Multiple-Machine Lower Bounds for Shop Scheduling Problems

Francis Sourd / *LIP6 - Université P. et M. Curie, 4 place Jussieu, F-75252 Paris Cedex 05, France / EMail: Francis.Sourd@lip6.fr*

Wim Nuijten / *ILOG S.A., 9 Rue de Verdun, BP 85, F-94253 Gentilly Cedex, France / EMail : nuijten@ilog.fr*

Subject classifications: 591 Production-Scheduling, Open shop, Job shop, Deterministic
Other key words: 630 Programming, Integer, Algorithms, Branch and bound

In order to compute lower bounds for shop scheduling problems, a lot of attention has been paid to adjustment techniques based on one-machine relaxations. We present such a new technique but, following the observation that machines are connected to each other through precedence constraints, we also study techniques that are based on the combination of precedence constraints and disjunctive constraints between operations that are processed on different machines. A computational study of the effectiveness of these new techniques is performed on job shop and flow shop instances.

The Job-Shop Scheduling Problem (JSSP) is certainly one of the most famous scheduling problems. The recent surveys about this problem [Blazewicz, Domschke & Pesch, 1996; Jain & Meeran, 1999] show that much research effort was devoted both to find optimal and approximate solutions. In the JSSP, each resource can process only one operation at a time, which can explain the interest of researchers for the problem.

1. Such resources are widely used for modeling industrial production processes.
2. In real life problems, these resources are often critical and therefore often demand extra attention.

As such we can remark that progress made on solving the JSSP can be useful for many more general scheduling problems involving unit-capacity resources (machines).

In this paper, we are interested in finding optimal solutions for the JSSP and the Flow Shop Scheduling Problem (FSSP). The algorithms to find such schedules are all based on the Branch-and-Bound approach. Therefore, their efficiency is directly depending on the quality of the lower bounds we are able to compute for these problems. In practice, the computation of these lower bounds is always based on the *earliest start times* (which are lower bounds on the start time of each operation) and the *latest completion times* (which are upper bounds on the end time of each operation) so that any improvement of an earliest start time or a latest completion time will generally improve the lower bound for the objective function. Over the years, several *time-bound adjustment techniques* have been developed to improve the earliest start times and the latest completion times [Carlier & Pinson, 1990; Carlier & Pinson, 1994; Baptiste & Le Pape, 1996; Nuijten & Le Pape, 1998]. We note that all these techniques are based on the one-machine problem, that is, the new earliest start time (or latest completion time) of an operation o is derived from properties of the operations that are processed on the same machine as o.

This paper presents two new classes of adjustment techniques. The first one also relies on the one-machine problem. It is based on properties of a *non-precedence* relation between operations. The second one comes from the observation that machines are "connected" to each other through precedence constraints. Thanks to the properties derived from the combination of precedence constraints and disjunctive constraints, we are able to adjust the earliest start time of an operation o by considering operations that are executed on another machine than the machine on which o is processed.

Definitions of the JSSP and FSSP are introduced in Section 1. Section 2 is devoted to the *non-precedence* conditions and to their applications to compute lower bounds for the JSSP. Section 3 presents an adjustement technique based on the combination of precedence and disjunctive constraints. Section 4 defines the distance between two operations, a notion that generalizes the ideas presented in Section 3. Finally, Section 5 discusses the computational efficiency of these new techniques to optimally solve instances of the JSSP and FSSP.

1 The Job-Shop and Flow-Shop Scheduling Problems

1.1 Definitions

Definition 1.1. An instance of the *Job Shop Scheduling Problem* (JSSP) consists of a set $\mathcal{O}$ of operations, a set $\mathcal{J}$ of jobs, and a set $\mathcal{M}$ of machines. For each operation $o \in \mathcal{O}$, $J(o)$ denotes the job to which o belongs, $M(o)$ is the machine by which o must be processed, and $pt(o) \in \mathbb{N}^+$ is the processing time of o. $\mathcal{O}_\mu$ denotes the set of operations that are to be scheduled on $\mu \in \mathcal{M}$: $\mathcal{O}_\mu = \{o \in \mathcal{O}, M(o) = \mu\}$. A binary relation $\rightarrow$ is given that decomposes $\mathcal{O}$ into chains, such that every chain corresponds to a job. A (feasible) *schedule* s is a function $s : \mathcal{O} \rightarrow \mathbb{N}$ that gives the start time of each operation, such that for all $o, o' \in \mathcal{O}$,

- $s(o) \geq 0$,
- $s(o) + pt(o) \leq s(o')$, if $o \rightarrow o'$, and
- $s(o) + pt(o) \leq s(o')$ or $s(o') + pt(o') \leq s(o)$, if $M(o) = M(o')$ and $o \neq o'$.

The *makespan* $C_{max}(s)$ of a schedule s is defined as the maximum completion time, i.e., $C_{max}(s) = \max_{o \in \mathcal{O}} (s(o) + pt(o))$. The problem is to find a schedule s with a minimum makespan. □

We refer to constraints defined on operations that are related by the binary relation $\rightarrow$ as *precedence constraints*. The constraints expressing that no pair of operations on the same machine can overlap in time are referred to as *resource constraints* or *disjunctive constraints*. We remark that the decision variant of the JSSP has been proven to be NP-complete [Garey, Johnson & Sethi, 1976].

Definition 1.2. An instance of the *Flow Shop Scheduling Problem* (FSSP) is an instance of the JSSP in which each job has $|\mathcal{M}|$ operations and the i^{th} operation of each job is executed on the machine μ_i where $\{\mu_1, \mu_2, \cdots, \mu_{|\mathcal{M}|}\} = \mathcal{M}$. □

Definition 1.3. Any instance of the JSSP or the FSSP can be represented by means of a *disjunctive graph* [Roy & Sussmann, 1964] $G(\mathcal{O}, A, E)$ where $A = \{(o, o'), o \rightarrow o'\}$ and $E = \{\{o, o'\}, o \neq o' \text{ and } M(o) = M(o')\}$. The arcs in A represent the given precedence relations and the edges in E represent the disjunctive constraints. Each vertex $o \in \mathcal{O}$ has a weight, equal to its processing time $pt(o)$. □

For each $E' \subseteq E$, an *orientation* on E' is a function $\omega : E' \rightarrow \mathcal{O} \times \mathcal{O}$ such that $\omega(\{o, o'\}) \in \{(o, o'), (o', o)\}$ for each $\{o, o'\} \in E'$. We denote $\{\omega(e),\ e \in E'\}$ by $\omega(E')$. The orientation $\omega(E')$ is said to be *feasible* if the digraph $G(\mathcal{O}, A \cup \omega(E'))$ is acyclic, which means that the binary relation $\rightarrow$ may be extended by the pairs of $\omega(E')$. The extended precedence relation will be denoted by $\stackrel{\omega}{\rightarrow}$. A *complete orientation* is an orientation on E while a *partial orientation* is on $E' \neq E$. Note that each feasible schedule determines a unique feasible complete orientation and conversely that each feasible complete orientation determines a unique earliest schedule.

We now introduce some definitions and notations for lower and upper bound functions applying to any feasible schedule s.

