ELSEVIER

European Journal of Operational Research 78 (1994) 146–161

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Adjustment of heads and tails for the job-shop problem

J. Carlier [a,*], E. Pinson [b]

[a] URA CNRS, HEUDIASYC, Université de Technologie de Compiègne, 60200 Compiègne Cedex, France
[b] Institut de Mathématiques Appliquées, Université Catholique de l'Ouest, B.P. 808, 49008 Angers Cedex 01, France

Abstract

The efficiency of recent enumerative methods for the job-shop problem crucially depends on immediate selections of disjunctive constraints leading to adjustment of heads and tails. This paper presents new investigations concerning this powerful tool. More efficient algorithms are proposed, and global operations are introduced. We also describe a new lower bound and a new branching scheme which are used to design a branch and bound method. Computational results show that these techniques permit to drastically reduce the size of the search trees.

Keywords: Scheduling; Job-shop; Branch and bound method

1. Introduction

In the job-shop problem, n jobs have to be processed on m machines, subject to both conjunctive and disjunctive constraints, in order to minimize the makespan (Muth and Thompson, 1963). It is an NP-hard problem in the strong sense (Lenstra et al., 1977), which remains probably one of the most computationally intractable combinatorial problem to date. This fact justifies the considerable amount of study which this problem has been subject to over the years (see the partial reference list).

In a previous paper (Carlier and Pinson, 1989), we proposed an efficient enumerative method which solved for the first time a particular instance with 10 jobs and 10 machines proposed by Muth and Thompson in 1963. In Carlier and Pinson (1990), we introduced the concept of immediate selection and explained how it permits to adjust heads and tails and to select directly some disjunctions. We also designed a polynomial algorithm for optimally adjusting heads and tails and consequently fixing disjunctive constraints, allowing us to efficiently prune the search tree associated with a new branch and bound method. This basic idea has been recently used by Brucker, Jurisch and Sievers (1991) and Brucker, Jurisch and Kramer (1992) for designing new enumerative methods for this problem. With some modifications leading to weaker conditions than the ones we proposed, they obtain an O(max $[d, n \log_2 n]$)-steps algorithm for adjusting heads and tails (d denotes the number of selected disjunctive constraints). They also propose new immediate selections and efficient algorithms to compute them.

* Corresponding author.

SSDI 0377-2217(94)00077-P

Applegate and Cook (1990), using a mixed integer linear programming formulation of the problem combined with cutting plane generations, some of them built on the same concept, obtained for the first time to the best of our knowledge, interesting computational results with such an approach.

The results obtained for all these approaches seem to prove the prime interest of the concept of immediate selection. So, the aim of this paper is to present some new investigations concerning this powerful tool.

The paper is organized as follows. In Section 2, we recall some backgrounds and basic results relying on the job-shop problem. In Section 3, we present the key idea of immediate selection and adjustment of heads and tails. We show how such computations can be performed both on local and global levels. On local level, we propose more efficient algorithms for optimally adjusting heads and tails and exhibiting immediate selections. These computations are called local operations. On global level, we propose a method for performing new adjustments of heads and tails, and using the local operations defined before. These computations are called global operations and are detailed in Section 4. In Section 5, we propose a new branching scheme and a new lower bound leading to a more efficient Branch & Bound method. This new enumerative method allowed us to solve the 10 jobs and 10 machines of Muth and Thompson developing a search tree with only 35 nodes. Moreover, four other open instances proposed by Adams et al. in 1986 are solved with this new method. Complementary computational experiments are reported.

2. Problem statement, background, basic notations and properties

2.1. The job-shop problem

In the job-shop problem, n jobs $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_n$ have to be processed on m different machines $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_m$. Each job $\mathcal{J}_j$ consists of a sequence of operations which have to be scheduled in a given order. Moreover, each operation can be processed only by one machine among the m available ones. Preemption of any operation is not allowed. The objective is to find an operating sequence for each machine so as to minimize the maximum of the completion times $C_{\max}$ of the jobs.

2.2. Disjunctive graph

Two operations i and j, executed by the same machine, cannot be simultaneously processed. So we associate with them a pair of disjunctive arcs $[i, j] = \{(i, j),(j, i)\}$. The problem is then modelled by a disjunctive graph $\mathcal{G} = (G, D)$, where $G = (X, U)$ is a conjunctive graph and D a set of disjunctions. Fig. 1 shows an example of disjunctive graph associated with a job-shop with 3 jobs and 3 machines.

2.3. Basic notations

In this paper, p_i denotes the processing time of operation i, r_i (resp. q_i) its head (resp. its tail). Denoting the value of one of the longest paths from i to j in G by $l(i, j)$, we have: $r_i = l(\mathrm{o}, i)$ and $q_i = l(i, *) - p_i$, where o and $*$ are the source and the sink in G.

2.4. Schedule

A schedule on a disjunctive graph $\mathcal{G} = (G, D)$ is a set of starting times $T = \{t_i \mid i \in X\}$ such that:
- the conjunctive constraints are satisfied:

$$t_j - t_i \geq p_i \quad \forall (i, j) \in U,$$

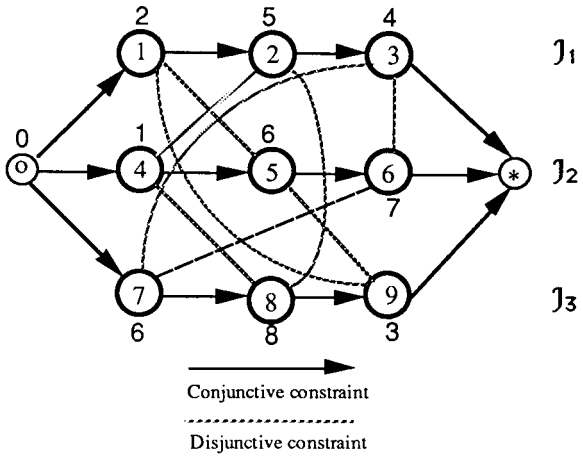

Fig. 1. A 3×3 job-shop instance.

