RAIRO. RECHERCHE OPÉRATIONNELLE

JACQUES CARLIER
EMMANUEL NÉRON

An exact method for solving the multi-processor flow-shop

RAIRO. Recherche opérationnelle, tome 34, n° 1 (2000), p. 1-25
<http://www.numdam.org/item?id=RO_2000__34_1_1_0>

RAIRO Operations Research
RAIRO Oper. Res. **34** (2000) 1-25

AN EXACT METHOD FOR SOLVING THE MULTI-PROCESSOR FLOW-SHOP (*)

by Jacques CARLIER ([1]) and Emmanuel NÉRON ([1])

Abstract. – *The aim of this paper is to present a new branch and bound method for solving the Multi-Processor Flow-Shop. This method is based on the relaxation of the initial problem to m-machine problems corresponding to centers. Release dates and tails are associated with operations and machines. The branching scheme consists in fixing the inputs of a critical center and the lower bounds are those of the m-machine problem. Several techniques for adjusting release dates and tails have also been introduced. As shown by our personal study, the overall method is very efficient.*

Keywords: Branch and bound, multi-processor flow-shop, m-machine problems, inputs and selection.

1. INTRODUCTION

In the Multi-Processor Flow-Shop, a set $I = \{J_1, J_2, ..., J_n\}$ of n jobs, must be sequentially processed on q centers $K_1, K_2, ..., K_q$. A job J_i can only be processed on one machine at a time, it consists of q operations $o_{i,1}, o_{i,2}, ..., o_{i,q}$. In the center, K_c, m_c identical machines are available. An operation, $o_{i,c}$, has a processing time $p_{i,c}$ and has to be processed without preemption on one machine of K_c.

In a Multi-Processor Flow-Shop, one has to find a set of starting times $t_{i,c}$ satisfying:

- the precedence constraints: for any job J_i, for $c \in [1, q-1]$: $t_{i,c+1} \geq t_{i,c} + p_{i,c}$,

(*) Received February 1998.
([1]) UMR CNRS 6599 HEUDIASYC, Centre de Recherches de Royallieu, Université de Technologie de Compiègne, 60205 Compiègne Cedex, France.

Recherche opérationnelle/Operations Research, 0399-0559/00/01/$ 7.00

– the resource constraints: for any $t \in \mathbb{R}$, $|\, o_{i,c}$ such that $t_{i,c} \leq t < t_{i,c} + p_{i,c}\,| \leq m_c$, and minimizing the makespan: $\min\{\max(t_{i,q} + p_{i,q})\}$.

The Multi-Processor Flow-Shop is NP-Hard, if there is more than one machine available in one of the centers, as it has been proved by Gupta [Gupta, 88], due to the fact that in this case it contains $\mathrm{P2}||C_{\max}$ as a sub-problem.

The problem is modeled with a graph, $G = (X, U)$. A starting dummy operation s^*, and an ending dummy operation e^* are introduced. Each operation corresponds to a node in X; each precedence constraint corresponds to an arc in U. Thus we can compute a release date and a tail for each operation $o_{i,c}$. If $l(h, l)$ denotes the longest path from h to l in the graph $G = (X, U)$, the release date of the operation $o_{i,c}$ is $r_{i,c} = l(s^*, o_{i,c})$ and its tail is $q_{i,c} = l(o_{i,c}, e^*) - p_{i,c}$.

Several branch and bound methods have been proposed for solving the Multi-Processor Flow-Shop [Brah and Hunsucker, 91] have presented a branching scheme based on the enumeration of the sequence of jobs and their assignment to a machine. This method has been improved by [Portmann *et al.*, 98; Vignier, 97], has proposed selection rules, in order to reduce the search tree [Perregaard, 95], has tested several branching schemes. New bounds have also been introduced by [Vandevelde, 94].

Our aim is to propose a new branch and bound method based on m-machine problems. We think it is a good trade-off between the time consumption and the size of the problems that can be solved. Indeed we use robust lower bounds which are easy to compute, powerful adjustment techniques, and an efficient branching scheme based on m-machine problems.