Definition 1.4. Let $E' \subset E$, $\omega : E' \rightarrow \mathcal{O} \times \mathcal{O}$ be a partial orientation of the disjunctive graph and $T \in \mathbb{N}$. Let $S(\omega, T)$ be the set of the feasible schedules $s : \mathcal{O} \rightarrow \mathbb{N}$ such that $C_{max}(s) \leq T$ and for all $(o, o') \in \omega(E')$, $s(o) + pt(o) \leq s(o')$.
An *earliest start time* function, denoted by $est_{\omega,T}$, is a function $\mathcal{O} \rightarrow \mathbb{N}$ such that $\forall_{o \in \mathcal{O}} \forall_{s \in S(\omega,T)}\ est_{\omega,T}(o) \leq s(o)$. Symmetrically, a *latest completion time* function, denoted by $lct_{\omega,T}$, is a function $\mathcal{O} \rightarrow \mathbb{N}$ such that $\forall_{o \in \mathcal{O}} \forall_{s \in S(\omega,T)}\ lct_{\omega,T}(o) \geq s(o) + pt(o)$. □

In what follows, we will assume that ω and T are given, that is we have a partial schedule and we are trying to complete the partial orientation while keeping the makespan less than T. We will omit to mention ω and T in the notations of the earliest start time and latest completion time. We also introduce the following notations:

- $est(S) = \min_{o \in S} est(o)$,
- $lct(S) = \max_{o \in S} lct(o)$,
- $pt(S) = \sum_{o \in S} pt(o)$.

1.2 Branch-and-bound scheme to search for an optimal schedule

Let us assume that an earliest start time function est is also a feasible schedule. Clearly, est is also an optimal schedule for minimizing the makespan. Unfortunately, in general est does not satisfy all the constraints of the Definition 1.1. In order to find an optimal schedule, we can improve est by increasing its values for some $o \in \mathcal{O}$ until we get a feasible schedule. Techniques to do such updates of est are called *adjustment techniques*. If it can be proven there exists no feasible schedule in $S(\omega, T)$ with an operation $\bar{o}$ starting before $t \in]est(\bar{o}), lct(\bar{o}) - pt(\bar{o})]$, the function est'

$$est'(o) = \begin{cases} est(o) & \text{if } o \neq \bar{o} \\ t & \text{if } o = \bar{o} \end{cases}$$

is clearly an earliest start time function. An *adjustment operator* is then based on:

1. a property (P) that shows that given the earliest start times and latest completion times of a subset $S \subset \mathcal{O}$ an operation o cannot start before a date $t(S)$ that is function of S.
2. an algorithm that finds all the operations that can be updated according to property (P) and that computes, for each such operation, the subset S such that $t(S)$ is maximal.

We refer to [Nuijten & Le Pape, 1998] for a list of adjustment operators for the JSSP. We observe that earliest start times and latest completion times play a symmetric role: from any operator that adjusts est, a dual operator updating lct can be derived. As such, in this paper we will only present adjustment techniques for the earliest start times.

Once all update operators we have are applied, the resulting earliest start time function is not necessarily a feasible schedule. The method to find a feasible schedule in $S(\omega, T)$ is then to divide this set into subsets $S(\omega_1, T), S(\omega_2, T), \cdots, S(\omega_k, T)$ such that any ω_i extends ω on a domain E_i (with $E' \subset E_i \subseteq E$) and $S(\omega, T) = \bigcup_{i=1}^{k} S(\omega_i, T)$. After that, for each $i \in [1, k]$, an optimal solution is searched in $S(\omega_i, T)$ by applying the adjustment operators and eventually subdividing $S(\omega_i, T)$. This process is iterated until est becomes a feasible schedule or until we can prove that the current set of operations $S(\omega, T)$ is empty (for example by finding an operation o such that $est(o) + pt(o) > lct(o)$). In the latter case, the search must continue in another subset that has not been explored yet.

In terms of computation time, we have to cope with the following dilemma. Clearly, an adjustment operator helps to prove that a set $S(\omega, T)$ is empty — and subsequently avoids to divide this set into sub-problems — but it also requires a certain amount of computional time. Therefore, an operator is worth being introduced if the time saved by not exploring some subsets is longer than the computational time to prove it.

2 Non-precedence conditions

This section presents new adjustment techniques for operations that are processed on the same machine. We first present a situation in which none of the already known operators can update the earliest start time function but where some adjustments can clearly be made. We then formulate two lemmas yielding two new adjustment operators which are able to cope with such a situation.

Figure 1 shows four operations that are to be scheduled on the same machine. Each line of the figure represents an operation. For example, the second line refers to o_2 and $est(o_2) = 0$, $lct(o_2) = 13$ and $pt(o_2) = 4$. If o_4 is scheduled before o_1, one can easily see that both o_2 and o_3 must be scheduled after o_1, which is impossible. So o_4 must be scheduled after o_1 and $est(o_4)$ can be set to 7. Furthermore, we can observe that for any $S \subseteq \{o_1, o_2, o_3\}$, $est(o_4) + pt(o_4) \leq lct(S) - pt(S)$, which helps to verify that none of the adjustment operators surveyed in [Nuijten & Le Pape, 1998] are able to find an update

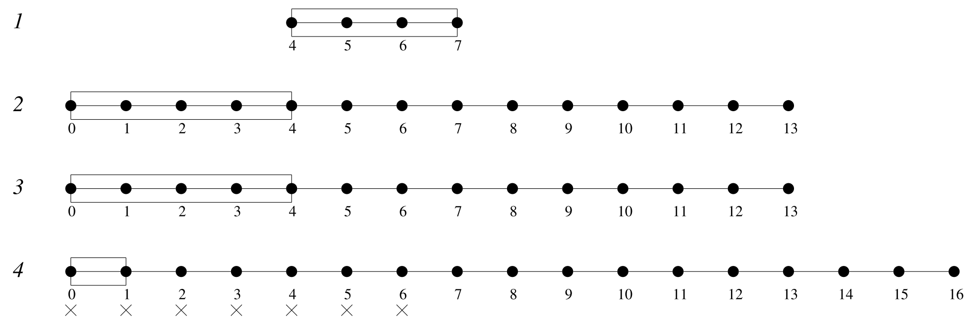

Figure 1: Four operations on the same machine

in such a situation. For the general form of this observation, we introduce the following definition.

Definition 2.1. Let $\mu \in \mathcal{M}$ be a machine, and let $S, S' \subset \mathcal{O}_\mu$ be two disjoint sets of operations. We will say that S is *not completely before* S' — denoted by $S \not\twoheadrightarrow S'$ — if there is no feasible schedule in which all operations of S are scheduled before all operations of S' (in other words, $S \not\twoheadrightarrow S'$ if and only if for all schedules s, $\exists_{(o,o') \in S \times S'}$ $s(o') < s(o)$). $\square$

If $S \not\twoheadrightarrow S'$, immediate deductions on the earliest start time and latest completion time of the operations in S or S' cannot be made. With no additional information, it is even impossible to say whether the first operation scheduled is in S or in S'. If S has a unique element o, at least one operation in S' must be scheduled before o so that o cannot start before $\min_{o' \in S'} est(o') + pt(o')$.

A well-known sufficient condition for $S \not\twoheadrightarrow S'$ is $est(S) + pt(S) + pt(S') > lct(S')$. In particular, if $est(o)+pt(o)+pt(S') > lct(S')$, $est(o)$ can be set to $\min_{o' \in S'} (est(o') + pt(o'))$. The so called *not-first* operator can find in quadratic time on each machine all such adjustments [Nuijten, 1994; Baptiste & Le Pape, 1996]. In what follows, we present new sufficient conditions to conclude that $o \not\twoheadrightarrow S'$ in order to have better adjustments on $est(o)$. These conditions are based on the conjunction of several *not completely before* relations.

Lemma 2.1. *Let $o \in \mathcal{O}_\mu$ be an operation and $S, S' \subset \mathcal{O}_\mu$ two disjoint sets of operations such that $o \notin S \cup S'$. If*

- $o \not\twoheadrightarrow S'$
- $\forall_{o' \in S'}\ S \cup \{o'\} \not\twoheadrightarrow o$

then $S \not\twoheadrightarrow o$.