– the disjunctive constraints are satisfied:

$t_j - t_i \geq p_i$ or $t_i - t_j \geq p_j$ $\forall (i, j) \in D$.

2.5. Selection

To build a schedule, we have to select the disjunctive constraints and thus to choose an operating sequence for each machine.

A *selection* A is a set of disjunctive arcs such that if $(i, j) \in A$, then $(j, i) \notin A$. The membership of (i, j) to A makes necessary to process operation i before operation j. We associate the conjunctive graph $G_A = (X, U \cup A)$ with selection A. By definition, a selection is complete if all the disjunctions of D are selected. It is consistent if the associated conjunctive graph is acyclic. A schedule corresponds to a consistent complete selection. Its makespan is the value of one of the longest path in G_A.

2.6. Solution

Let UB be a given integer. By definition, a *solution* of the job-shop problem is a schedule with a makespan smaller or equal than UB.

2.7. Clique of disjunctions

A *clique of disjunctions* is a set K of operations such that every couple of operations of K is in disjunction. Cliques are obtained for the job-shop by considering a group of operations processed on a specific machine. $e \in K$ (resp. $s \in K$) is called the input (resp. output) of the clique K if e (resp. s) is sequenced in T before (resp. after) all the other operations of K.

2.8. Jackson's preemptive schedule: Primal and dual version

Let us consider a subset I of operations that have to be processed on the machine. Jackson's preemptive schedule is the list schedule associated with the Most Work Remaining (MWR) priority

i	1	2	3	4	5	6
r_i	4	0	9	15	20	21
p_i	6	8	4	5	8	8
q_i	20	25	30	9	14	16

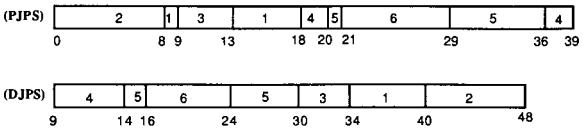

Fig. 2.

dispatching rule. To build it, we schedule, at the first moment t where the machine and at least one operation are available, the available operation with maximal tail: This operation is processed either up to its completion or until a more urgent operation becomes available. We update t, and iterate until all the operations are scheduled (Jackson, 1955).

Because of the symmetric role of heads and tails, we can design in the same way a symmetric version of this algorithm we call *Dual Jackson's Preemptive Schedule* (DJPS), in opposition with the previous version we call for this reason *Primal Jackson's Preemptive Schedule* (PJPS).

By using heap structures, both (PJPS) and (DJPS) can be computed in $O(n \log_2 n)$ steps.

Their makespan are lower bounds of the job shop problem, and are equal to $\text{Max}_{J \subseteq I} h(J)$, where (Carlier, 1982)

$$h(J) = \underset{j \in J}{\text{Min}}\, r_j + \sum_{j \in J} p_j + \underset{j \in J}{\text{Min}}\, q_j. \tag{1}$$

Fig. 2 shows an example of (PJPS) and (DJPS) built on 6 operations.

2.9. Basic property on (PJPS) and (DJPS)

From the construction of (PJPS) (resp. (DJPS)), we can deduce the following result. First, let us introduce some complementary notations: for any operation j of I, we denote by:

- t_j (resp. t'_j) its starting time in (PJPS) (resp. (DJPS));
- C_j (resp. C'_j) its completion time in (PJPS) (resp. (DJPS)).

Proposition 1. *Let $j \in I$. There exists a subset $K_j \subseteq I$ (resp. $K'_j \subseteq I$) s.t.:*

- $j \in K_j$ *(resp.* $j \in K'_j$*)*;
- $\forall k \in K_j$, $q_j \leq q_k$ *(resp.* $\forall k \in K'_j$, $r_j \leq r_k$*)*;
- $C_j = \text{Min}_{j \in K_j} r_j + \Sigma_{j \in K_j} p_j$. *(resp.* $C'_j = \text{Min}_{j \in K'_j}\, q_j + \Sigma_{j \in K'_j}\, p_j$*)*.

Proof. See Carlier and Pinson (1990). □

For the example above (see Fig. 2), we have for instance $K_1 = \{1, 2, 3\}$ and $K'_1 = \{1, 3, 5, 6\}$.

2.10. Immediate selections

Let us consider a non-selected disjunctive constraint $[i, j]$. If we can prove that in any solution, operation j is processed after (resp. before) operation i, we can select the disjunctive constraint $[i, j]$ in $i \rightarrow j$ (resp. $j \rightarrow i$) direction and we say that we have an immediate selection.

2.11. Adjustment of heads and tails

Let c be an operation. If we can prove that in any solution, $r_c \geq \alpha$ ($q_c \geq \beta$), then we can set $r_c = \alpha$ (resp. $q_c = \beta$) and we say that we have an adjustment of head (resp. tail).

As we pointed out in Carlier and Pinson (1990), adjustments of heads and tails are of prime importance for efficiently solving the job-shop problem, because they allow, via the immediate selections on disjunctive constraints, to exhibit more quickly a complete and consistent selection for the problem, and so a solution.

3. Local operations

3.1. Introduction

One of the most efficient way of solving the job-shop problem consists of relaxing it to m one-machine scheduling problems linked by the precedence constraints associated with jobs sequences, and thus to solve them sequentially or parallelly. In this section, we recall a way for determining immediate selections that we proposed in Carlier and Pinson (1990), namely immediate selection on disjunctions and immediate selection on ascendant sets. These operations being performed on each clique of disjunctions, we call them local operations. In this section, I denotes a subset of operations associated with a given machine.