The paper is organized as follows. In Section 2 we define the notion of selection, and its application to the m-machine problem. Section 3 is devoted to a general presentation of the branch and bound method, whose modules are described in detail in Section 4. Finally Section 5 is dedicated to experimentation, and Section 6 to the conclusion.

2. THE m-MACHINE PROBLEM

Our branch and bound method is based on the relaxation of the Multi-Processor Flow-Shop to m-machine problems. Recall that a center K_c is made of m_c parallel machines on which operations must be processed. Each operation is defined by a release date, a processing time, and a tail. Thus it constitutes a m_c-machine problem [Carlier, 84]. For each m_c-machine problem we build a "selection".

2.1 Selection and schedule of the m-machine problem

In the disjunctive case ($m = 1$), the resource constraints are modeled by D, the set of disjunctive constraints. Thus a selection A, is a set of disjunctive arcs (i, j) extracted from D, such that if $(i, j) \in A$, $(j, i) \notin A$. If $(i, j) \in A$, then i has to be processed before j: $t_j \geq t_i + p_i$. In the general case ($m > 1$), a selection is less restrictive. It allows i and j to be in process at the same time, but no more than m operations can be processed simultaneously:

DEFINITION 1: *A selection A, for a m-machine problem, is an ordered list of operations $\{i_1, i_2, ..., i_{h-1}, i_h\}$ such that: if i precedes j in A then $\{t_i < t_j$, or $t_i = t_j$ and $i < j\}$. A selection A, is complete if I is totally ordered.*

A selection can be seen as an order of the starting times of operations. Thus if a complete selection A is given, a schedule can be computed using the strict algorithm below. This schedule is an earliest one for the selection A; it is therefore optimal.

The strict algorithm

$t = 0$
While (not all operations are scheduled) **Do**
    Schedule the first unscheduled operation i of the selection at the smallest date s_i not smaller than t, where it can be scheduled.
    Adjust t $(t \leftarrow s_i)$
End Do

The following properties have been proved in [Carlier, 84]:

PROPERTY 1: *If S is the strict schedule associated with $A = \{1, 2, 3, ..., n-1, n\}$, then $s_1 \leq s_2 \leq ... \leq s_n$. Moreover if T is any schedule satisfying $t_1 \leq t_2 \leq ... \leq t_n$ then $\forall i\ s_i \leq t_i$, that is S is an earliest schedule.*

COROLLARY 1: *Strict schedules are dominant.*

In the strict algorithm, one waits for the unscheduled operation having the greatest priority. To illustrate this definition let us present an example of 6 operations on two machines. The schedule is built according to the strict algorithm and the selection $S = \{3, 6, 1, 4, 2, 5\}$.

Notice that the strict schedules obtained are not necessarily active, as it has been proved by Sprecher [Sprecher, 94]. One can see on the Gantt chart of Figure 1, that operation 1 can be shifted, but if operation 1 enters the process at $t = 0$, before the operation 6, the order fixed by the selection is violated.

Operation	1	2	3	4	5	6
Release date	0	3	0	2	4	3
Processing time	3	5	3	6	2	4

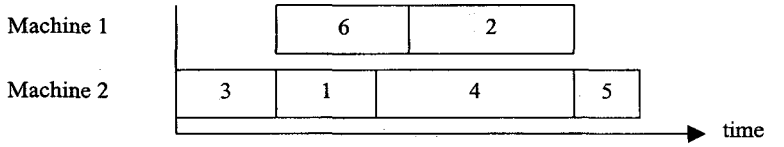

Figure 1. – The Strict Schedule deduced from a m-machine problem and a selection.

