalized SPT rules.

6. Conclusion

This paper examined the parallel machine scheduling problem (PMSP) with release dates to minimize sum of completion time. We proved some properties, new lower and upper bounds. These properties give rise to a branch-and-bound algorithm capable of yielding optimal solutions for problems with up to 45 jobs and 5 machines or 50 jobs and 3 machines or 120 jobs and 2 machines within a reasonable amount of computation time.

Our method is the first exact resolution of the proposed problem. It opens an interesting way in PMSP with sum of completion time criterion. We are attempting to improve the method using new lower bound, dominance properties and upper bound.

Acknowledgement

The authors are very grateful to the referees for their very constructive comments which improved very much the presentation of the paper.

References

Azizoglu, M., Kirca, O., 1999. On the minimization of sum of weighted flow time with identical and uniform parallel machines. European Journal of Operational Research 113, 91–100.

Barnes, J.W., Brennan, J.J., 1977. An improved algorithm for independent tasks to reduce mean finishing time. AIIE Transactions 17, 382–387.

Belouadeh, H., Potts, C.N., 1994. Scheduling identical parallel machines to minimize total weighted completion time. Discrete Applied Mathematics 48, 201–218.

Bianco, L., Ricciardelli, S., 1982. Scheduling of a single machine to minimize total weighted flow time subject to release dates. Naval Research Logistics 29, 151–167.

Cai, X., Lee, C.-Y., Li, C.-L., 1998. Minimizing total completion time in two-processor task system with prespecified allocations. Naval Research Logistics 45, 231–242.

Chen, B., Potts, C.N., Woeginger, G.J., 1998. A review of machine scheduling: Complexity, algorithms and approximability. In: Du, D.Z., Pardalos, P. (Eds.), Handbook of Combinatorial Optimization, vol. 3. Kluwer Academic Publishers, Dordrecht, pp. 21–169.

Cheng, T.C.E., Sin, C.C.S., 1990. A state of the art review of parallel machine scheduling research. European Journal of Operational Research 47, 271–292.

Chu, C., 1992. A branch-and-bound algorithm to minimize total flow time with unequal release dates. Naval Research Logistics 39, 859–875.

Chu, C., 1995. A new class of scheduling criteria and their optimization. RAIRO Recherche Opérationnelle/Operations Research 30, 171–189.

Conway, R.W., Maxwell, W.L., Miller, L.W., 1967. Theory of scheduling. Addison-Wesley, Reading, MA.

Du, J., Leung, J.Y.-T., Young, G.H., 1991. Scheduling chain structured tasks to minimize makespan and mean flow time. Information and Computation 92, 219–236.

Elmaghraby, S.E., Park, S.H., 1974. Scheduling jobs on a number of identical machines. AIIE Transactions 6, 1–13.

Hall, L., 1997. Approximation algorithms for scheduling. In: Hochbaum, D. (Ed.), Approximation Algorithms for NP-hard Problems.

Hariri, A.M.A., Potts, C.N., 1983. An algorithm for single machine sequencing with release dates to minimize total weighted completion time. Discrete Applied Mathematics 5, 99–109.

Hoogeveen, J.A., Lenstra, J.K., van de Velde, S.L., 1997. Sequencing and scheduling. In: Dell'Amico, M., Maffioli, F., Martello, S. (Eds.), Notated Bibliographies in Combinatorial Optimizations. Wiley, New York.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B., 1993. Sequencing and Scheduling: Algorithms and complexity. In: Graves, S.C., Rinnooy Kan, A.H.G., Zipkin, P.H. (Eds.), Handbooks in Operations Research and Management Science, Logistics of Production and Inventory, vol. 4. North-Holland, Amsterdam, pp. 445–522.

Lenstra, J.K., Rinnooy Kan, A.H.G., Brucker, P., 1977. Complexity of machine scheduling problems. Annals of Discrete Applied Mathematics 1, 343–362.

Leonardi, S., Raz, D., 1997. Approximation total flow time on parallel machines. In: Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, pp. 110–119.

Liao, C.-J., Lin, C.-H., 2003. Makespan minisation for two uniform parallel machines. International Journal of Productions Economics 84, 205–213.

Lin, B.M.T., Jeng, A.A.K., 2004. Parallel machine batch scheduling to minimize the maximum lateness and the number of tardy jobs. International Journal of Productions Economics 91, 121–134.

Phillips, C.A., Stein, C., Wein, J., 1995. Scheduling jobs that arrive over time. In: Proceedings of Fourth Workshop on Algorithm and Data Structures, Lecture Notes in Computer Science vol. 95. Springer, Berlin, pp. 86–97.

Phillips, C.A., Schulz, A.S., Shmoys, D.B., Stein, C., Wein, J., 1998. Improved bounds on relaxation of a parallel machine scheduling problem. Journal of Combinatorial Optimization 1, 413–426.

Sarin, S.C., Ahn, S., Bishop, A.B., 1988. An improved branching scheme for the branch and bound procedure of scheduling n jobs on m machines to minimize total weighted flow time. International Journal of Production Research 26, 1183–1191.

Yalaoui, F., Chu, C., 2002. Parallel machine scheduling to minimize total tardiness. International Journal of Productions Economics 76, 265–279.