3.2. Immediate selection on a disjunction

In Carlier and Pinson (1989), we proved the following simple result:

Proposition 2. *Let* $[i, j]$ *be a non-selected disjunctive constraints on* I. *If*

$$r_j + p_j + p_i + q_i > \mathrm{UB}, \tag{2}$$

then it is scheduled before j *in any solution, and we can select* $[i, j]$ *in the* $i \to j$ *direction and set* $r_j = \max(r_j, r_i + p_i)$ *and* $q_i = \max(q_i, q_j + p_j)$.

We propose below a procedure allowing the determination of all immediate selections on disjunctive constraints associated with operations of I in $O(n \log_2 n)$ steps. In this algorithm, $\mathscr{L}_1$ (resp. $\mathscr{L}_2$) denotes the ordered list on increasing (resp. decreasing) $(q_i + p_i)$-values (resp. $(r_i + p_i)$-values) $(i \in I)$. i_1 (resp. i_2) denotes the operation matching the current minimum (resp. maximum) value over $\mathscr{L}_1$ (resp. $\mathscr{L}_2$).

Procedure Disj(I)
While $\mathscr{L}_1 \neq \emptyset$ and $\mathscr{L}_2 \neq \emptyset$ do
  If $p_{i_1} + q_{i_1} \leq \mathrm{UB} - (r_{i_2} + p_{i_2})$ Then
    $\mathscr{L}_1 := \mathscr{L}_1 \setminus \{i_1\}$
  Else
    $r_{i_2} := \max\{r_{i_2}; \max_{i \in \mathscr{L}_1 \setminus \{i_2\}}[r_i + p_i]\}$
    $\mathscr{L}_2 := \mathscr{L}_2 \setminus \{i_2\}$
  Endif
Enddo

For i_2, Disj(I) computes the smallest value over $\mathscr{L}_1$ such that i_1 precedes i_2 because of Proposition 2. Consequently, all immediate selections on disjunctions involving i_2 can be deduced.

Proposition 3. *Algorithm* Disj(I) *runs in* $O(n \log_2 n)$.

Proof. Clearly, as $|\mathscr{L}_1| = n$ and $|\mathscr{L}_2| = n$, algorithm Disj(I) runs in $O(n \log_2 n)$ steps using a heap structure for the adjustment of r_{i_2}-values. □

3.3. Immediate selections on an ascendant set

3.3.1. Previous results

All the definitions and results of this section can be found in Carlier and Pinson (1990). In the sequel, and without loss of generality, we make the assumption that all heads and tails are different. In case of ties, we simply break them by taking into account the numbering of the operations in the disjunctive graph.

By definition, a subset J of operations is called an *ascendant set* of c if $c \notin J$ and

$$\operatorname*{Min}_{j \in J \cup \{c\}} r_j + \sum_{j \in J \cup \{c\}} p_j + \operatorname*{Min}_{j \in J} q_j > \mathrm{UB}. \tag{3}$$

Clearly, if J is an ascendant set of c, c is output of $K = J \cup \{c\}$, and we can set

$$r_c := \mathrm{Max}(r_c, \alpha_c), \quad \text{where} \quad \alpha_c = \operatorname*{Max}_{J' \subseteq J} \left[\operatorname*{Min}_{j \in J'} r_j + \sum_{j \in J'} p_j \right].$$

Let us introduce some complementary notations:

- for $j \in I$, p_j^- is the processed time of j before r_c in (PJPS), p_j^+ is the processed time of j after r_c in (PJPS);
- $K_c^- = \{j \in I \mid p_j^+ = 0\}$; $K_c^+ = \{j \in I \mid p_j^+ > 0\}$.

We proved that for finding α_c, we can look for a non-empty subset K^+ of K_c^+ satisfying

$$r_c + p_c + \sum_{j \in K^+} p_j^+ + \operatorname*{Min}_{y \in K^+} q_j > \mathrm{UB}. \tag{4}$$

If such a set exists, there exists a maximal set K_c^* satisfying (4), and we have the maximal adjustment

$$\alpha_c = \operatorname*{Max}_{j \in K_c^*} C_j.$$

These results can of course be transposed to the symmetric case, that is to say to the search of descendant sets for operation c. We also proposed the following simple procedure for finding K_c^*:

- *Build* (PJPS) *up to* $t = r_c$.
- *Take the operations of* K_c^+ *in the increasing order of the* q_j, *and find the first one* s *such that*

$$r_c + p_c + \sum_{\{j \in K_c^+ \mid q_j \geq q_s\}} p_j^+ + q_s > \mathrm{UB} \quad (\textit{if any exists}).$$

- *Define* $K_c^* = \{j \in K_c^+ \mid q_j \geq q_s\}$.

In the next section, we show that, using a specific data structure, s and K_c^* can be computed in $O(\log_2 n)$ steps, leading to an overall $O(n \log_2 n)$-step algorithm for optimally adjusting heads which we describe in Section 3.3.4.

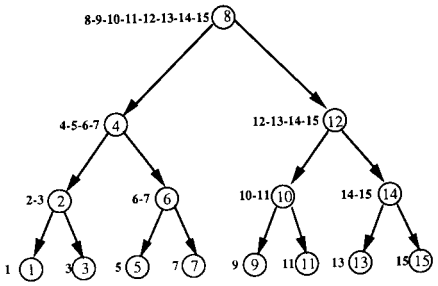

Fig. 3.