Proof. Let us suppose that $o \not\twoheadrightarrow S'$ and that for each $o' \in S'$, $S \cup \{o'\} \not\twoheadrightarrow o$. Let s be a feasible schedule. Because $o \not\twoheadrightarrow S'$, there is an operation $\bar{o} \in S'$ such that $s(\bar{o}) < s(o)$. Because $S \cup \{o'\} \not\twoheadrightarrow o$, there is an operation of S processed after o in s. So $S \not\twoheadrightarrow o$. $\square$

In the situation given by Figure 1, this lemma yields $o_4 \not\twoheadrightarrow o_1$. To see this define o to be o_1, S' to be $\{o_2, o_3\}$, and S to be $\{o_4\}$ in Lemma 2.1. So o_4 must be scheduled after o_1 and its earliest start time can be set to 7.

The second lemma is more general because it is based on a *not completely before* relation between two sets of operations but it requires a supplementary condition.

Lemma 2.2. *Let $o \in \mathcal{O}_\mu$ be an operation and $S, S' \subset \mathcal{O}_\mu$ two disjoint sets of operations such that $o \notin S \cup S'$. If*

i) $S \not\twoheadrightarrow S'$

ii) $\forall_{o' \in S'}\ pt(o') > lct(S) - est(S) - pt(S)$

iii) $\forall_{o' \in S'}\ \{o, o'\} \not\twoheadrightarrow S$

then $o \not\twoheadrightarrow S$

Proof. Let us suppose that o is scheduled before S. For all $o' \in S'$, o' cannot be scheduled before S because of iii) and o' cannot be scheduled in between S because of ii), so o' is scheduled after S. This means there is a schedule where all operations of S are scheduled before all operations of S', which is a contradiction with i). $\square$

In the situation given by Figure 1, this lemma also yields $o_4 \not\twoheadrightarrow o_1$, just like Lemma 2.1. To see this define o to be o_4, S' to be $\{o_2, o_3\}$, and S to be $\{o_1\}$ in Lemma 2.2.

Symmetrical lemmas could be written by considering a *not completely after* relation.

To decide whether $S \not\twoheadrightarrow S'$ is of course NP-complete but Lemmas 2.1 and 2.2 are still valid if we replace all the input conditions $S \not\twoheadrightarrow S'$ by the sufficient conditions $est(S) + pt(S) + pt(S') > lct(S')$. We can then write two algorithms (one for each lemma) that searches for all the o, S and S' which satisfy these sufficient conditions. The operator based on Lemma 2.1 updates $lct(o)$ with a time complexity $O(n^3)$ and the operator based on Lemma 2.2 updates $est(o)$ in $O(n^4)$ time where $n = |\mathcal{O}_\mu|$. Both operators have of course their symmetric version. We only implemented basic versions of these operators, permitting us to do some computational tests. These computational tests — see Section 5 — showed that there were very few adjustments found. Based on this we did not search for a better implementation. This is not to say that these adjustment operators can not be useful. The results did strenghten our belief we needed to look beyond one-machine problems.

3 Multiple-machine based adjustment techniques

3.1 Formulation

Following the observation that machines are "connected to each other" through precedence constraints, we now study adjustment techniques that are based on the combination of precedence and disjunctive constraints. The interest of such an approach is illustrated by the following example.

Figure 2 shows two operations o_2 and o_3 that are to be scheduled on the same machine. The operation o_1 can be scheduled on any machine but it is supposed to be preceded by both o_2 and o_3. The earliest completion time of the set $\{o_2, o_3\}$ is 7, therefore there is no schedule such that $s(o_1) < 7$. We must notice that such an update cannot be found by any algorithm that would only consider the operations scheduled on $M(o)$.

Given a partial schedule ω as defined in §1.1, we recall that the digraph $(\mathcal{O}, A \cup \omega(E'))$ representing the current set of precedence relations is naturally acyclic. We define the

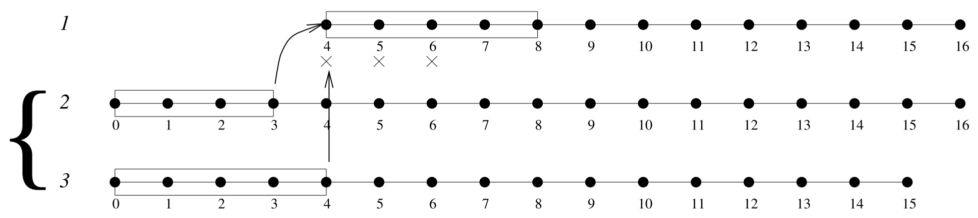

Figure 2: o_2 and o_3 on the same machine precede o (eventually on another machine)

length of a path as the sum of the processing times of the operations traversed by the path — thus excluding the processing times of its two ends. For instance, the length of the path $o_1 \overset{\omega}{\to} o_2 \overset{\omega}{\to} \cdots \overset{\omega}{\to} o_{j-1} \overset{\omega}{\to} o_j$ is $\sum_{1<k<j} pt(o_k)$. If there is a path from o to o', there also exists a longest path between these two vertices. The length $l^0(o, o')$ of this longest path is obviously a lower bound on the time between the end of o and the start of o'. For convenience, $l^0(o, o')$ will be set to $-\infty$ if there is no path from o to o' and we will assume that for all $x \in \mathbb{R}$, $x + (-\infty) = -\infty$. This definition can be extended to any set $S \subset \mathcal{O}$ as follows: $l^0(S, o) = \min_{o' \in S} l^0(o', o)$.

Lemma 3.1. *For any $\mu \in \mathcal{M}$, $S \subseteq \mathcal{O}_\mu$ and $o \in \mathcal{O} \backslash S$, we have:*

$$s(o) \geq est(S) + pt(S) + l^0(S, o)$$

Proof. The proof is straightforward: since all operations in S are to be scheduled on the same machine, the last operation of S cannot end before $est(S) + pt(S)$. Since $l^0(S, o)$ is a lower bound on the time between the end of the last scheduled operation in S and the start of o, we have $s(o) \geq est(S) + pt(S) + l^0(S, o)$. □

Note 3.1. With the convention $l^0(o, o') = -\infty$, if o is not a predecessor of o', the inequality $s(o) \geq est(S) + pt(S) + l^0(S, o)$ is valid for any $S \subseteq \mathcal{O}_\mu$ and any $o \notin S$. □

We remark that independently of our work, Peridy [1996] includes a few of the ideas presented until now. Unfortunately he did not succeed in obtaining a succesful implementation of his ideas.

Lemma 3.1 ignores the fact that operations on machine μ have deadlines, that is they must be completed before their latest completion time. Figure 3 shows that these deadlines can be important in some situations. If we only consider the two predecessors o_3 and o_4 of operation o_5, then $est(o_5)$ can be set to $3 + 2 + 2 = 7$. However, since operations o_1 and o_2 must be processed on the same machine, it is easy to see that in any feasible schedule, either o_3 or o_4 is scheduled after operations o_1 and o_2. As a consequence $est(o_5)$ can be set to 13. More generally, for any operation o and for any machine μ, operation o cannot start before the makespan of the problem (P_0) in which all the operations $o' \in \mathcal{O}_\mu$ must be scheduled on a single machine with a release date $est(o')$, a processing time $pt(o')$, a deadline $lct(o')$ and a tail $l^0(o', o)$. Following the standard three-field notation [Graham, Lawler, Lenstra & Rinnooy Kan, 1979], (P_0) is