2.2 Input and output of the m-machine problem

The notions of input and output have been introduced for a standard flowshop by Carlier [Carlier, 84]. In the disjunctive case, a clique of disjunctions can be defined: $\{i, j...\}$. Thus a selection A_d is a set $\{(i,j)(j,l),....\}$ such that, if $(i,j) \in A_d$, i must be processed before j $(t_i < t_j)$. If A_d is complete, *i.e.*, all the disjunctions are selected, the first operation processed is called the input, and the last operation processed is called the output of the clique of disjunctions.

Here we define them for the m-machine problem. Inputs and outputs exist in relation to an upper bound UB. An operation is called the input for the m-machine problem if it is scheduled before all the other operations of the set, in some schedule having a makespan smaller than or equal to UB. Symmetrically, an operation is the output of a set of operations if it is completed after all other operations.

DEFINITION 2: *Let I be a set of n operations, and UB an upper bound for the overall project duration.*

$i \in I$ is a feasible input *for I and UB, if there exists*(2) *a schedule of the m-machine problem $S = \{t_i / i \in I\}$ with a makespan lower than or equal to UB and verifying: $\forall j \in I$, $j \neq i$, then $t_i \leq t_j$.*

$i \in I$ is a feasible output for I and UB, if there exists a schedule of the m-machine problem $S = \{t_i / i \in I\}$ with a makespan lower than or equal to UB and verifying: $\forall j \in I$, $j \neq i$, then $t_i + p_i \geq t_j + p_j$.

(2) Lower bounds will be used to determine if there exists a schedule with a makespan lower than UB. Indeed our lower bounds which are described below, cannot detect all the impossibilities for i to be an input of I.

If i_1 is a feasible input for I, i_2 a feasible input for $I - \{i_1\}, ..., i_k$, a feasible input for $I - \{i_1, i_2 ... i_{k-1}\}$, then $A = \{i_1, i_2, ..., i_{k-1}, i_k\}$ is a partial selection for I. If $k = n$, A is a complete selection for I. Our branching rule is based on the enumeration of the feasible successive inputs of each m-machine problem, in order to obtain complete selections and thus to be able to compute the corresponding strict schedules.

3. THE FRAMEWORK OF OUR BRANCH AND BOUND METHOD

In this section, we present the main modules of our method. These modules will be described in detail in the following sections.

3.1 Initialisation

Some initializations are performed on the Multi-Processor Flow-Shop before traversing the search tree. The first one is the computing of release dates and tails from the graph of predecessors and the processing times of operations.

Our method uses an upper bound (UB) of the makespan, to make some adjustments to the release dates and tails. Thus the method can determine if a solution with a makespan smaller than UB exists or not. A dichotomizing search on the minimal value of UB is performed. Each step of the dichotomizing search consists of one iteration of the branch and bound method described below. At the beginning, UB is initialised at the makespan of a list schedule (Earliest Due Date). UB is adjusted, whenever we find a better solution in traversing the tree, or if we prove that no solution with a makespan smaller than or equal to UB exists.

Thus the problem we have to solve can be seen as a decision variant of the Multi-Processor Flow-Shop.

3.2 An algorithm for solving the decision variant of the multi-processor flow-shop

Before detailing the different modules of our branch and bound method, we introduce them and we explain how they are connected. The algorithm below (see Fig. 2), sums up how the branch and bound method is implemented. We use the following notation in this algorithm:

– N denotes the current node of the search tree;

```
// Preliminaries //
Determine the most critical center                                    (See 4.1)
Determine if the selection is built according to inputs or outputs    (See 4.1)
If (selection based on outputs is chosen)
       Reverse the problem                                            (See 4.1)
End If

// Branch and bound for the decision variant //
While (Best_Solution > UB) and (there exists a non-explored node) Do
       Choose the node N to be explored
       current_center ← the current center of N

       // Update best solution and determine current center //
       If (the current_center is completely selected)
              If (all centers are selected) and (solution ≤ UB)
                     END_B&B        // final leaf which is a solution //
              End If
              If (there exists one unselected center)
                     current_center ← the most critical unselected center (See 4.1)
                     Determine a solution according to N                  (See 4.7)
                     If (C_max(solution) ≤ UB)
                            END_B&B
                     End If
              End If
       End If

       // Bounding rules and local enumerations //
       If (lower_bounds(N) > UB)                                      (See 4.2)
              End_Node
       End If
       Apply local enumerations (N)                                   (See 4.5 - 4.6)

       // Branching Scheme //
       Determine the list of feasible inputs for current_center       (See 4.3)
       For each feasible input e Do
              Create a node, in which e is added to the partial selection of
              current_center
              Perform Simple adjustments for this new node            (See 4.4)
       End Do
```

Figure 2. – A general algorithm for the branch and bound method.