3.3.2. Data structure for computing K_c^*

Let us suppose that (PJPS) has been built up to time instant $t = r_c$. From the previous section, finding the maximal ascendant set for operation c (if any exists) is equivalent to finding an operation s of K_c^+ such that:

- $$r_c + p_c + \sum_{\{j \in K_c^+ | q_j \geq q_s\}} p_j^+ + q_s > \mathrm{UB}, \tag{5}$$
- q_s is minimal. (6)

In the following, we denote by s_c the operation satisfying (5)–(6) for operation c. By convention s_c will be set to $+\infty$ if K_c^* does not exist.

Let us consider the balance binary search tree $\mathscr{C}_\mathrm{I}$ built on the operations of I using a numerical key induced by the q_i-values. For a seek of clarity, and without loss of generality, we assume that the operations have been renumbered in the order of increasing q_i-values. So, we have

$$q_1 < q_2 < \cdots < q_n.$$

For instance, for $n = 15$ operations, we obtain the binary tree of Fig. 3 (here, the balance is equal to zero).

In this search tree, we denote for any node k associated with operation k:
$k_\mathrm{l}, k_\mathrm{r}, k_\mathrm{f}$: Its left successor, its right successor and its predecessor in $\mathscr{C}_\mathrm{I}$.
$\mathscr{L}_k, \mathscr{R}_k$: Its associated left and right subtrees.
$\mathscr{T}_k$: The subtree of root k.

By convention, $\mathrm{k_l}$ (resp. k_r, k_f) will be set to 0 if the left successor (resp. right successor, predecessor) of node k does not exist. $\mathscr{C}_I$ verifies the property

$$\forall k \in]n], \quad \forall i \in \mathscr{L}_k, \quad \forall j \in \mathscr{R}_k, \qquad q_i < q_k < q_j.$$

Moreover, we denote by:
v: The root of $\mathscr{C}_\mathrm{I}$.
$\mathscr{N}$: The set of leaves of $\mathscr{C}_I$.
$\mathscr{S}$: The set of operations j from I satisfying $p_j^+ > 0$.
With the convention above, we have $\mathscr{N} = \{k \in I | k_\mathrm{l} = k_\mathrm{r} = 0\}$.

Now, let us assign to any node k, in addition to its key q_k, the following quantities,

$$p_k^+,$$

$$\sigma_k = p_k^+ + \sum_{j \in \mathscr{R}_k} p_j^+, \tag{7}$$

$$\xi_k = \begin{cases} \max\limits_{j \in \mathscr{T}_k \cap \mathscr{S}} \left[q_j + \sum\limits_{\{i \in \mathscr{T}_k \mid q_i \geq q_j\}} p_i^+ \right] & \text{if } \mathscr{T}_k \cap \mathscr{S} \neq \emptyset, \\ -\infty & \text{otherwise.} \end{cases} \tag{8}$$

p_k^+, σ_k and ξ_k ($k \in]n]$) will be used for computing K_c^*.

As p_j^+-values decrease during (PJPS) construction, σ_k- and ξ_k-values will be modified at the same time. Next section deals with the initialization and the updating of these quantities in case of variation of p_j^+-values.

3.3.3. Computing the quantities σ_k, ξ_k and η_k.

From (7)–(8), we can deduce the following recursive definitions:

$$\sigma_k = \begin{cases} p_k^+ & \text{if } k_\mathrm{l} = k_\mathrm{r} = 0, \\ p_k^+ + \tau_{k_\mathrm{r}} & \text{otherwise,} \end{cases}$$

with the convention that $\sigma_0 = 0$.

$$\xi_k = \begin{cases} q_k + \sigma_k & \text{if } k_\mathrm{l} = k_\mathrm{r} = 0 \text{ and } p_k^+ > 0, \\ \max\left[\sigma_k + \xi_{k_\mathrm{l}}, \xi_{k_\mathrm{r}}\right] & \text{if } p_k^+ = 0, \\ \max\left[\xi_{k_\mathrm{l}} + \sigma_k, q_k + \sigma_k, \xi_{k_\mathrm{r}}\right] & \text{otherwise,} \end{cases}$$

with the convention that $\xi_0 = -\infty$.

Indeed, we have:

$$\xi_{k_\mathrm{l}} = \max_{j \in \mathscr{L}_k \cap \mathscr{S}} \left[q_j + \sum_{\{i \in \mathscr{L}_k \mid q_i \geq q_j\}} p_i^+ \right],$$

$$\xi_{k_\mathrm{r}} = \max_{j \in \mathscr{R}_k \cap \mathscr{S}} \left[q_j + \sum_{\{i \in \mathscr{R}_k \mid q_i \geq q_j\}} p_i^+ \right].$$

In particular, it is straightforward to check that

$$\xi_v = \max_{j \in \mathscr{S}} \left[q_j + \sum_{\{i \in I \mid q_i \geq q_j\}} p_i^+ \right]$$

σ_k- and ξ_k-values for $k \in I$ can be initialized by the following procedure. Of course, we have, at the beginning of the processing: $\forall j \in I$, $p_j^+ = p_j > 0$. $\mathscr{F}$ denotes a FIFO storage structure.