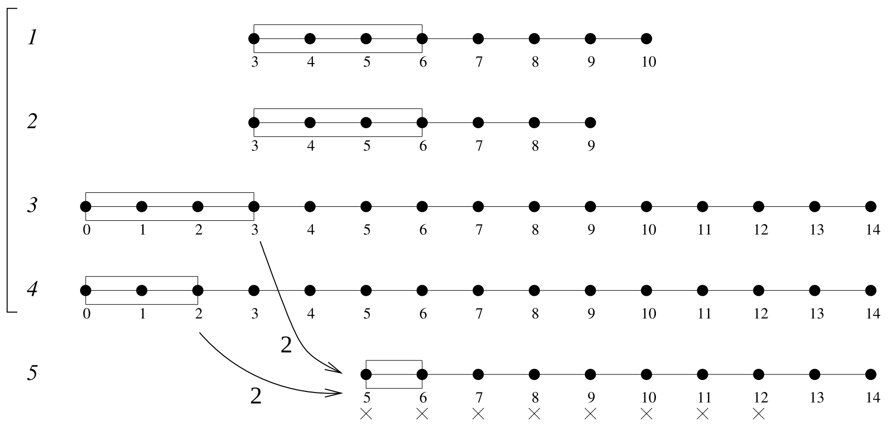

Figure 3: Operations $o_1, ..., o_4$ are to be scheduled on the same machine. At least 2 time units are required between the end of operations o_3 and o_4 and the start of o_5.

Lower bound	Related sub-problem	Complexity
$E_0(o, \mu)$	$1\|r_j, d_j\| \max_j(C_j + q_j)$	NP-complete
$E_1(o, \mu)$	$1\|prmp, r_j, d_j\| \max_j(C_j + q_j)$	$O(n \log n)$
$E_2(o, \mu)$	$1\|prmp, r_j\| \max_j(C_j + q_j)$	$O(n \log n)$

Table I: Three lower bounds for the start time of o.

denoted by $1|r_j, d_j| \max_j(C_j + q_j)$. $E_0(o, \mu)$ will denote the optimal makespan of the instance of (P_0) defined for o and μ.

Problem (P_0) is NP-complete in the strong sense because it is more difficult than the one-machine problem with release dates and deadlines. However its preemptive relaxation (P_1) : $1|prmp, r_j, d_j| \max_j(C_j + q_j)$ is solved in $O(n \log n)$ time [Sourd & Nuijten, 1999]. This problem can also be solved in quadratic time as an instance of the problem $1|prmp, r_j| \max_j f_j(C_j)$ where f_j are non-decreasing functions [Baker, Lawler, Lenstra & Rinnooy Kan, 1983]. The bound derived from (P_1) will be denoted as $E_1(o, \mu)$. The bound based on Lemma 3.1, namely $\max_{S \subseteq \mathcal{O}_\mu} est(S) + pt(S) + l^0(S, o)$, will be denoted by $E_2(o, \mu)$. $E_2(o, \mu)$ can also be computed in $O(n \log n)$ time by an algorithm given in [Carlier, 1982]. In that paper, it is also shown that $E_2(o, \mu)$ is the optimal makespan of the problem with relaxed deadlines — (P_2) : $1|prmp, r_j| \max_j(C_j + q_j)$.

In summary, we have the following lemma:

Lemma 3.2. *For any $o \in \mathcal{O}$ and for any $\mu \in \mathcal{M}$*

$$s(o) \geq E_0(o, \mu) \geq E_1(o, \mu) \geq E_2(o, \mu)$$

□

3.2 The adjustment operator

The adjustment operator we propose is an algorithm that, given a machine μ, updates the earliest start times of all the operations of the problem to make them compatible with one of the three lower bounds of Lemma 3.2. Let us introduce the set $\mathcal{P}(o,\mu)$ of the predecessors of o that are scheduled on μ: $\mathcal{P}(o,\mu) = \{o' \in \mathcal{O}_\mu \,|\, l^0(o',o) \geq 0\}$. We will also need the following characteristic function:

$$\mathbf{1}(o,\mu) = \begin{cases} \{o\} & \text{if } o \in \mathcal{O}_\mu \\ \emptyset & \text{otherwise} \end{cases}$$

The algorithm visits all the operations of the problem in a topological order. For each operation o, it updates the list of its predecessors that must be processed on machine μ and calculates their distance to o. With these informations the lower bound on $s(o)$ can then be computed. In the following description of the algorithm, *level* indicates the selected lower bound for $s(o)$ (cf. Lemma 3.2).

function `update_est`$(\mu, level)$
begin
  for all $o \in \mathcal{O}$ **do** $\mathcal{P}(o,\mu) \leftarrow \emptyset$
  for all $o, o' \in \mathcal{O}$ **do** $l^0(o,o') \leftarrow -\infty$
  for each $o \in \mathcal{O}$ (selected in a topological order) **do**
    $\mathcal{P}(o,\mu) \leftarrow \bigcup_{o' \stackrel{\omega}{\to} o} \mathcal{P}(o',\mu) \cup \mathbf{1}(o',\mu)$
    for each $o'' \in \mathcal{P}(o,\mu)$ **do**
      $l^0(o'',o) \leftarrow \max\left(0, \max_{o' \stackrel{\omega}{\to} o} \left(l^0(o'',o') + pt(o')\right)\right)$
  for each $o \in \mathcal{O}$ **do**
    $est(o) \leftarrow \max\left(est(o), E_{level}(o,\mu)\right)$
end

The values $\mathcal{P}(o,\mu)$ and $l^0(o'',o)$ are calculated by dynamic programming techniques. Indeed, when the main loop considers the operation o, all the values $\mathcal{P}(o',\mu)$ and $l^0(o,o'')$ for any direct predecessor o' of o are up-to-date. Clearly, the operations in $\bigcup_{o' \stackrel{\omega}{\to} o} \mathcal{P}(o',\mu) \cup \mathbf{1}(o',\mu)$ are the predecessors of o that must be processed on machine μ. Conversely, let o'' be a predecessor of o processed by machine μ. If o'' is a direct predecessor of o then $o'' \stackrel{\omega}{\to} o$ and $\mathbf{1}(o'',\mu) = \{o''\}$, otherwise o'' is a predecessor of at least one direct predecessor o' of o (*i.e.*, $o'' \in \mathcal{P}(o',\mu)$). So the equation $\mathcal{P}(o,\mu) = \bigcup_{o' \stackrel{\omega}{\to} o} \mathcal{P}(o',\mu) \cup \mathbf{1}(o',\mu)$ is valid. The proof of the validity of our equation to compute $l^0(o'',o)$ is similar.

Since $\mathcal{P}(o,\mu)$ is by definition the set $\{o' \in \mathcal{O}_\mu \,|\, l^0(o',o) \geq 0\}$, the implementation of the algorithm requires only one data structure, namely an array of integers of size $|\mathcal{O}_\mu| \times |\mathcal{O}|$ that stores all the l^0-values. For any $o \in \mathcal{O}$, $|\mathcal{P}(o,\mu)| \leq |\mathcal{O}_\mu|$. As usual we assume that each $l^0(o'',o)$-value can be accessed in constant time, so the algorithm computes the $l^0(o'',o)$-values for a given o and for all $o'' \in \mathcal{O}_\mu$ in a time $O(|\mathcal{O}_\mu| \times |\{o' \,|\, o' \stackrel{\omega}{\to} o\}|)$. Therefore, all the l^0-values are computed in $O(|\mathcal{O}_\mu| \times |A \cup \omega(E')|)$ time.

Since the complexity of (P_1) and (P_2) is $O(n \log n)$, the complexity of the second loop to update the earliest start times is $O(|\mathcal{O}| \times |\mathcal{O}_\mu| \log |\mathcal{O}_\mu|)$ when $level = 1$ or $level = 2$. When $level = 0$, the running time of this phase cannot of course be polynomially bounded. In our implementation, each $E_0(o,\mu)$-value is computed with a (simple) branch-and-bound algorithm based on [Carlier, 1982].

