Minimizing Makespan on Parallel Machines Subject to Release Dates and Delivery Times

Anis Gharbi and Mohamed Haouari

Ecole Polytechnique de Tunisie, BP 743, 2078, La Marsa. Tunisia.

Abstract

We consider the problem of minimizing the makespan on identical parallel machines subject to release dates and delivery times. The objective of this paper is to develop exact branch and bound algorithms to solve this strongly NP-hard problem. A preprocessing algorithm is devised to speed up the convergence of the proposed algorithms, and a new tight bounding scheme is introduced. The search tree is also reduced using a polynomial selection algorithm. Extensive computational experiments show that instances with up to 300 jobs can be solved exactly in a moderate CPU time.

Keywords : Scheduling, Identical parallel machines, Makespan, Release dates, Delivery times, Branch and Bound.

1 Introduction

We consider the following machine scheduling problem. A set J of n jobs has to be scheduled on m identical parallel machines ($2 \leq m < n$). Each job j ($j = 1, 2, ..., n$) has to be processed for p_j units of time by one machine out of the set of machines. The processing of job j cannot be started before its release date (or *head*) r_j, and each job j has a delivery time (or *tail)* q_j that must elapse between its completion on the machine and its exit from the system. All data are assumed to be deterministic and integer, and all machines are ready from time zero onwards. A schedule is defined as a non-negative vector $\sigma = (t_1, t_2, ..., t_n)$, where t_j denotes the start time of job j. A schedule is said to be *feasible* if:

- each machine processes at most one job at one time,
- no job is processed by more than one machine at one time,
- no operation is preempted, (i.e. operations cannot be interrupted),
- $t_j \geq r_j$ for all $j = 1, ..., n$.

The completion time of a schedule (or *makespan*) is defined as $C_{\max} = \underset{j=1,\ldots,n}{Max} \ (p_j + t_j + q_j)$. The objective is to find a feasible schedule of minimum completion time. Using the notation of [1], this problem is denoted by $P/r_j, q_j/C_{\max}$. It is worth noting that, using the same arguments as those of [2], the problem can be restated as a $P/r_j/L_{\max}$ (i.e. minimizing the maximum lateness on identical parallel machines.)

$P/r_j, q_j/C_{\max}$ is an extension of the classical identical parallel machine problem denoted by $P//C_{\max}$ which is a very fundamental problem in scheduling theory. It is also a generalization of the classical one machine problem $1/r_j, q_j/C_{\max}$, which is known to be strongly NP-hard [3]. Therefore, the $P/r_j, q_j/C_{\max}$ is NP-hard in the strong sense. However, the preemptive version denoted by $P/r_j, q_j, pmtn/C_{\max}$ is solvable in polynomial time using a network flow formulation [4]. Another interesting polynomially-solvable version is the case where all jobs have equal processing time, $P/r_j, q_j, p_j = p/C_{\max}$, which is solvable in $O(mn^2)$ time [5].

We now briefly review related problems for which exact approaches exist. On the one hand, parallel machine problems have received an enormous amount of attention from the research community. A survey of this literature is presented in [1]. On the other hand, $P/r_j, q_j/C_{\max}$, despite its theoretical and practical interest, has received only scant attention. Indeed, the literature regarding this problem is relatively scarce. To the best of our knowledge, the only exact algorithm proposed in the literature for this problem is that developed by Carlier in [6]. Worst-case performance of simple dispatching rules are analyzed in [6] and [7]. Bratley et al. [8] developed a simple enumerative algorithm for the problem $P/r_j, \bar{d}_j/C_{\max}$ where $\bar{d}_j$ represents the deadline of job j (i.e. the job has to be completed before this date). The problem with two unrelated machines, denoted by $R2/r_j, q_j/C_{\max}$, was recently addressed by Lancia who gave an enumerative method [9]. The uniform parallel machine version was investigated by Dessouky [10] who developed a B&B algorithm for $Q/r_j, q_j, p_j = 1/C_{\max}$. He reported the solution of instances with up to 80 jobs and 3 machines. Finally, several improved B&B algorithms have been proposed for the single machine problem $1/r_j, q_j/C_{\max}$. In particular, the algorithm proposed in [11] is considered very efficient.

Our motivation for the study of the $P/r_j, q_j/C_{\max}$ stems from its application in the context of the Multiprocessor Job Shop problem (MJS). The MJS problem is an important generalization of the well-known jobshop problem ($J//C_{\max}$), which may be described as follows. Assume that we have n jobs and Z centers z_i ($i = 1, 2, ..., Z$). Each center z_i contains m_i identical machines. Each job j ($j = 1, 2, ..., n$) consists of an ordered sequence of Z operations which must be processed in this order. Each operation O_{ij} has to be processed for p_{ij} units of time by one of the m_i machines of center z_i. We also make the traditional assumptions regarding jobs and machine availability. The problem is to find a feasible schedule which minimizes the makespan. During the last decade, several papers dealing with the MJS were published in the operations research literature. However, to the best of our knowledge, only heuristic approaches have been developed so far (see for instance [12], [13]). The MJS is related to $P/r_j, q_j/C_{\max}$ in the following way. Consider an instance of the MJS. If we select a center z_i and we relax the capacity constraint of all other centers (i.e. their respective capacities are assumed to be infinite). Then, one relaxation of the MJS as a $P/r_j, q_j/C_{\max}$ problem with m_i machines, can be obtained by setting, for all j $(j = 1, 2, ..., n)$:

$$\begin{cases} r_j = \sum\limits_{O_{hl} \in P_{ij}} p_{hl} \\ q_j = \sum\limits_{O_{hl} \in S_{ij}} p_{hl} \end{cases}$$

where P_{ij} and S_{ij} represent the sets of the predecessors and successors of operation O_{ij}, respectively. Clearly, an optimal solution of the derived $P/r_j, q_j/C_{\max}$ instance is a valid lower bound of the MJS problem. Thus, the challenge in developing an improved exact algorithm for the $P/r_j, q_j/C_{\max}$ problem is to achieve a significant step toward the exact solution of the MJS problem. It is worth noting that an interesting potential application of the $P/r_j, q_j/C_{\max}$ arises (in a slightly different form) in the context of the approximate solution of the MJS using a *shifting bottleneck approach* [14].

In this paper, we present two variants of a Branch and Bound (B&B) algorithm for deriving optimal solutions of the $P/r_j, q_j/C_{\max}$ problem. The paper is organized as follows. A preprocessing algorithm used to accelerate the convergence of the B&B algorithm is presented in Section 2. In Section 3, we review lower bounds from the literature, and present improved new bounds. In Section 4, several conditions are presented for the purpose of checking the feasibility of any instance and adjusting the heads and tails of jobs in order to tighten the lower bound. In Section 5, we present the details of the B&B algorithm including the branching scheme and several tree reduction rules. In Section 6, we present extensive computational experiments, and we conclude by a summary of our results and indicate some directions for future research.

2 A preprocessing algorithm

Several authors have shown that preprocessing often improves the efficiency of B&B algorithms (see for instance [15], [16]). In this section, we propose a preprocessing algorithm to simplify the problem by reducing the number of jobs to be scheduled. Preprocessing also enables us to improve the tightness of the lower bounds.

2.1 Selection rules and algorithm

The following lemma provides a sufficient condition which allows to identify the starting time of particular jobs in an optimal schedule.

Lemma 1. *Assume that the jobs of J ($|J| > m$) are ranked in non-decreasing order of their release dates. Denote by $j_{(k)}$ the job which has the kth smallest release date. Let $J_{r,m} = \{j_{(1)}, j_{(2)}, ..., j_{(m)}\}$ and $j_0 \in J_{r,m}$ such that $r_{j_0} + p_{j_0} = \min_{j \in J_{r,m}} (r_j + p_j)$.*

Assume that the following condition is satisfied :

$$r_{j_0} + p_{j_0} \leq r_{j_{(m+1)}} \tag{2.1}$$

then :

$$C^*_{\max}(J) = \max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J \backslash \{j_0\})\}$$

where $C^*_{\max}(S)$ denotes the optimal makespan for the $P/r_j, q_j/C_{\max}$ problem defined for the set of jobs S.

Proof. Clearly, we have

$$\max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J \backslash \{j_0\})\} \leq C^*_{\max}(J)$$

Consider the $P/r_j, q_j/C_{\max}$ defined for $J \backslash \{j_0\}$. In any optimal schedule for this problem, there is necessarily one machine m_0 which is idle from time zero to $r_{j_{(m+1)}}$. Then, scheduling j_0 on m_0 at time r_{j_0} yields a feasible schedule, for the $P/r_j, q_j/C_{\max}$ defined for J, with makespan equal to $\max\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J \backslash \{j_0\})\}$. ■

An immediate consequence of this lemma is that if a job j_0 satisfies condition (2.1), then there exists an optimal schedule where its starting time is fixed at r_{j_0}. Therefore, j_0 can be removed from the set of jobs to be scheduled.
We recursively apply Lemma 1 to the reduced jobset $J \backslash \{j_0\}$, until there is no job which satisfies condition (2.1) or there are no more than m unscheduled jobs.

Observation 1

The $P/r_j, q_j/C_{\max}$ is symmetric. That is, the symmetric problem obtained by reversing the roles of r_j and q_j has the same optimal makespan as the original one.

We will refer to the original problem as the *Forward* problem and to the symmetric one as the *Backward* problem. An immediate consequence of the above observation is the following lemma.

Lemma 2. *Assume that the jobs of J ($|J| > m$) are ranked in non-decreasing order of their delivery times. Denote by $j_{(k)}$ the job which has the kth smallest delivery time. Let $J_{q,m} = \{j_{(1)}, j_{(2)}, ..., j_{(m)}\}$ and $j_0 \in J_{q,m}$ such that $q_{j_0} + p_{j_0} = \min_{j \in J_{q,m}} (q_j + p_j)$.*

Assume that the following condition is satisfied :

$$q_{j_0} + p_{j_0} \leq q_{j_{(m+1)}} \tag{2.2}$$

then :

$$C^*_{\max}(J) = \max\left\{r_{j_0} + p_{j_0} + q_{j_0}, C^*_{\max}(J \backslash \{j_0\})\right\}$$

Thus, by performing a preprocessing based on condition (2.2), it is possible to remove a job j_0 satisfying this condition from the set of unscheduled jobs. Then, this preprocessing rule will be applied recursively to the jobset $J\backslash\{j_0\}$ until there is no job which satisfies condition (2.2) or there is no more than m unscheduled jobs. A maximal number of jobs can be fixed by iteratively applying the two symmetric preprocessing rules. Therefore, the problem can be solved on a reduced jobset, denoted by $\overline{J}$. Denote by J_R and J_Q the sets of jobs removed according to the preprocessing rules based on conditions (2.1) and (2.2), respectively. Lemmas 1 and 2 prove that there is an optimal schedule in which the jobs of J_R precede those of $\overline{J}$, and the jobs of J_Q succeed those of $\overline{J}$. A formal description of the preprocessing algorithm is given below.