Procedure Initialize(σ,ξ)
Build $\mathscr{C}_I$
$\mathscr{F} = \mathscr{N}$, $\sigma_0 = 0$, $\xi_0 = -\infty$, $\tau_0 = 0$
While $\mathscr{F} \neq \{0\}$ *do*
  Let k be the first element in $\mathscr{F}$
  $\sigma_k = p_k + \tau_{k_\mathrm{r}}$, $\xi_k = \max[\xi_{k_\mathrm{l}} + \sigma_k, q_k + \sigma_k, \xi_{k_\mathrm{r}}]$
  $\mathscr{F} = \mathscr{F} \cup \{k_f\} / \{k\}$
Enddo

As we saw previously, p_k^+-values decrease and become null during the construction of (PJPS). Procedure Update (σ, ξ, k, ε) performs the updating of σ_j and ξ_j quantities after a variation (positive or negative) ε of p_k^+ for a given operation k:

Procedure Update $(\sigma,\xi,k,\varepsilon)$
$\sigma_k = \sigma_k + \varepsilon$
If $p_k^+ = 0$ *then*
  $\xi_k = \max[\xi_{k_l} + \sigma_k,\ \xi_{k_r}]$
Else
  $\xi_k = \max[\xi_{k_l} + \sigma_k,\ q_k + \sigma_k,\ \xi_{k_r}]$
Endif
$j = k$
While $j \neq 0$ *do*
  $j = j_f$
  if $q_j < q_k$ *then*
    $\sigma_j = \sigma_j + \varepsilon$
  Endif
  If $p_j^+ = 0$ *then*
    $\xi_j = \max[\xi_{jl} + \sigma_j,\ \xi_{jr}]$
  Else
    $\xi_j = \max[\xi_{jl} + \sigma_j,\ q_j + \sigma_j,\ \xi_{jr}]$
  Endif
Enddo

Procedure update simply performs a backtrack from node k to the root v in $\mathscr{C}_I$ applying the recursive relations defining σ_j and ξ_j $(j \in I)$.

Now, let us suppose again that (PJPS) has been built up to $t = r_c$. Let us denote by p_k^+, σ_k, and ξ_k $(k \in I)$ the current values of the quantities defined above. Procedure Find(s_c) allows to determine the operation s_c defined in (5)–(6).

Procedure Find(s_c)
Update$(\sigma,\xi,c,-p_c)$
$\delta = \mathrm{UB} - (r_c + p_c)$
$s_c = +\infty$
$k = v$
While $\xi_k > \delta$ *do*
  If $\xi_{k_l} + \sigma_k > \delta$ *then*
    $\delta = \delta - \sigma_k,\ k = k_l$
  Else
    If $(q_k + \sigma_k > \delta)$ *and* $(p_k^+ \neq 0)$ *then*
      $s_c = k,\ k = 0$
    Else
      $k = k_r$
    Endif
  Endif
enddo
Update(σ,ξ,c,p_c)

Procedure Find(s_c) starts from the root $k = v$ of $\mathscr{C}_I$. If $\xi_k > \delta = \mathrm{UB} - (r_c + p_c)$, then there exists an ascendant set for operation c, and s_c is defined. If $\xi_{k_l} + \sigma_k > \delta$, then at least one operation in $\mathscr{L}_k$ verifies (5)–(6). We set $k = k_l$ and $\delta = \delta - \sigma_k$ in order to scan this subtree. If not, if $q_k + \sigma_k > \delta$, then k

is clearly the operation with minimal tail satisfying (5)–(6), and we stop the procedure with $s_c = k$. Otherwise, s_c is located in the subtree $\mathscr{R}_k$ and we set $k = k_r$ in order to explore it.

Proposition 4. a) *Procedure* Initialize (σ,ξ) *runs in* $O(n)$ *steps.*
b)*Procedures* Update (σ,ξ,k,ϵ) *and* Find(s_c) *run in* $O(\log_2 n)$ *steps.*

Proof. a): Evident by construction.

b): Procedure Update works by performing a backtrack from node k to the root v in $\mathscr{C}_I$. Procedure Find follows the single path connecting the root v to node s_c in $\mathscr{C}_I$. As the depth of such a balanced binary tree is $\log_2 n$, the result holds. □

Once s_c is computed, we can define $K_c^* = \{j \in K_c^+ | q_j \geq q_{s_c}\}$, and then adjust r_c by setting

$$r_c = \alpha_c \quad \text{with } \alpha_c = \max_{y \in K_c^*} C_j.$$

This adjustment simply forces operation c to be processed after the completion of all operations j of K_c^*. For (PJPS), this is equivalent to forbid the process of operation c until we have: $\forall j \in K_c^*$, $p_j^+ = 0$, and thus until

$$\forall j \in \{i \in I | q_i \geq q_{s_c}\}, \quad p_j^+ = 0. \tag{9}$$

Clearly, relation (9) is equivalent to

$$\nu < s_c, \tag{10}$$

where $\nu = \max_{k \in I}[k \mid p_k^+ > 0] = \max_{k \in \mathscr{S}}[k]$.

If t denotes the first time instant in (PJPS) for which (10) is satisfied, then we have $r_c = \alpha_c = t$. This simple property will be used in the algorithm designed in next section to avoid the explicit computation of α_c-values.

Now, we propose an algorithm for optimally adjusting heads on operations of I, based on the principle explained in Section 3.3.1, and which uses the above data structure and the related procedures for determining ascendant sets.

3.3.4. An O(n log$_2$n) algorithm for optimally adjusting heads on operations of I

The notations are the same as the one used in previous sections. Moreover, in this algorithm, we denote by:

t: The current time instant.
$\mathscr{U}$: The set of non-completed operations at the current time instant t.
$\mathscr{D}$: The set of delayed operations at the current time instant; $\mathscr{D} = \{j \in \mathscr{U} | s_j \leq \nu\}$.
$\mathscr{A}$: The set of available and non-delayed operations at the current time instant; $\mathscr{A} = \{j \in \mathscr{U} \mid r_j \leq t$ and $j \notin \mathscr{D}\}$

Procedure Adjust(I)
Initializations
Initialize(σ,ξ)
$t = min_{j \in I}\, r_j$
$\mathscr{U} = I$; $\mathscr{A} = \{j \in \mathscr{U} | r_j = t\}$; $\mathscr{S} = I$; $\mathscr{D} = \emptyset$
For every operation $j \in I$, $p_j^+ = p_j$