Whatever the value of k is, the complexity of the adjustment operator is quite high. We now introduce some dominance properties to improve its practical running time.

Lemma 3.3. *Let o be an operation in $\mathcal{O}$ and let $\mu \in \mathcal{M}$. If o has only one direct predecessor o' then for any $k \in \{0,1,2\}$, $E_k(o,\mu) = E_k(o',\mu) + pt(o')$.*

Proof. Let o'' be some operation in $\mathcal{O} - \{o, o'\}$. If o'' is not a predecessor of o, $l^0(o'', o) = l^0(o'', o') = -\infty$. Otherwise, o'' is not a direct predecessor of o and $l^0(o'', o) = l^0(o'', o') + pt(o')$. Therefore, for any $k \in \{0,1,2\}$, $E_k(o,\mu) = E_k(o',\mu) + pt(o')$. □

In practice, $E_{level}(o,\mu)$ is not computed for operations with only one predecessor because the same adjustment is found (in constant time) when the operator to update precedence constraints is called.

The following lemma considers the frequent case when the first operations on machine μ are already sequenced. It is assumed that for any pair $o, o' \in \mathcal{O}$ such that $o \stackrel{\omega}{\rightarrow} o'$ the est function verifies that $est(o) + pt(o) \leq est(o')$.

Lemma 3.4. *Let us assume that $\mathcal{O}_\mu = \{o_1, \cdots, o_i\} \cup S$, $o_1 \stackrel{\omega}{\rightarrow} \cdots \stackrel{\omega}{\rightarrow} o_i$ and $\forall_{o' \in S}\ o_i \stackrel{\omega}{\rightarrow} o'$. For any $k \in \{0,1,2\}$ and any $o \in \mathcal{O}$, $E_k(o,\mu)$ is less than the maximal value between $est(o)$ and the makespan of the instance of (P_k) restricted to the operations in S.*

Proof. Let us consider an optimal schedule $\mathcal{S}_S$ of the operations of S for problem (P_k). The operations $o_1, \cdots, o_i$ can be scheduled at their earliest start time to derive a feasible schedule of an instance of (P_k) with all the operations of $\mathcal{O}_\mu$. The makespan of this schedule is either the makespan of $\mathcal{S}_S$ or is $est(o_j) + pt(o_j) + l^0(o_j, o)$ for some $j \in [1, i]$. Since for any pair $o, o' \in \mathcal{O}$ such that $o \stackrel{\omega}{\rightarrow} o'$ it holds that $est(o) + pt(o) \leq est(o')$, we can deduce that $est(o_j) + pt(o_j) + l^0(o_j, o)$ is less than $est(o)$. □

Several tricks can speed up the algorithm. For example, for any $k \in \{0,1,2\}$, a simple heuristic can derive a schedule of the operations of (P_k). If this schedule is feasible for (P_k) and if its makespan is less than $est(o)$, we are sure that $E_k(o,\mu)$ cannot improve $est(o)$.

3.3 Adjustment operator calling conditions

We assume in this section that k is some fixed value in $\{0,1,2\}$ and $\mu \in \mathcal{M}$. In order to have $est(o) \geq E_k(o,\mu)$ for any operation $o \in \mathcal{O}$, the procedure `update_est`(μ, k) must be called each time $E_k(o,\mu)$ has been modified for some $o \in \mathcal{O}$ since the last call to the procedure. A necessary condition is that the instance of the scheduling problem whose makespan is $E_k(o,\mu)$ has been modified. More precisely, the conditions to call the adjustment operator are:

- if $k = 0$ or $k = 1$: there exists $o' \in \mathcal{O}_\mu$ such that $est(o')$ or $lct(o')$ has been modified or there exists $(o', o) \in \mathcal{O}_\mu \times \mathcal{O}$ such that $l^0(o', o)$ has been modified.
- if $k = 2$: there exists $o' \in \mathcal{O}_\mu$ such that $est(o')$ has been modified or there exists $(o', o) \in \mathcal{O}_\mu \times \mathcal{O}$ such that $l^0(o', o)$ has been modified.

In §3.2 we have shown that the complexity of our adjustment operator was quite high. Since the above condition to call the operator happens very often in a branch-and-bound scheme, the total time required to update the earliest start times is long. Unfortunately, we were not able to express better necessary conditions to limit the number of calls to

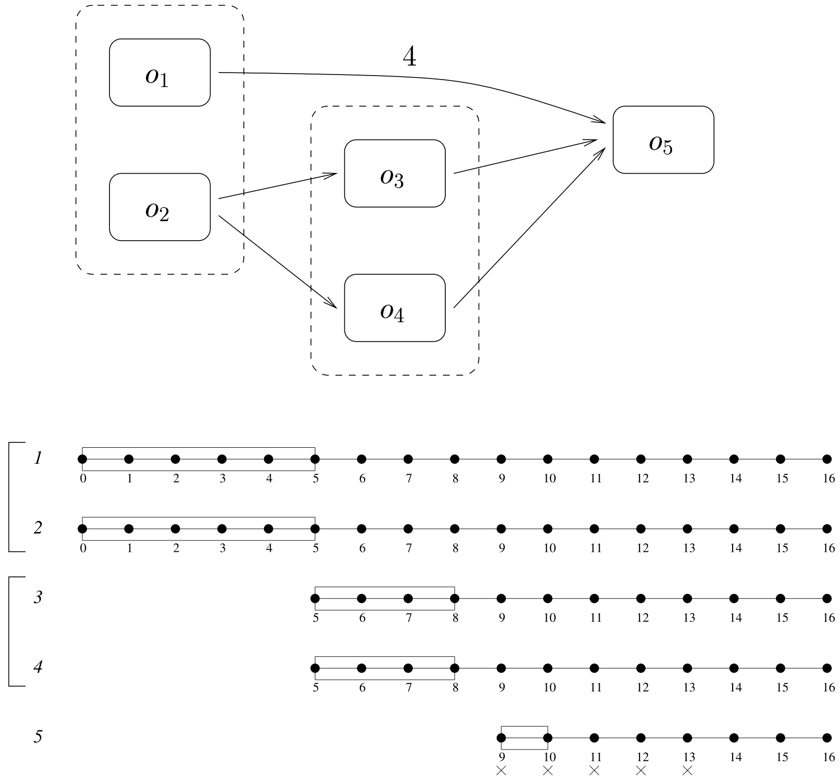

Figure 4: Five operations on three machines and their precedence graph

the adjustment operator so we search for weaker conditions that describe situations in which the $E_k(o, \mu)$-values are likely to have been modified. We tested the two following conditions to call `update_est`(μ, k) :

1. the earliest start time or the latest completion time of an operation on μ has been modified ;
2. a disjunctive constraint between two operations on μ has been fixed.

4 Stronger distances in the disjunctive graph

Figure 4 presents a situation where the three adjustment operators described in §3.2 improve the earliest start time of o_5. However, this adjustment is not the best one that can be found. This section generalizes the ideas presented in Section 3 in order to find still better earliest start times.