Preprocessing Algorithm

Step 0. *Set $J_R = \emptyset, J_Q = \emptyset$ and $\overline{J} = J$.*

Step 1. Test on condition (2.1)

1.1. *If $\left|\overline{J}\right| \leq m$, then stop.*

1.2. *If no job $j_0 \in \bar{J}$ satisfies condition (2.1), then go to step 2. Else, set $\overline{J} = \overline{J}\backslash \{j_0\}$, $J_R = J_R \cup \{j_0\}$ and go to step 1.1.*

Step 2. Test on condition (2.2)

2.1. *If $\left|\overline{J}\right| \leq m$, then stop.*

2.2. *If no job $j_0 \in \bar{J}$ satisfies condition (2.2), then go to step 3. Else, set $\overline{J} = \overline{J}\backslash \{j_0\}$, $J_Q = J_Q \cup \{j_0\}$ and go to step 2.1.*

Step 3. *If no job is removed in step 2, then stop. Else, go to step 1.*

Ranking jobs according to r_j, $r_j + p_j$, q_j, and $p_j + q_j$ requires $O(n \log n)$ time. There are at most $n - m$ iterations. Thus the complexity of the preprocessing algorithm is $O(n \log n)$.

Example 1. Consider the 9 job-2 machine instance defined by Table 1. The main steps of the preprocessing algorithm are the following.

Iteration 1.

Step 1. $J_{r,m} = \{9, 1\}$, $j_0 = 9$ and $j_{(m+1)} = 3$. Since $r_{j_0} + p_{j_0} = 2$ and $r_{j_{(m+1)}} = 2$, then j_0 satisfies condition (2.1). Therefore, job 9 is removed from $\bar{J}$ and added to J_R. Now, $J_{r,m} = \{1, 3\}$, $j_0 = 1$ and $j_{(m+1)} = 8$. Condition (2.1) is not satisfied by j_0 since $r_{j_0} + p_{j_0} = 3$ and $r_{j_{(m+1)}} = 2$.

Step 2. $J_{q,m} = \{8, 6\}$, $j_0 = 8$ and $j_{(m+1)} = 7$. Since $q_{j_0} + p_{j_0} = 7$ and $q_{j_{(m+1)}} = 7$, then j_0 satisfies condition (2.2). Therefore, job 8 is removed from $\bar{J}$ and added to J_Q. Similarly, jobs 6 and 7 are successively transferred from $\bar{J}$ to J_Q.

Iteration 2.

Step 1. $J_{r,m} = \{1, 3\}$, $j_0 = 1$ and $j_{(m+1)} = 2$. Since $r_{j_0} + p_{j_0} = 3$ and $r_{j_{(m+1)}} = 4$, then j_0 satisfies condition (2.1). Therefore, job 1 is removed from $\bar{J}$ and added to J_R. The next job satisfying condition (2.1) is job 2 which is moved to J_R.

Step 2. $J_{q,m} = \{4, 5\}$, $j_0 = 5$, $j_{(m+1)} = 3$ and $q_{j_0} + p_{j_0} = 16 > q_{j_{(m+1)}} = 15$. No job is added to J_Q. Therefore, the algorithm stops.

Hence, the preprocessing algorithm outputs: $J_R = \{9, 1, 2\}$, $J_Q = \{8, 6, 7\}$, $\bar{J} = \{3, 4, 5\}$.

insert Table 1 here

2.2 Global Schedule Construction Algorithm

After performing the preprocessing algorithm, the set J is partitioned into three subsets : J_R, J_Q and $\overline{J}$. A consequence of Lemmas 1 and 2 is that :

$$C^*_{\max}(J) = \max\left\{C^*_{\max}(\overline{J}), \max_{j\in J_R\cup J_Q}(r_j + p_j + q_j)\right\}$$

Assume that we solve the problem on the jobset $\overline{J}$, and that σ is a schedule of $\overline{J}$ with makespan equal to $C_{\max}(\overline{J})$. The algorithm described below integrates the jobs of J_R and J_Q in the schedule σ in order to construct a global schedule for the entire jobset J. We refer to this algorithm as the Global Schedule Construction Algorithm ($GSCA$). Let $u_i(\sigma)$ denote the availability time of the machine m_i $(i = 1, ..., m)$ in the schedule σ, and let σ^{-1} be the symmetric schedule of σ.

Global Schedule Construction Algorithm ($GSCA$).

Step 1.

1.1 *If $J_Q = \emptyset$, then go to step 2. Else, let $j_0 \in J_Q$ be the job such that $p_{j_0} + q_{j_0} = \max_{j\in J_Q}(p_j + q_j)$.*

1.2 *Let m_0 be the machine with availability time $u_0 = \min_{i=1,\ldots,m} u_i(\sigma)$. Schedule job j_0 on m_0 and set $u_0 = \max(u_0, r_{j_0}) + p_{j_0}$.*

1.3 *Set $J_Q = J_Q \backslash \{j_0\}$ and go to step 1.1.*

Step 2. *Set $\sigma = \sigma^{-1}$.*

Step 3.

3.1 *If $J_R = \emptyset$, then go to step 4. Else, let $j_0 \in J_R$ be the job such that $r_{j_0} + p_{j_0} = \max_{j \in J_R} (r_j + p_j)$.*

3.2 *Let m_0 be the machine with availability time $u_0 = \min_{i=1,\ldots,m} u_i(\sigma)$. Schedule job j_0 on m_0 and set $u_0 = \max(u_0, q_{j_0}) + p_{j_0}$.*

3.3 *Set $J_R = J_R \backslash \{j_0\}$ and go to step 3.1.*

Step 4. *Set $\sigma = \sigma^{-1}$.*

Lemma 3. *Given J_R, J_Q and σ with makespan $C_{\max}(\overline{J})$, the GSCA yields a schedule with makespan equal to*

$$C_{\max}(J) = \max\left\{C_{\max}(\overline{J}), \max_{j \in J_R \cup J_Q} (r_j + p_j + q_j)\right\}$$

Proof. First, we prove that if a job j_1 is removed before job j_2 from $\bar{J}$ according to condition *(2.2),* then we have $p_{j_1} + q_{j_1} \leq p_{j_2} + q_{j_2}$.
Indeed, we have $p_{j_1} + q_{j_1} = \min_{j \in \bar{J}_{q,m}} (p_j + q_j)$ and $p_{j_1} + q_{j_1} \leq q_{j_{(m+1)}}$.
On the other hand, we have

$$\begin{aligned} p_{j_2} + q_{j_2} &= \min_{j \in \bar{J}_{q,m} \backslash \{j_1\} \cup \{j_{(m+1)}\}} (p_j + q_j) \\ &= \min\{\min_{j \in \bar{J}_{q,m} \backslash \{j_1\}} (p_j + q_j), p_{j_{(m+1)}} + q_{j_{(m+1)}}\} \\ &\geq p_{j_1} + q_{j_1}. \end{aligned}$$

Consequently, the last job removed from $\bar{J}$ and put in J_Q is the job j_0 with maximal $p_j + q_j$, and we have $p_{j_0} + q_{j_0} \leq q_{j_{(m)}}$, where $j_{(m)}$ is the job of $\bar{J}$ with the m^{th} smallest tail. If j_0 is scheduled on m_0 (the machine with minimal availability time u_0) at the last position, then two cases are possible :

1- j_0 starts at r_{j_0} : It will therefore complete at $r_{j_0} + p_{j_0} + q_{j_0} \leq \max_{j \in J_Q} (r_j + p_j + q_j)$.

2- j_0 starts at u_0 : Note that there are at least m jobs in $\bar{J}$. Thus, there must exist a machine m_k in σ on which the last scheduled job j_k has a tail greater than or equal to $q_{j_{(m)}}$. We have

$$C_{\max}(\bar{J}) \geq u_k(\sigma) + q_{j_k} \geq u_0 + q_{j_{(m)}} \geq u_0 + p_{j_0} + q_{j_0}$$

Therefore, in both cases, fixing j_0 on machine m_0 yields a schedule with makespan not greater than $C_1 = \max\left\{C_{\max}(\bar{J}), \max\limits_{j \in J_Q}(r_j + p_j + q_j)\right\}$. By recursion, it is possible to schedule the jobs of J_Q according to non-increasing $p_j + q_j$ in such a way that the maximum completion time of the obtained schedule is C_1. By symmetry, the jobs of J_R can be scheduled in the same way. The maximum completion time of the resulting schedule is therefore

$$C_{\max}(J) = \max\left\{C_{\max}(\overline{J}),\ \max_{j \in J_R \cup J_Q}(r_j + p_j + q_j)\right\}. \blacksquare$$

Example 1 :(continued) Consider the instance defined by Table 1. We have $J_R = \{9, 1, 2\}$, $J_Q = \{8, 6, 7\}$ and $\overline{J} = \{3, 4, 5\}$. Consider the schedule of $\overline{J}$ with makespan equal to 25 depicted in Figure 1.

insert Figure 1 here

The *GSCA* yields a schedule of J, depicted in Figure 2, with makespan equal to 25.

insert Figure 2 here

An immediate consequence of Lemma 3 is the following corollary.

Corollary 1

If σ^* is an optimal schedule for $\overline{J}$ then GSCA yields an optimal schedule for J.

3 Lower bounds

3.1 Lower bounds from the literature

An obvious lower bound, which can be computed in $O(n)$ time, is:

$$LB_0(J) = \max_{j \in J}(r_j + p_j + q_j)$$

Another simple lower bound given in [6] is :

$$LB_1(J) = \min_{j \in J} r_j + \left\lceil \frac{1}{m} \sum_{j \in J} p_j \right\rceil + \min_{j \in J} q_j$$

This bound is based on the fact that all release dates and delivery times are assumed to be equal to $\min\limits_{j \in J} r_j$ and $\min\limits_{j \in J} q_j$, respectively. $LB_1(J)$ can be computed in $O(n)$.

LB_1 is still a lower bound even when it is computed over a subset of jobs. If this bound is computed over all possible subsets of jobs, including the total set, then the following lower bound which dominates LB_1 is obtained :

$$SLB_1(J) = \max_{S \subseteq J} LB_1(S)$$

In [6], it is proved that $SLB_1(J)$ is equal to the optimal makespan of the preemptive one machine problem $1/r_j, q_j, pmtn/C_{\max}$, obtained by replacing the m machines by one and dividing all the processing times by m. This problem is solved using Jackson's preemptive algorithm (see [11]). Therefore, $SLB_1(J)$ can be computed in $O(nlogn)$ time.