- current_center denotes the center in which we are building the selection at the current node N;
- END_Node means that the treatments performed on the current node have to be stopped. So another node has to be explored. If there is no other node to be explored, we can conclude that there does not exist any solution with a makespan smaller than or equal to UB;

- END_B&B means that a solution with a makespan smaller than or equal to UB has been found. The value of UB can be updated and the next step of the dichotomizing search on UB can be performed.

4. THE MODULES OF OUR BRANCH AND BOUND METHOD

The method we present in this section determines whether there exists a solution for the Multi-Processor Flow-Shop such that its makespan is smaller than or equal to UB. We now detail its different modules.

4.1 Preliminaries: Critical center

A lower bound is computed for each m_c-machine problem. The m-machine problem which has the largest lower bound defines the critical center which will be selected first. All treatments (branching scheme, bounding rules, etc.) basically focus on this m-machine problem.

The branching scheme is based on the enumeration of feasible inputs of a m-machine problem. But it may be more efficient to branch according to its outputs. Depending on the critical center, we will define a "way" for building the selection. The more the release dates are scattered, the fewer the number of nodes that have to be created (see Sect. 4.3). For example, if we consider the first center, all the release dates are equal to 0. So if we choose to build the selection according to its inputs, at the root of the search tree, n inputs are feasible. On the contrary, if we choose to build the selection according to its outputs, the number of nodes to be created will be reduced.

A scattering indicator is computed from the release dates and tails of the critical center, arranged in a non decreasing order such that:

$$r_{i1} \geq r_{i2} \geq \ldots \geq r_{in} \quad \text{and} \quad q_{j1} \geq q_{j2} \geq \ldots \geq q_{jn},$$

$$D_{Er} = \sum_{p=2\ldots n} (r_{ip} - r_{ip-1})^2, \qquad D_{Eq} = \sum_{p=2\ldots n} (q_{jp} - q_{jp-1})^2.$$

If D_{Er} is greater than D_{Eq}, we decide to build our search tree on input selection of the m-machine problems. Otherwise, the problem is transposed:

$$\forall K_j \in K, \qquad \forall J_i \in J,$$

$$r_{i,j} \leftarrow q_{i,(q-i+1)}, \quad q_{i,j} \leftarrow r_{i,(q-j+1)}, \quad p_{i,j} \leftarrow p_{i,(q-j+1)}, \quad m_j \leftarrow m_{q-j+1}.$$

Practically, K_j becomes $K_{(q-j+1)}$, where q is the number of centers. The main advantage of this transposition is to have only one kind of algorithm

based on "input selection". Finally, when a solution is built, the same transposition has to be made.

4.2 Bounding rules

In the following sections, J is a subset of I, with $|J| > m$.

THEOREM 1: $G'(J) = (1/m)[r_{i1} + \ldots + r_{im} + \Sigma p_i + q_{j1} + \ldots + q_{jm}]$ *is a lower bound of the m-machine problem, where m is the number of available machines,*

$r_{i1}, \ldots, r_{im}$ *are the m earliest release dates among the operations of J,*

$q_{j1}, \ldots, q_{jm}$ *are the m smallest tails among the operations of J.*

Proof: See [Carlier, 84].

We also use $\max_{i \in J} (r_i + p_i + q_i)$ as a lower bound.

The complexity in the worst case for computing $G'(J)$ is $O(m.n_j)$, where $n_j = \text{card}(J)$.