In Figure 4, o_1 and o_2 are processed by the machine μ_1 and o_3 and o_4 are processed by μ_2. The critical path method, that ignores the disjunctive constraints finds as earliest start time of o_5 the value $est(o_1) + pt(o_1) + l^0(o1, o5) = 9$. Since the two direct predecessors of o_5, namely o_3 and o_4, are processed on the same machine, o_5 will start after they are both completed, that is after time $5 + 3 + 3 = 11$. By considering operations o_1 and o_2 that are also two predecessors of o_5 with $l^0(o_1, o_5) = 4$ and $l^0(o_2, o_5) = 3$, we know that o_5 cannot start before time $5 + 5 + 3 = 13$. But a better approach of this situation is possible. It is based on the observation that in all feasible schedules, at least 6 time units, corresponding to the duration of o_3 and o_4, will be required between the end of o_2 and the start of o_5. This new "distance" between o_2 and o_5, which is of course more accurate than $l^0(o_2, o_5)$, allows to conclude that o_5 cannot start before $pt(o1) + pt(o_2) + \min(l^0(o1, o_5), 6) = 14$.

In order to formalize such a situation, let us introduce two virtual operations, the source $^{\mathrm{o}}$ and the sink $*$. Their durations are 0 and for any $o \in \mathcal{O}$, we assume that $^{\mathrm{o}} \to o \to *$. Let $\Omega = \mathcal{O} \cup \{^{\mathrm{o}}, *\}$. Clearly, the two functions $o \longmapsto l^0(^{\mathrm{o}}, o)$ and $o \longmapsto T - l^0(*, o)$ are respectively earliest start time and latest completion time functions. For any pair $(o, o') \in \Omega^2$, we define:

$$l^1(o, o') = \max_{\mu \in \mathcal{M}} \left(\max_{S \subseteq \mathcal{O}_\mu} \left(l^0(o, S) + pt(S) + l^0(S, o') \right) \right)$$

By analogy with the proof of Lemma 3.1, $l^1(o, o')$ is shown to be a lower bound for the time between the end of o and the start of o' in any schedule of $S(\omega, T)$. If all the $l^0 - distances$ are known, each $l^1(o, o')$-value can be computed in $O(\sum_{\mu \in \mathcal{M}} |\mathcal{O}_\mu| \log |\mathcal{O}_\mu|)$ or, more simply, in $O(n \log n)$ where n is the number of operations in $\mathcal{O}$. The same arguments are also valid to show that, for any $k \in \mathbb{N}$:

$$l^{k+1}(o, o') = \max_{\mu \in \mathcal{M}} \left(\max_{S \subseteq \mathcal{O}_\mu} \left(l^k(o, S) + pt(S) + l^k(S, o') \right) \right)$$

is a lower bound on the time between the end of o and the start of o'. Clearly, $l^k(o, o')$ increases with k and becomes constant after some K. Let $l^\infty(o, o')$ denote this limit value. All the $l^\infty(o, o')$-values can be directly calculated by an algorithm based on Ford-Bellman's algorithm in $O(n^3 \log n)$ time where $n = |\mathcal{O}|$. Then, for each $o \in \mathcal{O}$, $est(o)$ is set to $\max(est(o), l^\infty(^{\mathrm{o}}, o))$.

As in §3.1, we can introduce deadlines and release dates to compute the lower bound between the end of o and the start of o'. For each machine $\mu \in \mathcal{M}$, if we consider the one-machine problem in which each operation $o_i \in \mathcal{O}_\mu$ has:

- a processing time $p_i = pt(o_i)$
- a release date $r_i = est(o_i)$
- a deadline $d_i = lct(o_i)$
- a *head* $h_i = l^0(o, o_i)$
- a *tail* $t_i = l^0(o_i, o')$

then the minimum value of $\max_{o_i \in \mathcal{O}_\mu}(C_i + q_i) - \min_{o_i \in \mathcal{O}_\mu}(C_i - p_i - h_i)$ over the feasible schedules is a lower bound for the distance from o to o'. This problem is NP-hard but we have shown its preemptive variant can be solved in $O(n^3 \log n)$ time (with $n = |\mathcal{O}_\mu|$).

5 Computational tests

All the algorithms presented in Sections 2, 3 and 4 have been implemented using ILOG SCHEDULER. All tests are done on a PC Pentium II 300 MHz. The experiments reported in this section were performed with the classical job-shop benchmarks in the literature [Fisher & Thompson, 1963; Lawrence, 1984; Adams, Balas & Zawack, 1988; Applegate & Cook, 1991] whose optima are known [Jain & Meeran, 1999]. We also randomly generated 5 10×10 flow-shop instances that we called `fs01`...`fs05`. All of them can be solved in less than one hour by existing algorithms for JSSP. Unless otherwise stated, each experiment in this section reports the computational effort to prove the optimality of an instance.

5.1 Branching schemes

We tested the efficiency of the adjustment operators presented in this paper on two different branching schemes. Both branching schemes are based on the concept of *critical sets*: a set S of operations is critical if and only if $|S| \geq 2$, if all the operations in S are executed on the same machine and if S minimizes the value $lct(S) - est(S) - pt(S)$. For any set S of operations that are processed on a single machine, let us also define $\mathcal{F}(S)$ (resp. $\mathcal{L}(S)$) as the set of operations of S that can be scheduled in first (resp. last) position. More formally:

$$\begin{aligned} \mathcal{F}(S) &= \{o \in S \mid est(o) + pt(S) \leq lct(S - \{o\})\} \\ \mathcal{L}(S) &= \{o \in S \mid est(S - \{o\}) + pt(S) \leq lct(o)\} \end{aligned}$$

The first branching scheme (`BS1`) we implemented is derived from Nuijten & Le Pape [1998]. The machine μ that contains the critical set is selected and is then completely scheduled as follows:

1. Let $\hat{\mathcal{O}}_\mu$ be the set of the unscheduled operations of $\mathcal{O}_\mu$. If $\hat{\mathcal{O}}_\mu = \emptyset$, μ is completely scheduled. Otherwise:
2. Let $n_\mathcal{F} = |\mathcal{F}(\hat{\mathcal{O}}_\mu)|$ and $n_\mathcal{L} = |\mathcal{L}(\hat{\mathcal{O}}_\mu)|$.
3. If $n_\mathcal{F} = n_\mathcal{L}$, we calculate $d_\mathcal{F}$ being the difference between the two smallest earliest start times of the operations in $\mathcal{F}(\hat{\mathcal{O}}_\mu)$ and $d_\mathcal{L}$ being the difference between the two largest latest completion times of the operations in $\mathcal{L}(\hat{\mathcal{O}}_\mu)$.
4. If $n_\mathcal{F} < n_\mathcal{L}$ or $(n_\mathcal{F} = n_\mathcal{L} \ \wedge \ d_\mathcal{F} > d_\mathcal{L})$, create a descendant corresponding to the partial schedule in which the operation in $\mathcal{F}(\hat{\mathcal{O}}_\mu)$ that has minimal earliest start time is ranked before all the other operations of $\hat{\mathcal{O}}_\mu$. In case of ties, the operation with minimal latest completion time is selected. Go to step 1.
5. Otherwise, create a descendant corresponding to the partial schedule in which the operation in $\mathcal{L}(\hat{\mathcal{O}}_\mu)$ that has maximal latest completion time is ranked after all the other operations of $\hat{\mathcal{O}}_\mu$. In case of ties, the operation with maximal earliest start time is selected. Go to step 1.

When a machine is completely scheduled, the process goes on with the (not completely scheduled) machine that contains the critical set. Our reference algorithm based on `BS1` integrates all the adjustment operators presented in [Nuijten & Le Pape, 1998].

The second branching scheme (`BS2`) is derived from [Applegate & Cook, 1991].

1. Let S be a critical set of the partial schedule. If there is no critical set, the partial schedule is a feasible schedule. Otherwise:
2. Let $n_\mathcal{F} = |\mathcal{F}(S)|$ and $n_\mathcal{L} = |\mathcal{L}(S)|$.
3. If $n_\mathcal{F} < n_\mathcal{L}$, create $n_\mathcal{F}$ descendants corresponding to the $n_\mathcal{F}$ partial schedules in which each operation of $\mathcal{F}(S)$ is respectively ranked before all the other operations of S. Go to step 1.