The following $O(n)$ lower bound, proposed in [6], tries to take into account release dates and delivery times in a more effective way :

$$LB_2(J) = \left\lceil \frac{1}{m}\left(\overline{r}_1 + \overline{r}_2 + \cdots + \overline{r}_m + \sum_{j=1}^{n} p_j + \overline{q}_1 + \overline{q}_2 + \cdots + \overline{q}_m\right)\right\rceil$$

where $\overline{r}_k$ and $\overline{q}_k$ denote the k^{th} smallest release date and delivery time in J, respectively. Interestingly, we prove that the preprocessing algorithm improves LB_2.

Proposition 1 : $LB_2(\overline{J}) \geq LB_2(J)$

Proof. It suffices to prove that each removal of a job $j_0 \in J_R$ from J improves the bound LB_2. By definition, job j_0 belongs to $J_{r,m}$, where $J_{r,m}$ is the set of jobs in J with the m smallest heads. Let $J'_{r,m}$, $J'_{q,m}$ and J' be the respective updated sets $J_{r,m}$, $J_{q,m}$ and J if job j_0 is removed from J. Then,

$$\sum_{j \in J'_{r,m}} r_j + \sum_{j \in J'} p_j = \sum_{j \in J_{r,m}} r_j - r_{j_0} + r_{j_{(m+1)}} + \sum_{j \in J} p_j - p_{j_0}$$

Since $r_{j_0} + p_{j_0} \leq r_{j_{(m+1)}}$, then

$$\sum_{j \in J'_{r,m}} r_j + \sum_{j \in J'} p_j \geq \sum_{j \in J_{r,m}} r_j + \sum_{j \in J} p_j$$

On the other hand, $\sum_{j \in J'_{q,m}} q_j \geq \sum_{j \in J_{q,m}} q_j$. Therefore,

$$LB_2(J \backslash \{j_0\}) \geq LB_2(J)$$

Consequently, $LB_2(J \backslash J_R) \geq LB_2(J)$. Similarly, it can be proven that $LB_2(J \backslash J_Q) \geq LB_2(J)$. By applying the last inequality to $J \backslash J_R$ we obtain

$$LB_2(\overline{J}) = LB_2((J \backslash J_R) \backslash J_Q) \geq LB_2(J \backslash J_R) \geq LB_2(J) \blacksquare$$

It is noteworthy that an $O(mn^2)$ algorithm was proposed by Vandevelde [17] to compute the tight lower bound

$$SLB_2(J) = \max_{S \subseteq J} LB_2(S)$$

The following proposition shows that SLB_2 can be computed more efficiently if the subset $\overline{J}$ is considered instead of J.

Proposition 2 : $SLB_2(\overline{J}) = SLB_2(J)$.

Proof. Note that $LB_2(\overline{S}) \geq LB_2(S)$ for each set S containing any job of $J \backslash \overline{J}$. Therefore, there is necessarily a set of jobs $J^* \subseteq \overline{J}$ such that $LB_2(J^*) = SLB_2(J)$. Since $J^* \subseteq \overline{J} \subseteq J$, then

$$LB_2(J^*) \leq SLB_2(\overline{J}) \leq SLB_2(J)$$

Consequently, by definition of J^*, we have $SLB_2(J) = SLB_2(\overline{J})$. ■

The Jackson's Pseudo-Preemptive Schedule ($JPPS$) was introduced in [18]. It was shown that it provides an $O(n \log n + nm \log m)$ lower bound for the $P/r_j, q_j/C_{\max}$. In a pseudo-preemptive schedule, any operation is allowed to be preempted, and each machine can handle several jobs at one time. Moreover, each job can be processed by more than one machine at one time. The $JPPS$ is a generalization of Jackson's preemptive schedule whose makespan is a tight lower bound for $1/r_j, q_j/C_{\max}$. It is noteworthy that, contrary to the fact that Jackson's preemptive schedule is optimal for the one machine preemptive problem, the $JPPS$ *does not* provide the optimal solution for the m machine pseudo-preemptive problem. We denote by $JPPB$ the value $\lceil C_{\max}(JPPS) \rceil$ where $C_{\max}(JPPS)$ denotes the makespan of $JPPS$. It has been proven that $JPPB(J) = \max \{LB_0(J), SLB_2(J)\}$ [18].

3.2 New lower bounds

In this section, we introduce a bounding scheme based on a new lower bound referred to hereafter as the *Modified General Bound* (MGB). This bound is derived from the so-called *General Bound* (GB) proposed by Webster [19] for the parallel machine scheduling problem with availability constraints denoted by $P, NC_{inc}//C_{\max}$ [20]. In this case, a machine m_i ($i = 1, ..., m$) is available from time $u_i \geq 0$ onwards. For a review of scheduling problems with limited machine availability the reader is referred to Schmidt [20].

A relaxation of $P/r_j, q_j/C_{\max}$ as a $P, NC_{inc}//C_{\max}$ may be obtained as follows. Assume that all release dates and delivery times are set equal to zero, and that each machine m_i has an available time $u_i = \overline{r}_i$ ($i = 1, ..., m$), where $\overline{r}_i$ denotes the i^{th} smallest release date in J. In the remainder of this subsection, it is assumed without loss of generality that jobs are ranked in non-increasing order of their processing times, and that machines are ranked in non-decreasing order of their availability times.

3.2.1 The Modified General Bound

First we present the General Bound (GB) which is based on the two following theorems stated by Webster in [19] for the $P, NC_{inc}//C_{\max}$ problem. The first theorem gives sufficient conditions for the optimality of the *Longest Processing Time (LPT)* rule which schedules the job with the longest processing time on the first available machine. The second gives a sufficient condition for checking the optimality of any non-delay schedule (i.e. a schedule where no machine is kept idle while a job is waiting).

Theorem 1

If one of the four following conditions is verified, then the LPT rule is optimal.

$\mathbf{C}_1 : p_j/p_{j+1}$ *is integer for* $j = 1, ..., n-1$.

$\mathbf{C}_2 : n \leq m$ *and* $u_n - u_1 \leq p_j$ *for* $j = 1, ..., n-1$.

$\mathbf{C}_3 : u_i = 0$ *for all* $i = 1, ..., m$, *and* $p_{m-1} \geq p_m + \cdots + p_{n-2}$.

$\mathbf{C}_4 : u_i = 0$ *for all* $i = 1, ..., m$; $n \leq 2m$ *and* $p_{2m-n+1} \leq 2p_n$.

Theorem 2

Let $s_1, ..., s_n$ *be the job starting times in a non-delay schedule* σ *for a* $P, NC_{inc}//C_{\max}$ *problem, and let* j_0 *be the last job to finish processing. If* p_j/p_{j_0} *is integer for all* j *such that* $s_j < s_{j_0}$, *then* σ *is optimal.*

Consider two positive integers n' and ρ, and let τ be a common divisor of $\rho, (p_1 - \left\lfloor \frac{p_1}{\rho} \right\rfloor \rho), ..., (p_n - \left\lfloor \frac{p_n}{\rho} \right\rfloor \rho)$. A relaxed problem of $P, NC_{inc}//C_{\max}$ can be obtained if the set J of n jobs is replaced by $J'_\rho = J_1 \cup J_2 \cup J_3$ such that :

- J_1 is a set of n' ($n' \leq n$) jobs with respective processing times $\left\lfloor \frac{p_1}{\rho} \right\rfloor \rho$, $\left\lfloor \frac{p_2}{\rho} \right\rfloor \rho, ...,$ $\left\lfloor \frac{p_{n'}}{\rho} \right\rfloor \rho$.
- J_2 is a set of $\left\lfloor \frac{p_{n'+1}}{\rho} \right\rfloor + \cdots + \left\lfloor \frac{p_n}{\rho} \right\rfloor$ jobs with processing time ρ.
- J_3 is a set of $(p_1 - \left\lfloor \frac{p_1}{\rho} \right\rfloor \rho)/\tau + \cdots + (p_n - \left\lfloor \frac{p_n}{\rho} \right\rfloor \rho)/\tau$ jobs with processing time τ.

Denote by $P(J'_\rho)$ the $P, NC_{inc}//C_{\max}$ problem defined on J'_ρ. The optimal makespan of $P(J'_\rho)$ is defined as a general lower bound on the optimal one of $P(J)$. It will be denoted by $GB_\rho(J)$.

The number n' is selected such that $P(J_1)$ can be optimally solved using LPT rule. Consequently, the problem $P(J'_\rho)$ can also be optimally solved using LPT rule. In fact, consider the schedule σ obtained using LPT rule on the set J'_ρ, and denote by j_0 the last job to finish processing in σ. Two cases are possible :

- The job j_0 is in J_1 : In this case, σ is a feasible schedule of $P(J'_\rho)$ with makespan equal to the one of the LPT schedule of $P(J_1)$. Since $P(J_1)$ is a relaxation of $P(J'_\rho)$, and the LPT schedule is optimal for $P(J_1)$, then σ is optimal.

- The job j_0 is in $J_2 \cup J_3$: Note that, in an LPT schedule, jobs of J_1 are scheduled first, then those of J_2, and finally those of J_3. That is, each job j, starting before j_0 in σ, is such that p_j/p_{j_0} is integer. Therefore, by Theorem 2, we conclude that σ is optimal.

Clearly, $GB_\rho(J)$ is maximized if n' and τ are selected to be as large as possible for a given value of ρ. Consequently, for a given ρ, the number n' is selected to be the largest number for which at least one of the four conditions given by Theorem 1 is verified according to the data of the set J_1. The number τ is set to $\gcd(\rho, (p_1 - \left\lfloor \frac{p_1}{\rho} \right\rfloor \rho), ..., (p_n - \left\lfloor \frac{p_n}{\rho} \right\rfloor \rho))$.

Two values of ρ for which $P(J'_\rho)$ is a very tight relaxation of $P(J)$ were proposed by Webster [19]. These values are $\rho_1 = \gcd(p_1, p_2, ..., p_n)$ and $\rho_2 = p_{j_0}$, where j_0 is the last job to finish processing in the LPT schedule constructed for J. There is no dominance relation between $GB_{\rho_1}(J)$ and $GB_{\rho_2}(J)$. We will refer to the general bound $GB(J)$ as the maximum value between $GB_{\rho_1}(J)$ and $GB_{\rho_2}(J)$. The bound GB can be computed in $O(n \log n)$ time.

In order to introduce other valid lower bounds, we present two lemmas which are consequences of the application of the preprocessing algorithm. These lemmas show that adding some specific sets of dummy jobs has no effect on the optimal makespan.