Some lower bounds may improve the $G'(J)$ bound as $\max_{J' \subseteq J}[G'(J')]$, called the Sub-Set Bound but the corresponding improvement may not be useful in comparison with the increase of the computation time; a $O(n^2)$ algorithm was proposed by [Perregaard, 95], and [Vandevelde, 94], for computing the Sub-Set Bound. Indeed in our case the subset of operations, which realizes the maximum is often J. Another improvement of the bound based on $G'(J)$ is the Adjusted Sub-Set Bound (ASSB). This bound takes into account the fact that an operation is either processed alone on a machine, or its release date and its tail are not both active. Perregaard [Perregaard, 95], has proposed a $O(2^m m^2)$ for computing ASSB.

Many other bounds have been introduced for the m-machine problem. Perregaard [Perregaard, 95], has tested a lower bound based on the maximum flow. This bound is a graph formulation of the preemptive m-machine problem. But the worst case complexity associated with it is $O(n^4)$. The Jackson Pseudo Preemptive Schedule, proposed by [Carlier and Pinson, 98], gives also an efficient lower bound for the m-machine problem. It can be computed in $O(m.n \log n)$. [Baptiste *et al.*, 98] presented feasibility tests, that could be applied to the m-machine problem. These tests are based on energetic reasoning, and their cost is $O(n^2)$.

Our method is based on the $G'(J)$ bound, because of its low complexity.

4.2.1. $G'(J)$ applied to the previous centers

Figure 3 describes a Multi-Processor Flow-Shop. Next and previous centers of the current center are represented. I denotes the set of operations of the centers, J the set of unselected operations ($J \subseteq I$), at the current center. The predecessors and successors of the unselected operations of the current center are drawn in Figure 3.

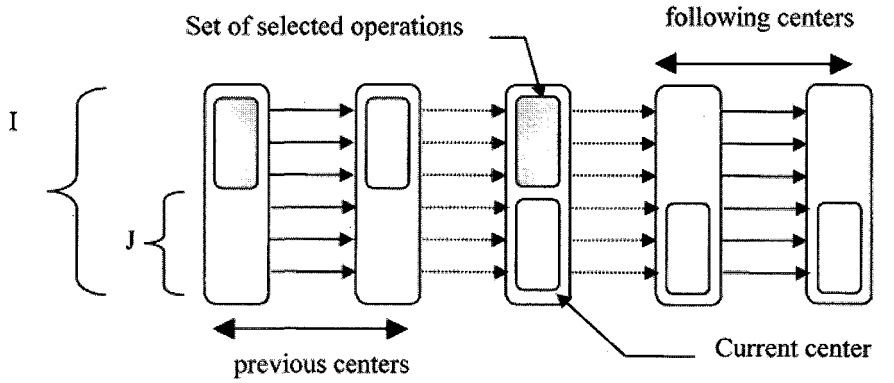

Figure 3. – $G'(J)$ applied to previous/following centers.

For each previous center, the set of operations is divided into two subsets. The set $I - J$ of operations which are the predecessors of the selected operations in the current center, and its complementary part: J. $I - J$ is the most useful subset, because of the modifications to the tails of the selected operations in the current center (see Sect. 4.4.1). These adjustments are propagated to the previous centers (see Sect. 4.4.2). The complexity is $O(mn_j)$, where $n_j = |J|$.

4.2.2. $G'(J)$ applied to the following centers

The lower bound is also computed for the following centers by taking into account the successors in each center of the unselected operations of the current center. The subset J is the most useful one because of the modifications to the release dates, and their propagation to the next centers (see Sect. 4.4.2). The complexity is: $O(m_l.n_j)$, with m_l, the number of machines available in center K_l.

If one of the lower bounds computed is greater than UB, any global selection is unfeasible. A backtrack in the search tree occurs.