Main step

While $\mathscr{U} \neq \emptyset$ *do*
  For every operation c *in* $\mathscr{A}$ *satisfying* $r_c = t$ *do*
    Find(s_c)
    If $s_c \neq +\infty$ *then* $\mathscr{A} = \mathscr{A} \setminus \{c\}$; $\mathscr{D} = \mathscr{D} \cup \{c\}$
  Enddo
  let i *be the operation in* $\mathscr{A}$ *with maximal tail*
  $t' = \min_{j \in \mathscr{U} \setminus \mathscr{A}} r_j$; $\varepsilon = \min(p_i^+, t' - t)$; $t = t + \varepsilon$; $p_i^+ = p_i^+ - \varepsilon$
  Update(σ, ξ, i, $-\varepsilon$)
  If $p_i^+ = 0$ *then* $C_i = t$; $\mathscr{U} = \mathscr{U} \setminus \{i\}$; $\mathscr{S} = \mathscr{S} \setminus \{i\}$; $\mathscr{A} = \mathscr{A} \setminus \{i\}$
  For every operation j *in* $\mathscr{D}$ *satisfying* $s_j > \nu$ *do*
    $\alpha_j = t$; $\mathscr{D} = \mathscr{D} \setminus \{j\}$; $\mathscr{A} = \mathscr{A} \cup \{j\}$
  Enddo
  $\mathscr{A} = \mathscr{A} \cup \{j \in \mathscr{U} | r_j = t\}$
  If $\mathscr{A} = \emptyset$ then $t = \min_{j \in \mathscr{U} \setminus \mathscr{D}} r_j$; $\mathscr{A} = \{j \in \mathscr{U} | r_j = t\}$
Enddo

Proposition 5. a) *Procedure* Adjust(I) *computes optimal heads adjustments on operations of* I *in* $O(n \log_2 n)$ *steps.*
b) *Moreover, the determination of these adjustments leads to the immediate selection of all the disjunctive constraints associated with the ascendant sets* J *of any operation* c.

Proof. a): We use heap structures for determining, at each iteration of the main step, the operation i from $\mathscr{A}$ with maximal tail, every operation j from $\mathscr{A}$ satisfying $r_j = t$, ν, and every operation j in $\mathscr{D}$ satisfying $s_j > \nu$. All these computations can be performed in $O(\log_2 n)$. Update $(\sigma, \xi, j, \varepsilon)$ runs in $O(\log_2 n)$ steps and the adjustment of one head does not lead to modification of p_j^+-values. Moreover, the number of preemptions in (PJPS) does not exceed n-1 (see for instance Carlier, 1982). So the total number of iterations for the main step does not exceed $2n$-1. Thus, the overall complexity is $O(n \log_2 n)$ steps.
b): See Carlier and Pinson (1990). □

3.4. Other simple immediate selections

It is straightforward to see that other immediate selections than the two particular cases discussed above can be found using the same idea: to position some operations in relation to some other ones by relaxing locally the non preemption condition. To illustrate this purpose, we propose in this section a direct extension of the last particular case developed in the previous section.

Proposition 6. *Let* i *be the operation of* $J \subseteq I$ *defined by* $q_i = \mathrm{Min}_{j \in J} q_j$, *and assume that* $p_i^- > 0$. *If* $IS(c,J) + p_i^- > UB$, *then* i *is processed before* c *in any solution, where* $IS(c,J)$ *is the quantity defined by*

$$\mathrm{IS}(c,J) = r_c + p_c + \sum_{j \in J} p_j^+ + \mathrm{Min}_{j \in J} q_j. \tag{9}$$

Proof. $\mathrm{IS}(c,J) + p_i^-$ corresponds to a makespan lower bound of a schedule in which c is processed before i. □

We actually work on a general framework exploiting these immediate selections which are of course more powerful than the previous ones.

4. Global operations

4.1. Introduction

In this section, we propose a new way of providing adjustments of heads and tails, and immediate selections of disjunctive constraints for the job shop problem. This method uses the local operations defined in previous section as a building block. In addition, we present a new lower bound based on the same principle.

4.2. Adjustment of heads and tails: Global operations

Let us consider the procedure Local which consists of determining all immediate selections (algorithms Disj and Adjust in their primal and dual versions) on each machine, propagating heads and tails adjustments over the current conjunctive graph. Let c be a given operation, and let us increase progressively the value r_c, reapplying each time the procedure Local. Obviously, such a process terminates with one of the two following issues:

a) The maximal possible value UB-$(p_c + q_c)$ for r_c has been reached without detecting any inconsistency for the global problem.
b) For a certain value of r_c, say f_c, we obtain an inconsistency for the global problem ((PJPS) – or (DJPS) – has a makespan greater than UB for some machine(s)).

Clearly, in case b), there exists no solution with c scheduled after $t = f_c - 1$. So we can set $q_c := \text{UB} + 1 - (f_c + p_c)$ to enforce operation c to start before this date. Of course, the symmetric result allowing to adjust r_c holds. In the computational experiments we present in Section 5, these new heads and tails adjustments are quite naively implemented. Finding f_c is achieved by a simple dichotomic search on the interval $[r_c;\text{UB} - (p_c + q_c)]$, and these tests are performed once per operation, in the order of their decreasing processing times. Of course, such adjustments are quite time consuming if systematically applied to all operations. But they lead to very good results, allowing, for example, to limit the search tree of our new enumerative method to 35 nodes for optimally solving the 10 jobs–10 machines instance of Muth & Thompson, with a CPU time of 520″ (against 3430 nodes without them, for a CPU time of 200″). We report in next section some complementary experiments proving their interest.