Lemma 4

Consider the sets $D_1(J)$ and $D_2(J)$, each includes m dummy jobs such that :

$$D_1(J) = \{j_k \ (k = 1, ..., m) \ / \ r_{j_k} = 0, \ p_{j_k} = \overline{r}_k(J) \text{and} \ q_{j_k} = \overline{q}_m(J)\}$$

$$D_2(J) = \{j_k \ (k = 1, ..., m) \ / \ r_{j_k} = \overline{r}_m(J), \ p_{j_k} = \overline{q}_k(J) \text{and} \ q_{j_k} = 0\}$$

where $\overline{r}_k(J)$ and $\overline{q}_k(J)$ denote the k^{th} smallest release date and delivery time in J, respectively.

Then,

$$C^*_{\max}(J) = C^*_{\max}(J \cup D_1(J) \cup D_2(J))$$

Proof. Clearly, jobs of $D_1(J)$ and $D_2(J)$ satisfy the conditions (2.1) and (2.2), respectively. Therefore,

$$C^*_{\max}(J \cup D_1(J) \cup D_2(J)) = \max\{C^*_{\max}(J), LB_0(D_1(J)), LB_0(D_2(J))\}$$

Denote by $\overline{J}$ the set obtained after applying preprocessing algorithm to J, and by $D_1(\overline{J})$ the set of m dummy jobs such that :

$$D_1(\overline{J}) = \left\{ j_k \ (k = 1, ..., m) \ / \ r_{j_k} = 0,\ p_{j_k} = \overline{r}_k(\overline{J}) \text{ and } q_{j_k} = \overline{q}_m(\overline{J}) \right\}$$

Note that $\overline{r}_k(\overline{J}) < \min\limits_{j \in \overline{J}} (r_j + p_j)$ for all $k = 1, ..., m$. Indeed, if there was a job $j_0 \in \overline{J}$ such that $r_{j_0} + p_{j_0} = \min\limits_{j \in \overline{J}} (r_j + p_j)$ and satisfying $r_{j_0} + p_{j_0} \leq \overline{r}_k(\overline{J})$, then j_0 would be in J_R and not in $\overline{J}$.

Let j_0' denote a job of $\overline{J}$ with tail equal to $\overline{q}_m(\overline{J})$. Thus, for every $j_k \in D_1(\overline{J})$ $(k = 1, ..., m)$, we have :

$$\begin{aligned} r_{j_k} + p_{j_k} + q_{j_k} &= \overline{r}_k(\overline{J}) + \overline{q}_m(\overline{J}) \\ &< \min_{j \in \overline{J}} (r_j + p_j) + \overline{q}_m(\overline{J}) \\ &\leq r_{j_0'} + p_{j_0'} + q_{j_0'} \\ &\leq LB_0(\overline{J}) \\ &\leq C^*_{\max}(J) \end{aligned}$$

Therefore, $LB_0(D_1(\overline{J})) < C^*_{\max}(J)$.

On the other hand, we have $LB_0(D_1(J)) \leq LB_0(D_1(\overline{J}))$. So, $LB_0(D_1(J)) < C^*_{\max}(J)$. Similarly, it can be proven that $LB_0(D_2(J)) < C^*_{\max}(J)$. ■

Lemma 5

Consider the sets $D_1'(J)$ and $D_2'(J)$, each includes $m-1$ dummy jobs such that :

$D_1'(J) = \{j_k(k = 1, ..., m-1)/r_{j_k} = \overline{r}_1(J), p_{j_k} = \overline{r}_{k+1}(J) - \overline{r}_1(J) \text{ and } q_{j_k} = \overline{q}_m(J)\}$

$D_2'(J) = \left\{j_k(k = 1, ..., m-1)/r_{j_k} = \overline{r}_m(J), p_{j_k} = \overline{q}_{k+1}(J) - \overline{q}_1(J) \text{ and } q_{j_k} = \overline{q}_1(J)\right\}$

where $\overline{r}_k(J)$ and $\overline{q}_k(J)$ denote the k^{th} smallest release date and delivery time in J, respectively.

Then,

$$C^*_{\max}(J) = C^*_{\max}(J \cup D_1'(J) \cup D_2'(J))$$

Proof. The proof of Lemma 5 is similar to that of Lemma 4 and is therefore omitted. ■

As described above, the $P, NC_{inc}//C_{\max}$ can be formulated as a relaxation of $P/r_j, q_j/C_{\max}$. Therefore, the bound GB is a valid lower bound for $P/r_j, q_j/C_{\max}$. Note that the relaxation scheme takes into consideration the m smallest heads of the jobs. Unfortunately, their tails are totally ignored. An attempt to consider the m smallest tails is to add to the set J, the set $D_2(J)$ of dummy jobs described in Lemma 4. Since the obtained $P/r_j, q_j/C_{\max}$ problem has the same optimal makespan as the initial one, then the bound $GB(J \cup D_2(J))$ remains a lower bound for the $P/r_j, q_j/C_{\max}$ problem defined for J.

Another valid lower bound can be derived if the aforementioned set of dummy jobs $D_2'(J)$ is added to the set J. Note that, in this case, each machine has to wait at least an amount of time equal to $\overline{q}_1(J)$ after finishing processing. Consequently, $GB(J \cup D_2'(J)) + \overline{q}_1(J)$ is a valid lower bound for the $P/r_j, q_j/C_{\max}$ problem defined for J.

It is worth noting that when the Backward problem is considered, then two additional lower bounds $GB^{-1}(J \cup D_1(J))$ and $GB^{-1}(J \cup D_1'(J)) + \overline{r}_1(J)$ can also be derived, where GB^{-1} is the general bound computed for the relaxation of the Backward problem. The following proposition shows that all the four derived general bounds dominate LB_2.

Proposition 3

i) $GB(\overline{J} \cup D_2(\overline{J})) \geq LB_2(\overline{J})$

ii) $GB(\overline{J} \cup D_2'(\overline{J})) + \overline{q}_1(\overline{J}) \geq LB_2(\overline{J})$

iii) $GB^{-1}(\overline{J} \cup D_1(\overline{J})) \geq LB_2(\overline{J})$

iv) $GB^{-1}(\overline{J} \cup D_1'(\overline{J})) + \overline{r}_1(\overline{J}) \geq LB_2(\overline{J})$

Proof. The proof of (*ii*) is similar to the one of (*i*). The proofs of (*iii*) and (*iv*) can be derived by symmetry from those of (*i*) and (*ii*), respectively. Thus, only (*i*) will be proven.

Clearly, for any set S, we have :

$$\left\lceil \frac{1}{m}\left(\overline{r}_1(S) + \overline{r}_2(S) + \cdots + \overline{r}_m(S) + \sum_{j \in S} p_j\right)\right\rceil \leq GB(S)$$

If the latter inequality is applied to the set $\overline{J} \cup D_2(\overline{J})$, then we obtain :

$$\left\lceil \frac{1}{m}\left(\sum_{k=1}^{m} \overline{r}_k(\overline{J}) + \sum_{j \in \overline{J}} p_j + \sum_{k=1}^{m} \overline{q}_k(\overline{J})\right)\right\rceil \leq GB(\overline{J} \cup D_2(\overline{J}))$$

That is, $LB_2(\overline{J}) \leq GB(\overline{J} \cup D_2(\overline{J}))$. ■

Note that there is no dominance relation between the four general bounds mentioned above. Our experimental results show that these bounds are equal in nearly all cases with a slight advantage for the general bound $GB(\overline{J} \cup D_2'(\overline{J})) + \overline{q}_1(\overline{J})$. This bound will be referred to as the *Modified General Bound,* and will be denoted by $MGB(J)$. An immediate consequence of Proposition 3 is the following corollary :

Corollary 2

Denote by J^* the set of jobs such that $LB_2(J^*) = SLB_2(\overline{J})$, then

$$MGB(J^*) \geq SLB_2(\overline{J})$$

It is noteworthy that two additional variants of $MGB(J)$ can be constructed :

$i)$ By considering the set J instead of $\overline{J}$.

$ii)$ By computing the general bound without adding dummy jobs.

The resulting bound does not dominate the bound $LB_2(J)$. Moreover, our experimental results show that $MGB(J)$ consistently dominates its variants.

3.2.2 The Subset Modified General Bound

Another consequence of Proposition 3 is that it would be interesting to compute the value of $\max_{S \subseteq \overline{J}} MGB(S)$. We propose a greedy algorithm, described below, to compute an approximation of this value. The resulting bound will be denoted by the *Subset Modified General Bound* ($SMGB$). Denote by S^* the set such that $MGB(S^*) = SMGB(J)$. In the computation of $SMGB(J)$, the set S^* is initialized to $\overline{J}$. At each iteration, a particular job j_0 is removed from the current set S^*. The job j_0 is the job whose removal yields the most improvement to $MGB(S^*)$. We iterate until no improvement is found. Note that it is useless to continue the procedure if $|S^*| \leq m+1$. Indeed, each removal of a job will lead, at best, to a value equal to $LB_0(J)$. A formal description of the algorithm is as follows: The jobs $j_1, j_2, ..., j_{|S|}$ denote the different jobs of the set S.

Computation of $SMGB(J)$

Step 0. *Set $S^* = \overline{J}$ and $SMGB(J) = MGB(\overline{J})$.*

Step 1.

1.1 *If $|S^*| \leq m+1$, then stop. Else, set $k = 1$ and $LB = 0$.*

1.2 *Set $S = S^* \backslash \{j_k\}$. If $MGB(\overline{S}) > LB$, then set $LB = MGB(\overline{S})$ and $j_0 = j_k$.*

1.3 *Set $k = k + 1$. If $k \leq |S^*|$, then go to step 1.2.*

Step 2. *If $LB \leq SMGB(J)$ then stop. Else, set $SMGB(J) = LB$, $S^* = S^* \backslash \{j_0\}$ and go to step 1.*

The bound $SMGB(J)$ can be computed in $O(n^2 \log n)$ time.