4.2.3 Cutting rules associated with release dates of machines

A new cutting rule is used in our method, which is based on the machine release dates. We define for the m machines of the current center, the availability times of the machines $R_1, R_2, ...R_m$ with $R_1 \leq R_2 \leq ... \leq R_m$. It is important to notice that these machine release dates depend on a partial schedule, which is built according to a partial selection.

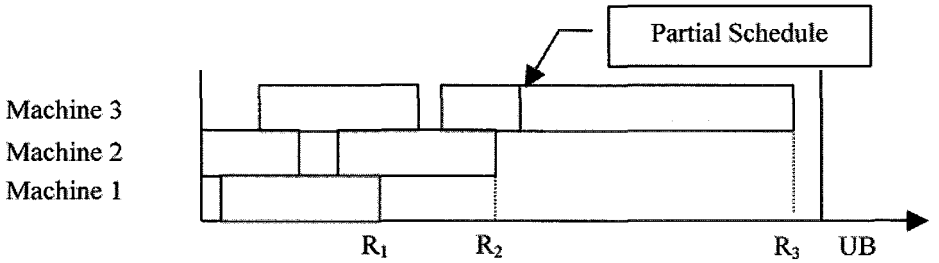

Figure 4. – Example of machine release dates.

THEOREM 2: *Let:* $G'_{\text{machine}}(J) = (1/m)\,[\max(R_1, r_{i1}) + \max(R_2, r_{i2}) + \ldots + \max(R_m, r_{im}) + \Sigma_i p_i + q_{j1} + q_{j2} + \ldots + q_{jm}]$.

If $R_m + q_{jm} < UB$ and $G'_{\text{machine}}(J) > UB$, there does not exist any schedule with a makespan smaller than or equal to UB.

Recall that:

$r_{i1}, ..., r_{im}$ are the m earliest release dates among the operations of J,

$q_{j1}, ..., q_{jm}$ are the m smallest tails among the operations of J.

Proof: Let S be a schedule with a makespan smaller than or equal to UB. The proof is divided into two steps:

PROPERTY 2: *If there exists a machine without any operation to process in S, there exists a preemptive schedule, S', with a makespan smaller than or equal to UB, in which all the machines have to process an operation or a part of an operation.*

Proof: Let us suppose that the maximal number of machines used for scheduling J in S, is strictly smaller than m:

- if the machine without any operation to process is not the machine m, let l denote this machine. R_l verifies: $R_l \leq R_m$, thus the operations scheduled on machine m, can be scheduled on machine l, the machine without any operation to process becomes the machine m (see next case).
- if the machine without any operation to process is the machine m: let us denote $\delta = UB - R_m - q_{jm}$, $\delta > 0$. A part of the operation j_m

(with a processing time equal to δ, a release date equal to r_{jm}, and a tail equal to q_{jm}), which is scheduled on some machine k, can be scheduled between $t = UB - q_{jm} - \delta$ and $t = UB - q_{jm}$ on machine m, without modifying the makespan. In this solution S' (deduced from S, by relaxing the constraint of non-preemption) each machine has an operation or a part of an operation to process.

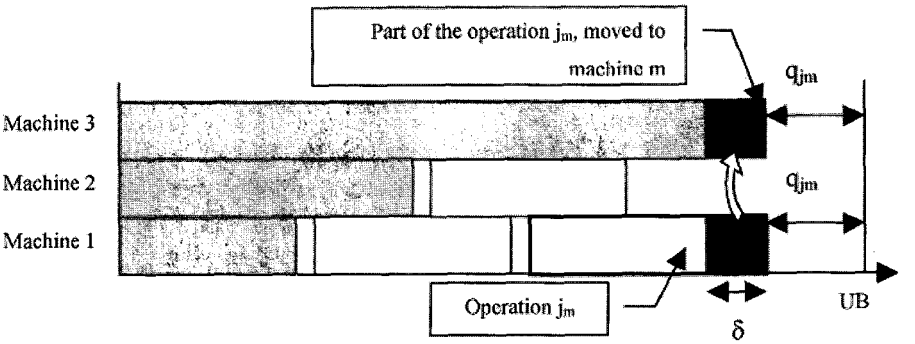

Figure 5. – Illustration of Property 2.