Operator call	*level* for $E_{level}(o,\mu)$ 0		1		2	
	% nodes	% time	% nodes	% time	% nodes	% time
When necessary	11.9	*19200*	27.6	*63.6*	30.4	*65.9*
When new μ-bounds	14.8	*4420*	27.9	*61.9*	30.9	*61.0*
When new μ-precedences	17.8	*2630*	30.2	*53.2*	32.7	*53.3*

Table II: Comparison of the 9 variants of the adjustment operator

4. If $n_{\mathcal{F}} \geq n_{\mathcal{L}}$, create $n_{\mathcal{L}}$ descendants corresponding to the $n_{\mathcal{L}}$ partial schedules in which each operation of $\mathcal{L}(S)$ is respectively ranked after all the other operations of S. Go to step 1.

The reference algorithm based on `BS2` integrates only the adjustment operators of [Carlier & Pinson, 1990] since the other operators usually do not improve performance of this branching scheme.

5.2 The non-precedence conditions

We tested the effect of the two algorithms based on Lemma 2.1 and 2.2 on the algorithm `BS1`. The tests were performed on the twenty first well known job-shop instances of [Lawrence, 1984]. As said, the preliminary results were discouraging: when our two operators are added, the number of backtracks decreases for only a couple of instances and this reduction is less than one percent. This while the computation time is at least three times longer. So even if the results were obtained with basic versions of the operators, based on these results we decided to not search for a better implementation. This is not to say that these adjustment operators can not be useful on other type of shop scheduling problems though.

However, this negative result is in accordance with some remarks of Martin & Shmoys [1996] and Peridy [1996], and it leads to the opinion that the one-machine relaxation due to Carlier & Pinson [1989] and Nuijten & Le Pape [1998] finds most of the deductions that may help in solving instances of the JSSP. We actually think that further improvements should be indeed searched for in adjustment operators that reason on the relations between sets of operations on different machines.

5.3 The multiple-machine based adjustment operator

In Section 3, several variants of the adjustment operator based on the procedure `update_est` were presented. Table II presents the impact of these 9 variants on `BS1`. The 9 variants correspond to the 3 possible values of k to choose the lower bound $E_k(o,\mu)$ for $s(o)$ and to the three "types of events" that trigger the call to `update_est`. For each variant, we present two values. %nodes and %time are respectively the average percentage of the number of nodes and of the time of the branch and bound process. The basis 100 corresponds to the algorithm `BS1`. The average was computed on instances `ft10`, `la19` and `la20`.

Table II shows that the lower bound E_1 is effectively better than E_2 but one computation of E_1 is longer than one computation of E_2. If we compare the global computation time, we can state they are equivalent. E_0 is distinctly a better bound than E_1. Unfortunately, it is very long to be computed since it requires to solve an NP-complete problem.

		BS1		BS1-E1		BS1-E2	
Instance	Size	nodes	*time*	nodes	*time*	nodes	*time*
ft10	10×10	2816	*18.4*	876	*10.0*	920	*9.5*
la19	10×10	3743	*20.5*	1525	*14.8*	1547	*10.7*
la20	10×10	8660	*38.5*	2112	*16.4*	2616	*17.8*
la24	15×10	1092002	*11800*	379443	*7340*	431440	*7280*
la25	15×10	165534	*1800*	76437	*1440*	86009	*1380*
la36	15×15	899	*13.5*	526	*19.7*	519	*16.8*
la40	15×15	40851	*466*	18644	*468*	19739	*437*
abz5	10×10	2604	*15.5*	1227	*13.5*	1297	*13.2*
orb1	10×10	4440	*38.6*	2825	*35.0*	2951	*34.8*
orb2	10×10	4967	*32.4*	1118	*13.2*	1138	*12.8*
orb3	10×10	11987	*91.5*	2878	*37.0*	2990	*35.6*
orb4	10×10	900	*5.6*	367	*4.0*	377	*4.0*
orb5	10×10	1920	*10.9*	582	*5.7*	586	*5.3*
JSSP average		100.0	*100.0*	39.1	*72.4*	41.4	*67.4*
fs01	10×10	3154267	*12660*	693594	*3980*	703316	*3732*
fs02	10×10	49453	*356*	24813	*275*	25761	*262*
fs03	10×10	15703	*49.2*	2522	*11.2*	2524	*11.0*
fs04	10×10	9203	*50.2*	3901	*30.5*	4048	*29.3*
fs05	10×10	3462	*21.6*	1445	*10.3*	1454	*9.7*
FSSP average		100.0	*100.0*	34.5	*48.0*	35.3	*47.7*

Table III: BS1 based algorithms

		BS2		BS2-E1		BS2-E2	
Instance	Size	nodes	*time*	nodes	*time*	nodes	*time*
ft10	10×10	4751	*21.2*	2193	*19.6*	2368	*19.3*
la19	10×10	2925	*8.8*	2240	*10.7*	2309	*10.2*
la20	10×10	482	*1.8*	404	*2.2*	414	*2.0*
la24	15×10	205259	*1960*	101585	*1939*	235317	*2355*
la25	15×10	51043	*662*	37849	*797*	40231	*618*
la36	15×15	780	*13.6*	643	*24.2*	657	*20.0*
la40	15×15	15780	*196*	12448	*341*	13581	*286*
abz5	10×10	3099	*13.3*	1898	*11.4*	1976	*10.7*
orb1	10×10	24905	*117*	16434	*125*	17020	*116*
orb2	10×10	5005	*17.3*	1286	*7.8*	1359	*7.4*
orb3	10×10	19576	*89.2*	12523	*96.0*	13262	*90.0*
orb4	10×10	2978	*10.4*	2210	*11.5*	2279	*11.1*
orb5	10×10	867	*3.7*	707	*4.6*	722	*4.3*
JSSP average		100.0	*100.0*	66.5	*114*	74.2	*105*
fs01	10×10	106906	*455*	69824	*385*	70549	*361*
fs02	10×10	168340	*820*	103950	*699*	105620	*657*
fs03	10×10	2900	*10.6*	1753	*9.6*	1759	*9.0*
fs04	10×10	7347	*28.2*	3777	*22.0*	3829	*19.9*
fs05	10×10	5057	*24.2*	3795	*23.3*	3867	*22.7*
FSSP average		100.0	*100.0*	62.8	*86.9*	63.6	*81.7*

Table IV: BS2 based algorithms

This table also shows that calling `update_est` only when a disjunction is set on machine μ is a good heuristic: the number of nodes is scarcely increased by the relaxation of the constraint so that the CPU time is distinctly decreased. Thus, in all the other tests, we applied only the two variants with E_1 or E_2; in both variants, `update_est`$(\mu, level)$ is called only when a disjunction is fixed on the machine μ.

BS1-E1 and BS1-E2 are the two algorithms derived from BS1 by integrating the procedure `update_est`. The performance of these three algorithms are compared in Table III.

		BS1+Shaving		BS1-E2+shaving	
Instance	Size	nodes	*time*	nodes	*time*
ft10	10×10	94	*40.0*	51	*40.3*
la19	10×10	135	*38.4*	64	*33.4*
la20	10×10	74	*21.0*	36	*18.7*
la24	15×10	4293	*2460*	2252	*2250*
la25	15×10	5762	*3560*	4069	*4280*
la39	15×15	10096	*9630*	863	*2472*
la40	15×15	357	*325*	247	*579*
fs01	10×10	4036	*1571*	2897	*1788*
fs02	10×10	2037	*838*	1110	*762*
fs03	10×10	109	*34.0*	68	*36.6*
fs04	10×10	206	*77.5*	129	*76.7*
fs05	10×10	195	*105*	139	*111*
	average	100.0	*100.0*	56.1	*100.8*

Table V: Shaving based algorithms

Similarly, Table IV reports the performance of BS2 and its two variants BS2-E1 and BS2-E2. In both tables, the CPU times are measured in seconds and the "average" line reports the mean deviation of the number of nodes and of the computation time between the three algorithms, scaled respectively on a basis of 100 nodes and 100 seconds.