Example 2 : Consider the 7 job - 3 machine instance defined by Table 2.

insert Table 2 here

We have $\bar{J} = J$ and $D_2'(\bar{J}) = \{j_1, j_2\}$ such that $p_{j_1} = p_{j_2} = 7$. The $P, NC_{inc}//C_{\max}$ problem defined on $\bar{J} \cup D_2'(\bar{J})$ is such that the machine availabilities are $\{1, 1, 1\}$. For $\rho_2 = 7$, we have $n' = 7$, $J_2 = \emptyset$, $\tau = 1$, $|J_3| = 11$ and $MGB(J) = 23$ (see Figure 3a).

insert Figure 3a here

Let MGB be computed over the set $S = J \backslash \{7\}$. We have $\bar{S} = S$ and $D_2'(\bar{S}) = \emptyset$. In the corresponding $P, NC_{inc}//C_{\max}$ problem, the machine availabilities are $\{1, 1, 1\}$. For $\rho_1 = 1$, we have $n' = 5$, $|J_2| = 1$, $J_3 = \emptyset$ and $MGB(S) = 24$ (see Figure 3b). This value corresponds to $SMGB(J)$.

insert Figure 3b here

Note that $LB_2(J) = SLB_2(J) = 22$ and the optimal makespan is equal to 24 (see Figure 3c).

insert Figure 3c here

4 Adjustment of heads and tails and feasibility tests

Assume that we are given a $P/r_j, q_j/C_{\max}$ instance and denote by LB and UB a lower and an upper bound on its optimal solution, respectively. Let $\bar{J}$ be the set of jobs obtained after performing the preprocessing algorithm. We associate a deadline $d_j = UB - q_j$ with each job $j \in \bar{J}$. In this section, we develop several rules which provide sufficient conditions to prove that there is no feasible schedule with makespan less than or equal to UB. The importance of these rules is twofold. They are used in the B&B algorithm for discarding infeasible nodes, and they permit the adjustment of the heads and tails, so that the lower bounds are tightened. The proposed rules are based on the following observation which provides a simple way to compute a lower bound on the number of machines which are necessarily loaded at any time.

Observation 2

Assume that there exists a job j such that $d_j - r_j < 2p_j$. Then, in any non-preemptive schedule, there is necessarily one machine which has to process job j during the interval $[d_j - p_j; r_j + p_j]$.

Proof. It suffices to remark that the latest starting time of job j is $d_j - p_j$, and its earliest finishing time is $r_j + p_j$. ■

Clearly, the following feasibility condition holds.

Condition 4.1 : *The instance is infeasible if there is a time $t \in [0, UB]$ such that the number of machines loaded at t is strictly greater than m.*

Let $\pi = \left\{j \in \bar{J}; d_j - r_j < 2p_j\right\}$. For each job $j \in \pi$, we refer to its *fixed* part the amount of time $2p_j - (d_j - r_j)$ which has to be processed in $[d_j - p_j, r_j + p_j]$, and to its *free* part the amount of time $p'_j = d_j - (r_j + p_j)$ which has to be processed in $[r_j, d_j - p_j] \cup [r_j + p_j, d_j]$. Note that each job of $\bar{J} \backslash \pi$ is composed only of a free part $p'_j = p_j$ which has to be processed in $[r_j, d_j]$. That is, a free part of a job $j \in \bar{J}$ is :

$$p'_j = \begin{cases} d_j - (r_j + p_j) & \text{if } d_j - r_j < 2p_j \\ p_j & \text{otherwise} \end{cases}$$

Therefore $p'_j = \min\left\{p_j, d_j - (r_j + p_j)\right\}$ for all $j \in \bar{J}$. Let $e_1, e_2, ..., e_H$ be the different values of r_j, d_j, $d_j - p_j$ and $r_j + p_j$ $(j \in \bar{J})$ ranked in increasing order. For each interval $E_h = [e_h, e_{h+1}]$ $(1 \leq h \leq H - 1)$, we have the following notations :

- m_h the number of machines which are idle during the time interval E_h.
- J_h the set of jobs which free parts may be processed during the time interval E_h.
- $n_h = |J_h|$.

Clearly, the amount of work in E_h $(h = 1, ..., H - 1)$ cannot exceed

$$A_h = \min\left\{\sum_{j \in J_h} p'_j \ ; \ (e_{h+1} - e_h) \times \min(n_h, m_h)\right\}$$

The infeasibility holds if the total time needed to process all the free parts exceeds the total amount of work possible in the intervals. This leads to the following feasibility condition.

Condition 4.2 : *The instance is infeasible if* $\sum_{h=1}^{H-1} A_h < \sum_{j\in\bar{J}} p'_j$.

A *time window* is defined as an interval of time $[a_k, b_k] \subseteq [0, UB]$, in which there is at least one idle machine. Let W denote the set of the different time windows, and let $w = |W|$. Provided the number of loaded machines at any time, the set W can be easily derived. These time windows will be used in order to adjust heads and tails of any job $j_0 \in \bar{J}\backslash\pi$. The job j_0 can start processing at r_{j_0} in a feasible schedule if there exists a time window $[a_k, b_k]$ $(k = 1, ..., w)$ such that $[r_{j_0}, r_{j_0} + p_{j_0}] \subseteq [a_k, b_k]$. That is, $a_k \leq r_{j_0} < r_{j_0} + p_{j_0} \leq b_k$. The earliest starting time of job $j_0 \in \bar{J}\backslash\pi$ may be computed by the following algorithm.

Adjusting heads

Step 1. *Find the maximal* k $(1 \leq k \leq w)$ *such that* $a_k \leq r_{j_0}$.

Step 2. *While* $(r_{j_0} + p_{j_0} > b_k)$, *Begin Set* $k = k + 1$, $r_{j_0} = a_k$, *End (While).*

Similarly, job $j_0 \in \bar{J}\backslash\pi$ can finish processing at d_{j_0} in a feasible schedule if there exists k $(k = 1, ..., w)$ such that $a_k \leq d_{j_0} - p_{j_0} < d_{j_0} \leq b_k$. In this case also, it is possible to adjust d_{j_0} to the latest possible finishing time using a similar procedure.

Adjusting deadlines and tails

Step 1. *Find the minimal* k $(1 \leq k \leq w)$ *such that* $b_k \geq d_{j_0}$.

Step 2. *While* $(d_{j_0} - p_{j_0} < a_k)$, *Begin Set* $k = k - 1$, $d_{j_0} = b_k$, $q_{j_0} = \max(q_{j_0}, LB - d_{j_0})$ *End (While).*

Clearly, if $w = 1$, then the above adjustments are not possible. It is worth noting that the heads and tails of a job $j_0 \in \pi$ can also be adjusted using a similar approach. However, no adjustment holds if $|\pi| < m$. Note that a job $j_0 \in \pi$ such that $r_{j_0} + p_{j_0} = d_{j_0}$ has a fixed starting time r_{j_0} and a fixed finishing time d_{j_0} in any feasible schedule. Thus, no adjustments have to be performed on such a job. After performing these adjustments a third feasibility condition is the following :

Condition 4.3 : *If there is a job* $j_0 \in \bar{J}$ *such that* $r_{j_0} + p_{j_0} > d_{j_0}$, *then the instance is infeasible.*

The algorithm described below and denoted by *Feasibility and Adjustment Procedure* (FAP) is designed in order to check the feasibility of any instance and to adjust the heads and tails of the jobs.

FAP description

Step 1. *Determine the set π. If $|\pi| < m$, then stop.*

Step 2. *If Condition 4.1 is satisfied, then the instance is infeasible. Stop.*

Step 3. *If Condition 4.2 is satisfied, then the instance is infeasible. Stop.*

Step 4. *Determine the set W of time windows. If $w = 1$, then stop.*

Step 5. *Adjust the head, deadline and tail of every job $j \in \bar{J}$.*

Step 6. *If Condition 4.3 is satisfied, then the instance is infeasible. Stop.*

Step 7. *If there is no adjustment, then stop. Else, update $\bar{J}$ and go to step 1.*

Example 3 : Consider the 6 job-3 machine instance defined by Table 3. Assume that we are given $LB = 19$ and $UB = 20$. The deadlines are set to:

$$d_1 = 13,\ d_2 = 14,\ d_3 = 15,\ d_4 = 10,\ d_5 = 12,\ d_6 = 16$$

At the first iteration, we have $\bar{J} = J$ and $\pi = \{2, 5, 6\}$. Using Observation 2, there must exist one machine loaded with job 2 during the interval $[5, 12]$, a second one loaded with job 5 during the interval $[3, 11]$, and a third one loaded with job 6 during the interval $[9, 15]$. Hence, the three machines are necessarily loaded during the interval $[9, 11]$. Therefore, we have $W = \{[0, 9]\,; [11, 20]\}$.

Note that job 1 is released at $r_1 = 10$ but cannot start processing before time 11. Consequently, its release date is adjusted to $r_1 = 11$. Moreover, job 3 has a deadline equal to $d_3 = 15$ but cannot be processed in the time window $[11, 20]$ since $d_3 - p_3 = 8 < 11$. Therefore, its deadline and tail are adjusted to $d_3 = 9$ and $q_3 = 10$, respectively. In a similar way, the deadline of job 4 is adjusted to $d_4 = 9$.

At the second iteration, we have $\bar{J} = \{2, 3, 4, 5, 6\}$, $J_Q = \{1\}$ and $\pi = \{2, 3, 4, 5, 6\}$. Using Observation 2, there must exist one machine loaded with job 2 during the interval $[5, 12]$, a second one loaded with job 3 during the interval $[2, 8]$, a third one loaded with job 4 during the interval $[7, 8]$ and a fourth one loaded with job 5 during the interval $[3, 11]$. Therefore, there must exist *four* machines loaded during the interval $[7, 8]$. Consequently, the instance is identified to be infeasible (cf. Condition 4.1).

insert Table 3 here

Clearly, the adjustment of heads and tails may improve the tightness of the bounds described in section 3. Moreover, after applying the FAP, it is possible to improve a lower bound in the following way. Initially, the lower bound is set to $LB = \max\{LB, LB_0(J)\}$. If the FAP proves that there is no feasible schedule with makespan less than or equal to a trial value $C \in [LB, UB]$, then the lower bound is updated to $C + 1$. Therefore, a FAP based lower bound, denoted FAP_LB, may be obtained by performing a bisection search on the interval $[LB, UB]$. For that purpose, we propose the following algorithm. The interval $[C^-, C^+]$ denotes the current search interval of C.

Computation of FAP_LB

Step 0. *Set* $LB = \max\{LB, LB_0(J)\}$, $C^- = LB$, $C^+ = UB$ *and* $FAP\_LB = C^-$.
Step 1. *Set* $C = \left\lfloor \frac{C^- + C^+}{2} \right\rfloor$.
Step 2. *For all* $j \in \bar{J}$, *set* $d_j = \min\{d_j, C - q_j\}$ *and* $q_j = \max\{q_j, FAP\_LB - d_j\}$.
Step 3. *Apply FAP to the instance obtained in step 2. If the instance is infeasible, then set* $C^- = C + 1$ *and* $FAP\_LB = C^-$. *Else, set* $C^+ = C$.
Step 4. *If* $C^- = C^+$, *then stop. Else, go to step 1.*

5 A time windows based branch and bound algorithm

5.1 Data representation

With each node N of the B&B tree is associated a data set containing for each job j its processing time p_j, a lower bound $LB(N)$, an upper bound $UB(N)$, and a time window $[r_j(N), d_j(N)]$, where $r_j(N)$ and $d_j(N)$ represent the release date and the deadline of job j, respectively. The deadline is initially set to $d_j(N) = UB - q_j(N)$, where $q_j(N)$ is the tail of job j. Clearly, any feasible schedule has a makespan less than or equal to UB if the jobs are completed before their respective deadlines. Node N represents the feasible schedules which satisfy the condition that each job is processed within its corresponding time window. Moreover, the application of the preprocessing algorithm to the instance defined by the node N, generates three sets $J_R(N)$, $J_Q(N)$ and $\overline{J}(N)$ (following the notation of section 2).