4.3. Immediate selection of disjunctive constraints

Let $[i, j]$ be a non-selected disjunction. Impose that i is scheduled before j, and apply the procedure Local presented in the previous section (determination of all immediate selections on each machine with propagation of heads and tails adjustments over the current conjunctive graph). If we obtain an inconsistency for the global problem, we can deduce that i will be processed after j in any solution. So, we can select the disjunction $[i, j]$ in the $j \rightarrow i$ direction. Of course, the global operation consisting of testing every non selected disjunction is quite time consuming, but it allows in practice to select directly a large number of supplementary disjunctions.

4.4. Lower bound

The new lower bound we propose here is based on an idea we presented in Carlier and Pinson (1990). Let us fix the quantity UB, and apply the procedure described in section 4.2. Obviously, such a procedure terminates with one of the two following issues:

a) We can detect no more immediate selections.
b) (PJPS) – or (DJPS) – has a makespan greater than UB on some machine(s).

In case a), we can decrease UB, and reapply the procedure. Necessarily, for a certain value UB′ of UB, this procedure terminates with the second issue, and UB′ + 1 is a lower bound of the optimal makespan for the job-shop problem. At each step, the new value of UB can be computed in such a way that we are sure to detect at least one immediate selection at the next procedure run. I_k denoting a subset of operations associated with machine $\mathcal{M}_k$, we define the three following quantities:

$\delta_k^1 = \max\{d_{ij} | i, j \in I_k, d_{ij} \leq \text{UB}\}$, where $d_{ij} = \max[r_i + p_i + p_j + q_j, r_j + p_j + p_i + q_i]$.

$\delta_k^2 = \max\{r_i + p_i + \xi_v(i) | i \in I_k\}$, where $\xi_v(i)$ denotes the value of ξ_v at time instant $t = r_i$ in Adjust(I) (see Section 3.3.4).

$\delta_k^3 = \max\{q_i + p_i + \xi'_v(i) | i \in I_k\}$, where $\xi'_v(i)$ denotes the value of ξ'_v at time instant $t = q_i$ in the dual version of Adjust(I).

From Section 3, it is straightforward to check that by taking the value

$$\text{UB} = \max_{k \in]m]} (\delta_k^1, \delta_k^2, \delta_k^3) - 1,$$

we are certain to detect at least one new immediate selection.

In Section 5.4, we report, for each test problem and at the root of the search tree, the lower bound computed by using this principle. The results show the superiority of this new lower bound over the classical ones proposed so far.

5. Branch and bound method

5.1. Introduction

In this section, we present a new branch and bound method using the ideas developed in the previous sections. This enumerative method uses both a new lower bound and a new branching scheme we detail hereafter. Computational results are reported.

5.2. Lower bound

The lower bound is the same as the one used in our previous branch and bound method (Carlier and Pinson, 1990).

5.3. Branching scheme

Different branching schemes for the job-shop problem have been proposed in the literature. The one we propose here relies on the idea developed in the previous section for the lower bound computation. I_k having the same meaning than before, let us consider a subset $J \subset I_k$ and an operation $c \in I_k \setminus J$ such that

$$\text{Max}\{h(c \rightarrow J), h(c \downarrow J), h(J \rightarrow c)\} > \text{UB}, \tag{10}$$

$$\text{Max}\{\text{Min}\{h(c \rightarrow J), h(c \downarrow J)\}, \text{Min}\{h(c \downarrow J), h(J \rightarrow c)\}\} \leq \text{UB}, \tag{11}$$

where

$$h(c \rightarrow J) = r_c + p_c + \sum_{j \in J} p_j + \min_{j \in J} q_j,$$

$$h(c \downarrow J) = \min_{j \in J} r_j + \sum_{j \in J} p_j + \min_{j \in J} q_j + p_c,$$

$$h(J \rightarrow c) = \min_{j \in J} r_j + \sum_{j \in J} p_j + p_c + q_c.$$

Table 1

Test origin	Test characteristics		B&B1		B&B2 (local)		B&B2 (local + global)		Optimal makespan	Lower bound
	# jobs	# machines	# nodes	CPU (s)	# nodes	CPU (s)	# nodes	CPU (s)		
(Muth & Thompson '63)	10	10	4336	253	3122	135	37	450	930	868
(Muth & Thompson '63)	20	5	44	15	49	14	40	1000	1165	1165
(Adams et al. '88)	10	10	197	12	45	9	2	55	945	901
(Adams et al. '88)	10	10	111	11	53	10	12	100	784	777
(Adams et al. '88)	15	10	5882	594	4912	475	73	4732	927	913
(Adams et al. '88)	15	10	70	13	75	12	14	1081	1032	1032
(Adams et al. '88)	20	10	125	174	102	135	22	1100	1355	1355
(Adams et al. '88)	30	10	143	207	161	218	2	480	1850	1850
(Adams et al. '88)	15	15	4416	7163	3910	6201	23	3583	1268	1233

So, J is not an ascendant set of c, but one of the three conditions leading to an immediate selection holds. Let us suppose for example that (J, c) verifies

$$h(c \rightarrow J) > \mathrm{UB}, \qquad h(c \downarrow J) \leq \mathrm{UB}, \qquad h(J \rightarrow c) \leq \mathrm{UB}.$$

We branch by creating one subproblem (P1) where c is scheduled after at least one operation of J, and a second subproblem (P2) where c is scheduled after all the operations of J. The other cases can be derived in the same way. The choice of (J, c) is based on the fact that $\mathrm{Max}\{h(c \rightarrow J), h(c \downarrow J), h(J \rightarrow c)\}$ is a lower bound of the optimal makespan for the job-shop problem. In order to improve as quickly as possible the global lower bound, we focus on the pair (J, c) satisfying (10)–(11) with maximum second minimum over the list $\{h(c \rightarrow J), h(c \downarrow J), h(J \rightarrow c)\}$, and the maximum is taken over all the machines. If no pair (J, c) satisfies the conditions (10)–(11), then we simply branch on a non selected disjunctive constraint using a penalty function as defined in Carlier and Pinson (1989).