For all the tested instances, the number of nodes is significantly decreased. In many cases, it is divided by a factor greater than 2.

We can draw several other conclusions from these tables. Firstly, if we compare the reduction of the number of nodes for the two branching schemes, we observe that the impact of the new adjustment operator is better with BS1 than with BS2. As a result, whereas BS1-E1 and BS1-E2 run faster than BS1, the running time of the three algorithms based on BS2 are equivalent. This can be explained by the fact that BS2 takes more carefully advantage of the sub-problem structure than BS1. BS2 finds more quickly the inconsistencies of the sub-problems. In some situations, BS1 can be unable to detect such inconsistencies. The new adjustment operator can help it to discover them. More generally, we can assume that good lower bounds help bad branching schemes more than they help good branching schemes. The results also show there still is some work to be done on improving the efficiency of the used adjustment operators.

Furthermore, for both BS1 and BS2, it clearly appears that the new adjustment operator is more efficient with flow-shop instances (or with job-shop instances that "look like" flow-shop such as ft10). This result is not surprising : in JSSP, the predecessors of an operation o can be processed on any machine whereas in FSSP, they are processed only on the machines that precede $M(o)$. As a consequence, in each one-machine sub-problem, there are more operations with positive tails, which makes the bound better.

We also tested the performance of the adjustment operator in combination with shaving techniques [Carlier & Pinson, 1994; Martin & Shmoys, 1996]. Table V compares two variants of BS1 and BS1-E2 in which we added a procedure to at each node shave on the ordering of the operations (rather than on the time bounds). With the new adjustment operator, the number of nodes is decreased but processing times of both algorithms are equivalent. This again shows that the new adjustment operators are effective, but some improvement of their efficiency would be desirable. We can remark

Instance	Size	LB	UB	nodes	*time*
`rec09`	20×10	1520	1520	297599	*4301.7*
`rec13`	20×15	1836	(1930)	6108741	*206289*
`yam1`	20×20	851	(888)	3567728	*165422*
`yam3`	20×20	842	(893)	10390469	*783212*
`yam4`	20×20	922	(968)	1861649	*210472*

Table VI: New bounds for some flow-shop and job-shop instances.

that the performances are similar for job-shop and flow-shop instances.

With `BS2-E2`, we solved an open flow-shop instance [Reeves, 1995]. With a variant of `BS2-E2` with shaving, we improved some lower bounds of job-shop instances [Yamada & Nakano, 1992]. Results are reported in Table VI. Values in brackets are actually known upper bounds.

5.4 Distances under disjunctive constraints

If in `BS1` or `BS2` we integrate the adjustment operator based on the distances between operations that is presented in Section 4, we can observe that the number of nodes in the search trees is about 5% lower than the number of nodes of `BS1-E1`. In our tests, l^1 is often better than l^0 but, due to the small sizes of the test instances, there are only very few differences between l^1 and l^∞. In any case this operator seems to be too time consuming to be efficient in a branch-and-bound scheme.

6 Conclusion

We have presented new adjustment operators for the JSSP and the FSSP. Extensive computational tests are done that show that especially the adjustment operators that simultaneously consider operations on multiple machines by reasoning on the combination of resource constraints and precedence constraints are very effective for both these problems. Using these operators we succeeded in improving on the best known lower bounds of some job-shop and flow-shop instances known from the literature, including the closing of an open flow-shop instance.

Negative results on a new adjustment operator that is based on a one-machine relaxation, in combination with the already existing impressive body of work on one-machine relaxations, strengthened our belief that the multiple-machine operators like we introduced in this paper constitute the most promising path to improve performance on shop scheduling problems.

One thing we want to do in the future is to improve the efficiency of the used adjustment operators. One way of doing this is by finding additional ways to efficiently predict that an operator will not be able to make any deductions. Another possibility is to heuristically do such a prediction. We are also working on including other properties like setup times between operations in the operators.

Acknowledgements

The authors are indebted to Philippe Chrétienne for his careful reading of drafts of the paper. The work of the first author is partially financed by ILOG S.A., under research contract ILOG/UPCM no. 980220.

References

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

Applegate, D., and W. Cook [1991], A computational study of the job-shop scheduling problem, *ORSA Journal on Computing* **3**, 149–156.

Baker, K.R., E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan [1983], Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints, *Operations Research* **26**, 111–120.

Baptiste, P., and C. Le Pape [1996], Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling, *Proc. 15th Workshop of the U.K. Planning Special Interest Group.*

Blazewicz, J., W. Domschke, and E. Pesch [1996], The job shop scheduling problem: Conventional and new solution techniques, *European Journal of Operational Research* **91**, 1–33.

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

Carlier, J., and E. Pinson [1989], An algorithm for solving the job-shop problem, *Management Science* **35**, 164–176.

Carlier, J., and E. Pinson [1990], A practical use of Jackson's preemptive schedule for solving the job shop problem, *Annals of Operations Research* **26**, 269–287.

Carlier, J., and E. Pinson [1994], Adjustment of heads and tails for the job-shop problem, *European Journal of Operational Research* **78**, 146–161.

Fisher, H., and G.L. Thompson [1963], Probabilistic learning combinations of local job-shop scheduling rules, in: J.F. Muth and G.L. Thompson (eds.), *Industrial Scheduling*, Prentice Hall.

Garey, M.R., D.S. Johnson, and R. Sethi [1976], The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research* **1**, 117–129.

Graham, R.E., E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan [1979], Optimization and approximation in deterministic sequencing and scheduling, *Annals of Discrete Mathematics* **4**, 287–326.

Jain, A.S., and S. Meeran [1999], Deterministic job-shop scheduling : Past, present and future, *European Journal of Operational Research* **113**, 390–434.

Lawrence, S. [1984], *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Martin, P.D., and D.B. Shmoys [1996], A new approach to computing optimal schedules for the job shop scheduling problem, in: S.T. McCormick W.H. Curnigham and M. Queyranne (eds.), *Proceedings fifth international IPCO conference, Vancouver, Canada*, LNCS 1084, 389–403.

Nuijten, W.P.M. [1994], *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*, Ph.D. thesis, Eindhoven University of Technology.

Nuijten, W., and C. Le Pape [1998], Constraint-based job shop scheduling with ilog scheduler, *Journal of Heuristics* **3**, 271–286.

Peridy, L. [1996], *Le problème de job-shop : arbitrage et ajustements*, Ph.D. thesis, Université de Technologie de Compiègne, in French.

Reeves, C.R. [1995], A genetic algorithm for flowshop sequencing, *Computers and Operations Research* **22**, 5–13.

Roy, B., and B. Sussmann [1964], *Les problèmes d'ordonnancement avec contraintes disjonctives*, Note DS No. 9 bis, SEMA, Paris, France.

Sourd, F., and W. Nuijten [1999], *Scheduling with tails and deadlines*, Technical Report

1999-032, LIP6, Université Pierre et Marie Curie, Paris, France.

YAMADA, T., AND R. NAKANO [1992], A genetic algorithm applicable to large-scale job-shop problems, in: R. Männer and B. Manderick (eds.), *Proceedings of the 2nd International Workshop on Parallel Problem Solving from Nature*, 281–290.