5.2 Upper bound

For each node N an upper bound $UB(N)$ is computed in the following way. Firstly, note that the makespan of the schedule constructed by $GSCA$ after applying Jackson's algorithm to $\overline{J}$ is not necessarily better than Jackson's schedule obtained on J. Therefore, we implement two approximate schedules provided by Jackson's algorithm applied to the sets J and $\overline{J}$, respectively. Clearly, there is no dominance between the two approximations. Two more approximate schedules may be obtained by considering the Backward instance. We denote by $JS(I)$ (respectively, $JS^{-1}(I)$), the makespan of the schedule obtained by $GSCA$ after

Jackson's schedule (respectively, the Backward Jackson's schedule) has been constructed for a set of jobs $I \subseteq J$. The value of the upper bound computed for the node N is :

$$UB(N) = \min\left\{JS(J), JS(\overline{J}(N)), JS^{-1}(J), JS^{-1}(\overline{J}(N))\right\}$$

We denote by UB the minimum value of $UB(N)$ over all nodes of the tree.

5.3 Lower bound

We implemented our algorithm in two versions using two different lower bounds. The first version of our algorithm has been implemented using, at each node N, the lower bound:

$$LB(N) = \max\{LB_0(J), JPPB(\overline{J}(N))\}$$

In the second version, the implemented bound is:

$$LB(N) = \max\{LB_0(J), SMGB(\overline{J}(N))\}$$

Clearly, the computation of $SMGB$ is interrupted whenever a subset S satisfying $MGB(S) \geq UB$ is found. For both versions, we compute first $LB_0(J)$, and the corresponding bound is computed only if $LB_0(J) < UB$. Otherwise, the node is pruned. A third version of the algorithm has been implemented using the lower bound $MGB(J^*)$, where J^* denotes the subset such that $LB_2(J^*) = SLB_2(\bar{J}(N))$. Surprisingly, the performance of this version was similar to that of the first version.

5.4 Branching rules

At each iteration of the B&B algorithm, we select a node N with a minimal lower bound. The tail $q_j(N)$ of every job j is then adjusted as follows :

$$q_j(N) = \max\{q_j(N), LB(N) - d_j(N)\} \tag{5.1}$$

During the computations, whenever an improved upper bound UB is found, all the deadlines are adjusted as follows :

$$d_j(N) = \min\{d_j(N), UB - q_j(N)\} \tag{5.2}$$

For each job j we define a margin M_j such that :

$$M_j = d_j(N) - r_j(N) - p_j$$

Note that in order to process a job j within its time window, its start time t_j must lie in the interval $I_j = [r_j(N), d_j(N) - p_j]$. For branching, a job j^* is selected (the selection criterion is discussed below), and the corresponding interval I_{j^*} is split into two disjoint intervals $I^1_{j^*} = [r_{j^*}(N), r_{j^*}(N) + \lceil M_{j*}/2 \rceil - 1]$ and $I^2_{j^*} = [r_{j^*}(N) + \lceil M_{j*}/2 \rceil, d_{j^*}(N) - p_{j^*}]$. Thus, two new nodes N_1 and N_2 are created. For node N_1 we define :

$$r_{j*}(N_1) = r_{j*}(N)$$

$$d_{j*}(N_1) = r_{j*}(N) + \lceil M_{j*}/2 \rceil + p_{j*} - 1$$
$$q_{j*}(N_1) = \max\{q_{j*}(N), LB(N) - d_{j*}(N_1)\}$$

For node N_2 we define :

$$r_{j*}(N_2) = r_{j*}(N) + \lceil M_{j*}/2 \rceil$$
$$d_{j*}(N_2) = d_{j*}(N)$$
$$q_{j*}(N_2) = q_{j*}(N)$$

All the data of jobs in $J \setminus \{j^*\}$ are copied from node N to nodes N_1 and N_2. It is noteworthy that this branching strategy was used in [6]. However, we implemented a slightly different criterion for selecting the job to be branched. Job j^* is selected as follows. Consider a schedule with the minimal makespan among the four ones constructed for node N (section 5.2). Let j_0 be the critical job with maximal tail of this schedule (a critical job is a job with completion time equal to the makespan of the schedule). Assume that it is processed on machine m_0. Let j_1 be the first job processed by m_0 before j_0 and starting at its release date. The critical path is defined as the set containing j_1, j_0 and all the jobs in between that are processed by m_0. Then, the selected job $j^* \in \overline{J}(N)$ is the one with smallest tail among those on the critical path and with tail smaller than q_{j_0}. Where there is no job satisfying this condition, the selected job j^* is the one with the largest margin. It is noteworthy that if job j^* is selected not as the job with the largest margin, then one must make sure that M_{j^*} is not zero. Otherwise, nodes N_1, N_2 and N will be identical and the algorithm will stall.

Finiteness of the B&B algorithm : When all margins are zero, the schedule is fixed and it is easy to check its feasibility. When a margin is negative, then there is no feasible schedule. All margins become zero or negative after at most $\sum_{j \in J} \lceil \log_2(M_j + 1) \rceil$ branchings, where M_j ($j \in J$) are the initial margins. Consequently, the method is finite.

Before computing the lower bounds for N_1 and N_2, one has to perform some preliminary operations. First, the preprocessing algorithm is applied to node N_1(respectively N_2). Then FAP is applied to N_1(respectively N_2) and FAP_LB is computed. In case of feasibility, the upper bound $UB(N_1)$ (respectively $UB(N_2)$) is computed and UB is updated. Finally, the lower bound is computed for a node N only if it was not proven infeasible and $\left|\overline{J}(N)\right| > m$.

5.5 Synthesis of the B&B algorithm

A pseudo-code description of our branch and bound algortihm is the following:

```
Step 0. Initialization
  0.1. Make a root node $R$ containing the data set of the problem.
  0.2. Apply the preprocessing algorithm to node $R$.
  0.3. Compute $UB(R)$. Set $UB = UB(R)$.
  0.4. If $\left|\bar{J}(R)\right| \leq m$, then $C^*_{\max} = UB$. Stop. Else, compute $LB(R)$ and set $\Sigma = \{R\}$ ($\Sigma$ :
       is the set of candidate nodes).
Step 1. Termination test, Node selection and Adjustment
  1.1. If $\Sigma = \emptyset$, then go to step 4. Else, Select a node $N \in \Sigma$ with minimal lower bound.
  1.2. If $LB(N) \geqslant UB$, then go to step 4.
  1.3. Adjust $q_j(N)$ and $d_j(N)$ using equations (5.1) and (5.2) for all $j \in \overline{J}(N)$. If a margin
       is negative, then prune node $N$ and go to step 1.1.
Step 2. Preprocessing and FAP
  2.1. Apply preprocessing algorithm to node $N$.
  2.2. Apply FAP to node $N$, and compute $FAP\_LB(N)$. If $N$ is feasible, $\left|\overline{J}(N)\right| > m$ and
       $FAP\_LB(N) < UB$ then go to step 3. Else, prune $N$ and go to step 1.
Step 3. Branching
  3.1. Select $j^*$ and create nodes $N_1$ and $N_2$. Set $\Sigma = \Sigma \cup \{N_1, N_2\}$.
  3.2. For i = 1,2 do
    3.2.1. Apply preprocessing to $N_i$ and compute $UB(N_i)$.
    3.2.2. If $UB(N_i) < UB$, then set $UB = UB(N_i)$.
    3.2.3. If $\left|\bar{J}(N_i)\right| \leq m$, then prune node $N_i$ and set $\Sigma = \Sigma \backslash \{N_i\}$.
    3.2.4. Compute LB($N_i$). If $LB(N_i) \geq UB$, then prune $N_i$ and set $\Sigma = \Sigma \backslash \{N_i\}$.
    3.2.5. Apply $FAP$ to $N_i$ and compute $FAP\_LB(N_i)$. If $FAP\_LB(N_i) \geq UB$, then
           prune $N_i$ and set $\Sigma = \Sigma \backslash \{N_i\}$.
  3.3. Prune N, set $\Sigma = \Sigma \backslash \{N\}$ and go to step 1.
Step 4. Optimal makespan Set $C^*_{\max} = UB$. Stop.
```

5.6 A Forward-Backward implementation

In order to take advantage of the symmetry of the $P/r_j, q_j/C_{\max}$, the Forward and the Backward problems are solved in parallel. In this way, the convergence of the algorithm may be accelerated through an exchange of information between the two problems. In this section, we propose a *pseudo-parallel* implementation of a B&B algorithm, hereafter called the *Forward-Backward B&B algorithm*. Two search trees are constructed in a way similar to that described in section 5.5. We alternate the exploration of both trees in the following way. Assume that at an iteration, a node from the Forward (Backward) is explored, then at the next iteration a node from the Backward (Forward) is explored. The upper bound UB is taken as the minimal value of $UB(N)$ over all the nodes N of both trees. At each iteration, the lower bound $LB(N)$ of a node N in a given tree is updated to $\max\{LB(N), LB_{\min}\}$, where $LB_{\min}$ is the minimal lower bound over all the nodes of the other tree. The process continues until a schedule is proved optimal.

6 Computational experiments

In this section we present the results of an analysis of the performance of the two B&B algorithms presented in this paper as well as Carlier's algorithm. The following notation is adopted :

- JP : the algorithm described in section 5.5 and based on the lower bound $JPPB$
- SM *:* the algorithm described in section 5.5 and based on the lower bound $SMGB$
- CAR : the algorithm presented by Carlier in [6]

The three algorithms are coded in C and compiled with Visual C++ 5.0. All the computational experiments were carried out on a PentiumIII 733 MHz Personal Computer with 64 MB RAM.

6.1 Test problems

The test problems are generated in a similar way as in [6]. The number of jobs n is taken equal to 50, 100, 150, 200, 250 and 300 jobs. The number of machines m is taken equal to 2, 3, and 4 machines. The processing times are drawn from the discrete uniform distribution on [1,10]. The heads and tails are drawn from the discrete uniform distribution on $[1, K\frac{n}{m}]$, where K is a positive integer set equal to 1, 3, 5, and 7. We combined these problem characteristics to obtain 72 classes of randomly generated test problems. For each class, 10 instances are generated which result in 720 test problems. A CPU time limit of 300 sec. is fixed for each run.