5.4. Computational results

We have implemented this branch and bound method in Fortran 77 on a workstation IBM RS6000/320H, and tested it on about 30 benchmarks originating in the literature (Muth and Thompson, 1963; Adams et al., 1988). In Table 1, we compare our previous method (Carlier and Pinson, 1990) indicated by (B&B1) with the new one (indicated by B&B2) for which we distinguish two different implementations:

- the first one including only local operations;
- the second one including both local and global operations.

Global operations are implemented in a quite naive way. They are performed, at each level of the search tree, once per operation taken in the order of their decreasing processing times.

Table 2

Test origin	Test characteristics		B&B2 (local + global)		Best makespan	Lower bound
	# jobs	# machines	# nodes	CPU (s)		
(Adams et al. '88)	15	10	1335	71231	1048 [a]	1033
(Adams et al. '88)	20	10	207	25307	1235 [a]	1235
(Adams et al. '88)	20	10	414	62312	1216 [a]	1216
(Adams et al. '88)	20	10	96	11238	1175	1119
(Adams et al. '88)	15	15	32	3903	1397 [a]	1397

[a] Makespan proved to be optimal.

In nearly all problems we tested, B&B2 with only local operations leads to a significant reduction of the search tree size and computational time ($\cong 15\%$ in average). The addition of global operations in B&B2 leads also to important reduction of the search tree size, but in general with an increase of the corresponding computational time.

For validating the contribution of global operations, we made complementary tests summarized in Table 2. For these latter, we focused on seven instances, proposed in Adams et al., (1988) that remained unsolved up today. We were able to solve optimally four of them with B&B2 including both local and global operations. For the last 20 jobs–10 machines instance, we give only the best solution we found within a limit of 20 hours of CPU time. For the two last instances, namely problems with 15 jobs and 15 machines, we were unable to improve the best known solution within a limit of 20 hours of CPU time.

6. Conclusion

The new tools presented in this paper seem to be very powerfull in view of the results we obtain by integrating them in our branch and bound procedures. They permit for all tested problems to reduce significantly the size of the search tree associated with our new enumerative method and to solve open test problems. Nevertheless, they are still too time consuming and need to be refined in order to become really operational. A first way of achieving this goal is to use some heuristic rules in order to apply global operations only under certain conditions. Parallelly, we are working on mathematical characterizations leading on one hand to algorithmic improvements for the global operations proposed in this paper, and on the other hand to define a general framework for immediate selections in order to exploiting new ones.

References

Adams, J., Balas, E., and Zawack, D. (1988), "The Shifting Bottleneck Procedure for job-shop scheduling", *Management Science* 34, pp. 391–401.

Applegate, D., and Cook, W. (1990), "A computational study of job-shop scheduling", CMU-CS-90-145, to appear in *ORSA Journal of Computing*.

Barker, J.R., and McMahon, G.B. (1985), "Scheduling the general job shop", *Management Science* 315.

Balas, E. (1969), "Machine sequencing via disjunctive graphs: An implicit enumeration algorithm", *Operations Research* 17, pp. 941–957.

Bouma, R.W. (1982), "Job shop scheduling: A comparison of three enumeration schemes in a branch and bound approach", Master's Thesis, Faculty of Econometrics and Operations Research, Erasmus University Rotterdam.

Brucker, P., Jurisch, B., and Sievers, B. (1991), "A Branch and Bound algorithm for the Job-shop scheduling problem", to appear in *Discrete Applied Mathematics*.

Brucker, P., Jurisch, B., and Kramer, A. (1992), "The Job-shop and immediate selection", Technical Report, Osnabrücker Schriften zur Mathematik, F M/I, Universität Osnabrück.

Carlier, J. (1982), "One machine problem", *European Journal of Operational Research* 11, pp. 42–47.

Carlier, J. (1984), "Problèmes d'ordonnancements à contraintes de ressources: Algorithmes et complexité", Thèse d'état.

Carlier, J., and Pinson, E. (1989), "An algorithm for solving the job shop problem", *Management Science* Vol. 35 2, pp. 164–176.

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

French, S. (1982), "*Sequencing and scheduling: an introduction to the mathematics of the job shop*", Wiley, New York.

Garey, M.R. and Johnson, D.S. (1979), "*Computers and intractability*", Freeman, San Francisco, CA.

Grabowsky, J. (1982), "A new algorithm of solving the job-shop problem", *Operational Research in Progress*, pp. 57–75.

Jackson, J.R. (1955), "Scheduling a production line to minimize maximum tardiness", Research Report 43, Management Science Research Project, University of California, Los Angeles, CA.

Lageweg, B.J., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1976), "Minimizing maximum lateness on one machine: Computational experiences and some applications", *Statistica Neerlandica* 30, 25–41.

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

McMahon, G.B. and Florian, M. (1975), "On scheduling with ready times and due dates to minimize maximum lateness", *Operations Research*, Vol. 23 3, pp. 475–482.

Muth, J.F., and Thompson, G.L., (1963), "Industrial scheduling", Prentice-Hall, Englewood Cliffs, NJ.

Pinson, E. (1988), "Le problème de job-shop", Thèse de doctorat de l'Université PARIS VI.

Potts, C.N. (1980), "An adaptive branching rule for the permutation flow-shop problem", *European Journal of Operational Research* 5, pp. 19–25.