6.2 Performance of the algorithms

The results of our analysis are reported in Tables 4, 5 and 6. For each algorithm, we provide:

- $Time$: mean CPU time (in sec.)
- $Mean\ NN$: mean number of nodes
- $Max\ NN$: maximal number of nodes
- US : percentage of instances for which optimality was not proved after reaching the time limit
- $Mean\ Gap$: average gap of unsolved instances, where the gap is defined as the ratio

$$100(\frac{UB - LB}{LB})$$

- $Max\ Gap$: maximal gap of unsolved instances

Global performance analysis

Table 4 provides strong evidence that the two algorithms developed in this paper consistently outperform Carlier's algorithm (CAR). The fastest algorithm is JP, while SM is the best with respect to the mean number of explored nodes.

We observe that CAR solved only 76.66% of the instances within the time limit of 300 sec., while JP solved 94.17% of the instances within this time limit. The mean CPU time of JP is 18.56 sec., and this algorithm is 3.98 times faster than CAR. Moreover, even the slowest proposed algorithm (SM) is on average more than twice as fast as CAR. Not surprisingly, the smallest computing times are obtained by JP. This algorithm is based on the pseudo-preemptive bound which has a lower complexity than the subset modified general bound.

The difference between our algorithms and CAR is even more dramatic when the mean number of explored nodes criterion is considered. Indeed, CAR explores on average 56.64 times more nodes than SM does. Moreover, the *maximal* number of explored nodes obtained with SM is less than the *mean* number of those obtained with CAR. It is worth noting that SM provided the smallest mean number of explored nodes, which suggests that the lower bound $SMGB$ is the tightest one.

insert Table 4 here

Analysis of the performance with respect to n

The results presented in Table 5 show that large instances with up to 300 jobs are solved within a very moderate CPU time. For instance, 93.33% of the 300-job instances were solved to optimality by JP with a mean CPU time of 22.54 sec. The maximal gap of the unsolved instances is 0.24%. Memory limitations prevent us to go beyond this problem size limit.

Table 5 shows that for all the algorithms the mean (maximal) number of explored nodes and the mean (maximal) gap clearly decrease as n increases. However, the CPU time and the percentage of unsolved instances (US) exhibit a more complex (and correlated) behavior. Indeed, we observe that for all the algorithms (except CAR), the mean computing time decreases when n varies from 50 to 150 and then increases when n varies from 150 to 300. This may be explained by noting that the total computation time is roughly equal to the product of the total number of explored nodes by the computing time required by each node. As it has been noticed above, the total number of explored nodes decreases as n increases. However, the computational burden at each node increases as n increases and this trend becomes dominant for larger values of n.

insert Table 5 here

Analysis of the performance with respect to m and K

Table 6 depicts the behavior of JP when the number of machines m and the value of K vary. It is worth noting that the other two algorithms exhibit a similar behavior. The value of K seems to be the factor which most determines the difficulty of the problems. This result is consistent with the one observed in [6]. When $K \in \{5, 7\}$ (i.e., instances with large heads and tails) the problem can be considered as "easy". We observe that for this problem class all the 360 instances are solved in a negligible CPU time. In contrast, it appears that problems with $K = 1$ and 3 are much harder to solve, and this difficulty strongly increases when m increases. The hardest problem class, for which 33.33% of the instances remained unsolved after reaching the time limit, is obtained with $m = 4$ and $K = 1$. This (relatively) modest performance may be explained by the fact that when $K\frac{n}{m}$ is small, then the heads and tails are also small. Therefore, very few jobs may satisfy conditions (2.1) and (2.2), which implies that the preprocessing algorithm will fail to reduce the jobset significantly. It is worth noting that JP provided a mean gap of 0.49% for these hard instances.

insert Table 6 here

Impact of the different components

Since it is well known that the performance of B&B algorithms strongly relies on the quality of the implemented bounds, one may legitimately wonder whether an implementation of improved lower and upper bounds suffices to make CAR competitive with JP and/or SM. In order to investigate this issue, we embedded within CAR the lower and/or upper bounds developed in this paper, and we compared the performance of the resulting algorithm with JP and SM. Tables 7 and 8 depict the results of the experiments. In these tables, each entry represents the ratio of the measure obtained with the "improved" CAR with respect to JP or SM. We observe that although the new lower and/or upper bounds significantly improve the performance of CAR, the resulting improved variants of CAR are still lagged far behind JP and SM. Indeed, embedding the lower bound $JPPB$ together with the improved upper bound yields an algorithm which requires, on average, 75% more CPU time and explores 285% more nodes than JP does. Also, the implementation of $SMGB$ together with the upper bound yields an algorithm which requires 32% more CPU time and explores 64% more nodes than SM does.

insert Table 7 here

insert Table 8 here

Pushing the investigation a step further, we have evaluated the pertinence of each component (lower/upper bounds, preprocessing, FAP, Forward-Backward strategy). For that purpose, we have implemented five variants of our fastest algorithm (namely JP). These variants are the following:

- $JP\backslash LB$: The lower bound $JPPB$ has been replaced by the lower bound which has been implemented in CAR. This latter bound is equal to $\max(LB_0, SLB_2^h)$, where SLB_2^h is an approximate value of SLB_2
- $JP\backslash UB$: Only one upper bound is computed for each node. This bound corresponds to the makespan of Jackson's schedule constructed for the entire jobset J
- $JP\backslash PREP$: The preprocessing algorithm is not implemented
- $JP\backslash FAP$: The FAP is not implemented
- $JP\backslash FB$: There is no Forward-Backward implementation

We compared each of these variants to JP. The results are displayed in Table 9. We adopted the following notation:

$Time_{inc}$: the percentage increase of the mean CPU time
$Mean\ NN_{inc}$: the percentage increase of the mean number of nodes
US_{inc} : the percentage increase of the number of unsolved instances

Table 9 shows the worth of implementing each of the proposed components since JP consistently outperforms all the five variants. Indeed, we observe that $JP\backslash UB$ and $JP\backslash FB$ yield the two most important increases of the mean CPU time (71.17% and 66.37%, respectively). It is worth to note that a very similar behavior is observed for the percentage of unsolved instances. Moreover, $JP\backslash LB$ and $JP\backslash FAP$ exhibit a dramatic increase in the mean number of explored nodes (268.50% and 105.57%, respectively). Furthermore, skipping the preprocessing phase increases the mean CPU time by 15.40% and the mean number of explored nodes by 20.69%.

insert Table 9 here

7 Conclusion

We presented two variants of a time windows-based B&B algorithm for $P/r_j, q_j/C_{\max}$. The importance of this problem is twofold. Firstly, it generalizes several well-known identical parallel machine problems. Secondly, it arises as a strong relaxation of the Multiprocessor Job Shop problem. A new preprocessing algorithm was proposed in order to fix some jobs. This algorithm makes it possible not only to accelerate the convergence, but also to tighten the lower bounds. Several lower bounds were analyzed, and a new tight lower bound based on the parallel machine problem with availability constraints was introduced. Moreover, several rules for adjusting heads and tails, as well as a selective algorithm were developed in order to enhance the performance of the B&B algorithm. Finally, a Forward-Backward strategy was implemented and proved to be very useful. Extensive computational experiments, carried out on a set of 720 randomly generated instances prove that the proposed algorithms consistently outperform the only exact algorithm published so far [6]. Instances with up to 300 jobs and 4 machines are solved in a moderate computing time.

Future research effort needs to be focused on the class of problems which has been characterized to be the most intractable. This class refers to instances with short heads and tails and large number of machines. For that purpose, it will be worth developing new tighter bounds. A second issue worthy of future investigation, is to extend the present work to the case of uniform and unrelated machines.

Acknowledgments We thank two anonymous referees as well as the associate editor for their valuable comments which significantly improved the readability of the paper.

References

[1] **Lawler EL, Lenstra JK, Rinnooy Kan AHG and Shmoys D**. *Sequencing and Scheduling : Algorithms and Complexity.* Handbooks in Operations Research and Management Science 1993; 4, 445-522.

[2] **Lageweg BJ, Lenstra JK and Rinnooy Kan AHG.** *Minimizing Maximum Lateness on One Machine: Computational Experience and Some Applications.* Statistica Neerlandica 1976; 30, 25-41.

[3] **Garey MR and Johnson DS**. *Strong NP-completeness results : Motivation, examples and implications.* Journal of the Association of Computer Machinery 1978; 25, 499-508.

[4] **Horn WA.** *Some simple scheduling algorithms.* Naval Research Logistics Quarterly 1974; 21, 177-185.

[5] **Simons B and Warmuth M**. *A fast algorithm for multiprocessor scheduling of unit-length jobs.* SIAM Journal on Computing 1989; 18, 690-710.

[6] **Carlier J.** *Scheduling jobs with release dates and tails on identical machines to minimize the makespan.* European Journal of Operational Research 1987; 29, 298-306.

[7] **Gusfield D.** *Bounds for naive multiple machine scheduling with release times and deadlines.* Journal of Algorithms 1984; 5, 1-6.

[8] **Bratley P, Florian M and Robillard P.** *Scheduling with earliest start and due date constraints on multiple machines.* Naval Research Logistics Quarterly 1975; 22, 165-173.

[9] **Lancia G.** *Scheduling jobs with release dates and tails on two unrelated parallel machines to minimize the makespan.* European Journal of Operational Research 2000; 120, 277-288.

[10] **Dessouky MM.** *Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize the maximum lateness.* Computers and Industrial Engineering 1998; 34, 793-806.

[11] **Carlier J.** *The one machine sequencing problem.* European Journal of Operational Research 1982; 11, 42-47.

[12] **Nuijten WPM and Aarts EHL**. *A computational study of constraint satisfaction for multiple capacitated job shop scheduling.* European Journal of Operational Research 1996; 90, 269-284.

[13] **Sule DR and Vijayasundaram K.** *A heuristic procedure for makespan minimization in job shops with multiple identical processors.* Computers and Industrial Engineering 1998; 35, 399-402.

[14] **Adams J, Balas E and Zawack D.** *The shifting bottleneck procedure for job shop scheduling.* Management Science 1988; 34, 391-401.

[15] **Hoffman K and Padberg M.** *Improving LP-representations of zero-one linear programs for branch-and-cut.* ORSA Journal on Computing 1991; 3, 121-135.

[16] **Savelsbergh MWP.** *Preprocessing and probing techniques for mixed integer programming problems.* ORSA Journal on Computing 1994; 6, 445-454.

[17] **Vandevelde A.** *Minimizing the makespan in a multiprocessor flow shop.* Master's Thesis, Eindhoven University of Technology, The Netherlands, 1994.

[18] **Carlier J and Pinson E.** *Jackson's Pseudo Preemptive Schedule for the $Pm/r_j, q_j/C_{\max}$ scheduling problem.* Annals of Operations Research 1998; 83, 41-58.

[19] **Webster ST.** *A general lower bound for the makespan problem.* European Journal of Operational Research 1996; 89, 516-524.

[20] **Schmidt G.** *Scheduling with limited machine availability.* European Journal of Operational Research 2000; 121, 1-15.

j	1	2	3	4	5	6	7	8	9
r_j	1	4	2	7	8	12	14	2	0
p_j	2	3	7	3	5	6	4	5	2
q_j	14	14	15	14	11	2	7	2	1

Table 1. Data of the 9 job - 2 machine instance of example 1

j	1	2	3	4	5	6	7
r_j	1	1	1	2	2	2	8
p_j	9	8	4	8	7	1	9
q_j	8	8	9	8	8	10	1

Table 2. Data of the 7 job - 3 machine instance of example 2

j	1	2	3	4	5	6
r_j	10	3	1	6	2	8
p_j	1	9	7	2	9	7
q_j	7	6	5	10	8	4

Table 3. Data of the 6 job - 3 machine instance of example 3

	Time	*Mean NN*	*Max NN*	*US*	*Mean Gap*	*Max Gap*
JP	18.56	750.78	50817	5.83	0.58	1.51
SM	27.28	334.96	16721	7.50	0.57	1.52
CAR	73.91	18974.06	1053799	24.44	0.69	4.93

Table 4. Performance of the algorithms

		Time	*Mean NN*	*Max NN*	*US*	*Mean Gap*	*Max Gap*
JP	$n = 50$	28.08	2972.29	50817	9.16	1.15	1.51
	$n = 100$	22.63	796.93	13248	7.50	0.63	0.78
	$n = 150$	7.66	164.18	6828	2.50	0.39	0.48
	$n = 200$	13.59	211.16	5588	4.16	0.35	0.38
	$n = 250$	16.86	190.95	3772	5.00	0.27	0.30
	$n = 300$	22.54	169.16	3022	6.66	0.21	0.24
SM	$n = 50$	32.67	1537.04	16721	10.83	1.10	1.51
	$n = 100$	30.81	329.14	3811	10.00	0.67	1.52
	$n = 150$	8.80	37.69	1301	2.50	0.39	0.48
	$n = 200$	21.20	43.31	640	4.16	0.35	0.38
	$n = 250$	30.54	34.77	348	6.66	0.30	0.56
	$n = 300$	39.65	27.80	218	10.83	0.22	0.24
CAR	$n = 50$	110.06	101445.59	1053799	36.66	1.48	4.93
	$n = 100$	80.10	9114.04	179455	26.66	0.71	2.01
	$n = 150$	57.64	1639.52	39417	19.16	0.47	1.10
	$n = 200$	75.35	1157.44	16723	25.00	0.33	1.06
	$n = 250$	55.80	308.50	2833	18.33	0.27	0.55
	$n = 300$	64.54	179.26	1971	20.83	0.25	0.95

Table 5. Performance of the algorithms for each problem size

		Time	*Mean NN*	*Max NN*	*US*	*Mean Gap*	*Max Gap*
$m = 2$	$K = 1$	1.31	16.41	254	0.00	-	-
	$K = 3$	16.03	1571.76	50817	5.00	0.85	1.49
	$K = 5$	0.05	2.95	21	0.00	-	-
	$K = 7$	0.02	1.51	18	0.00	-	-
$m = 3$	$K = 1$	34.79	984.90	29956	10.00	0.44	1.16
	$K = 3$	35.50	2436.53	33180	11.66	0.73	1.05
	$K = 5$	0.05	3.15	39	0.00	-	-
	$K = 7$	0.01	1.16	9	0.00	-	-
$m = 4$	$K = 1$	104.87	2558.15	30249	33.33	0.49	1.51
	$K = 3$	30.10	1429.78	34761	10.00	0.73	1.42
	$K = 5$	0.02	1.78	19	0.00	-	-
	$K = 7$	0.01	1.26	16	0.00	-	-

Table 6. Sensitivity of JP to the variation of m and K

	$Time_{ratio}$	$Mean\ NN_{ratio}$	US_{ratio}
CAR	3.98	25.27	4.19
$CAR + JPPB$	2.96	9.55	3.12
$CAR + UB$	2.48	9.02	2.61
$CAR + JPPB + UB$	1.75	3.85	1.83

Table 7. Relative performance of the bounds with respect to JP

	$Time_{ratio}$	$Mean\ NN_{ratio}$	US_{ratio}
CAR	2.70	56.64	3.25
$CAR + SMGB$	2.28	6.41	2.55
$CAR + UB$	1.68	20.22	2.03
$CAR + SMGB + UB$	1.32	1.64	1.55

Table 8. Relative performance of the bounds with respect to SM

	$Time_{inc}$	$Mean\ NN_{inc}$	US_{inc}
$JP \setminus LB$	36.20	268.50	38.07
$JP \setminus UB$	71.17	71.04	78.55
$JP \setminus PREP$	15.40	20.69	16.63
$JP \setminus FAP$	3.98	105.57	4.80
$JP \setminus FB$	66.37	84.73	73.75

Table 9. Impact of different components on JP

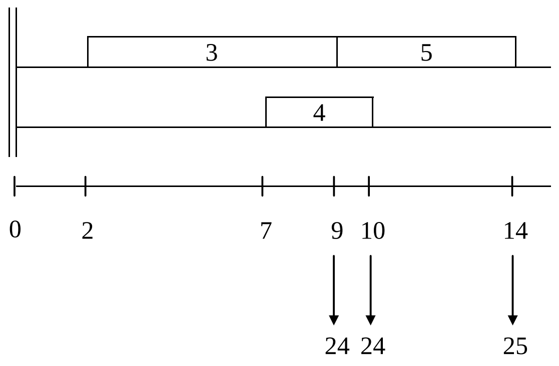
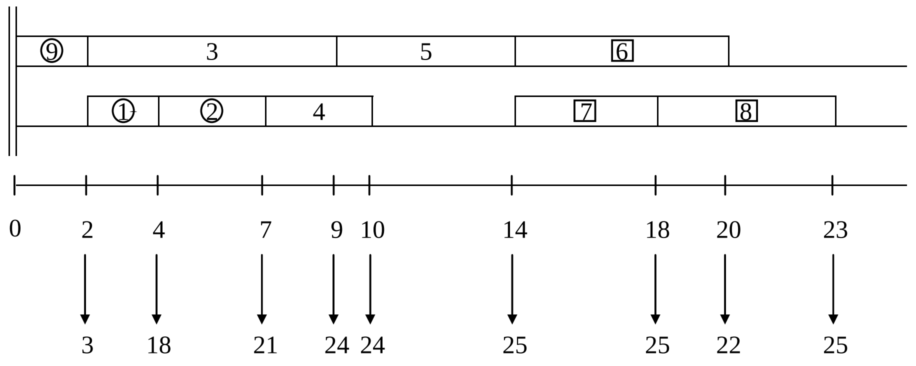
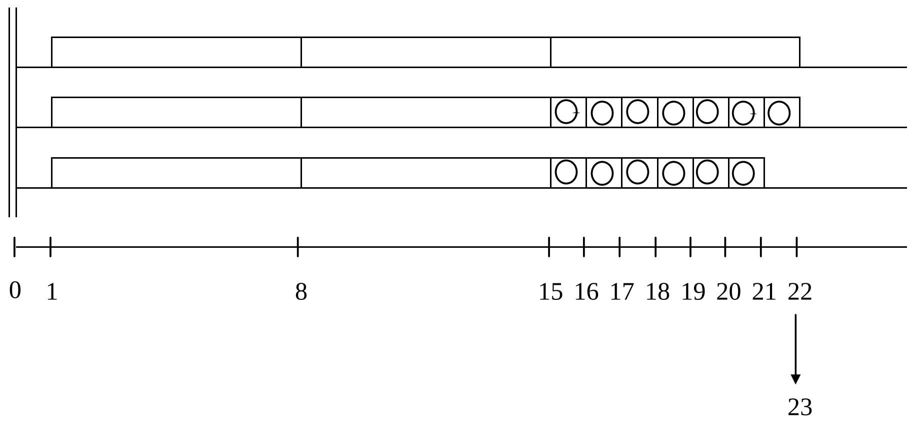


Figure 1. Schedule of $\bar{J}$

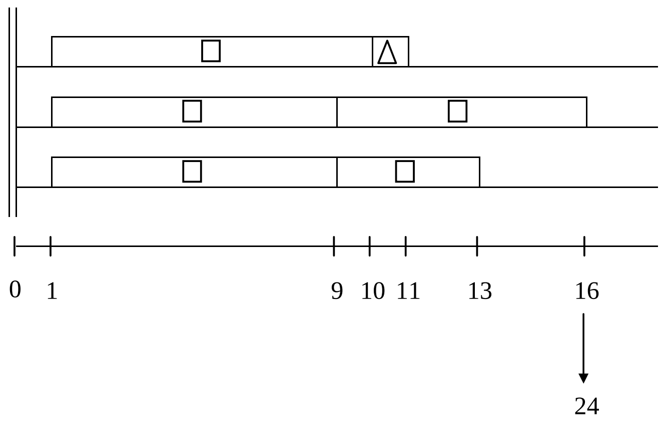


○ Jobs of J_R □ Jobs of J_Q

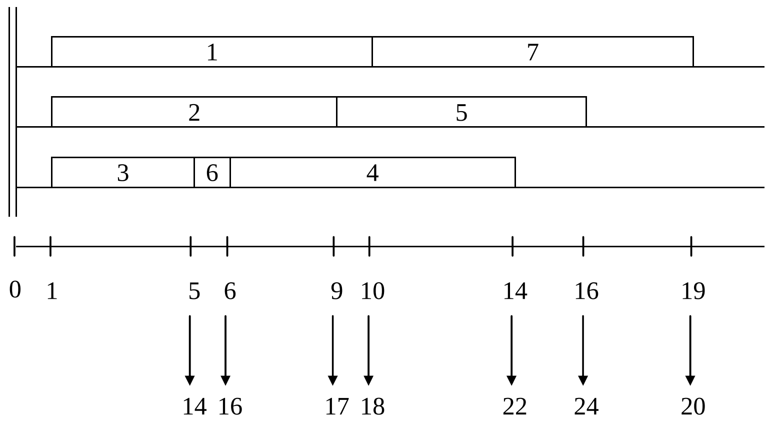
Figure 2. Schedule of J obtained after application of *GSCA*



$3a.\, MGB(J) = 23$



$3b.\, SMGB(J) = 24$



$3c.\, C^*_{\max}(J) = 24$

□ *Jobs of J_1* △ *Jobs of J_2* ○ *Jobs of J_3*

Figure 3. $SLB_2(J) < MGB(J) < SMGB(J) = C^*_{\max}(J)$