PROPERTY 3: *If* $R_m + q_{jm} < UB$, $G'_{\text{machine}}(J)$ *is a lower bound of* UB.

Proof: We will sum the idle times and the processing times of all machines: if one machine has no operation to process the schedule is transformed, by using Property 2, into a schedule in which all the machines have an operation or a part of an operation to process. Thus we assume that all the machines have an operation or a part of an operation to process:

- m machines are used, so $m \times UB$ time units are available to process the operations;
- let us sequence the machines in increasing order $k_1, k_2, ..., k_m$ of their starting times, *i.e.*, the starting times of the first operation or part of operation that is processed on this machine;
- clearly the machine k_1 is idle from 0 to R_1 and also from 0 to r_{i1}. So it is idle from 0 to $\max(R_1, r_{i1})$;
- machine k_2 is idle from 0 to $\max(R_2, r_{i2})$, ..., and machine k_m is idle from 0 to $\max(R_m, r_{im})$;
- one machine is idle between $t = UB - q_{j1}$ and $t = UB$, another machine is idle between $t = UB - q_{jp}$ and $t = UB$...;
- the machines must process a total work equal to the sum of the processing times of the operations of J.

By adding processing times and idle times of all machines we obtain:

$$[\max(R_1, r_{i1}) + \max(R_2, r_{i2}) + \ldots + \max(R_m, r_{im}) + \Sigma p_i + q_{j1} + \ldots + q_{jm}] \leq m \times UB.$$

So $G'_{\text{machine}}(J) = (1/m)[\max(R_1, r_{i1}) + \max(R_2, r_{i2}) + \ldots + \max(R_m, r_{im}) + \Sigma p_i + q_{j1} + \ldots + q_{jm}$ is smaller than UB.

The worst case complexity for computing this lower bound is the same as for the $G'(J)$: $O(m.n)$ for the first m minimum release dates, and first m tails; $O(n)$ for the sum of the processing times. This bound is only applied to the current center since the machine release dates are only defined for the current center.

4.3 Branching Scheme based on total selection

At each node of the search tree a partial selection, PA, is built on the current center, and a schedule corresponding to this partial selection is computed. A special case occurs when the partial selection is a complete one. Then another center is chosen: it is the most critical center among the unselected centers, *i.e.*, the one which realizes the maximum of $G'(J)$. Its partial selection is empty at this moment.

If there exists some unselected operations in the current center we determine a set of possible inputs. Here J is the set of unselected operations.

First of all, we compute the new time R_1 to take into account, the earliest time a machine becomes available. At this time R_1 we determine a list of inputs for the current node. If less than m operations are unselected, we consider each operation as a feasible input for J. Otherwise ($|J| > m$) for each unselected operation $e \in J$, we compute:

$$F_e(e) = (1/m) \times [\max(R_1, r_e, r'_{i1}) + \max(R_m, r_e, r'_{im}) + \Sigma p_i + q_{j1} + \ldots + q_{jm}],$$

where: R_1: the earliest machine release date,

$r'_{i1}, \ldots, r'_{im}$: the m earliest release dates among the operations of J-$\{e\}$.

THEOREM 3: *If* $(R_m + q_{jm} < UB)$, $F_e(e)$ *is a lower bound of the makespan of any schedule in which* e *is selected as an input for* J.

Proof: If e is an input, no operation can start before the starting time t_e of operation e which verifies $t_e \geq r_e$. Thus no operation can start on machine

i before $t = \max(r_e, R_1)$. The remainder of the proof is the same as that previously used to demonstrate Theorem 1 (see Sect. 4.2.3): by summing the processing/idle time units on all the machines we deduce that $F_e(e)$, is a lower bound of the makespan of the schedule in which e is selected as an input for J.

According to Theorem 2, $F_e(e) > UB$ implies that if e is considered as an input, the corresponding schedule will have a lower bound greater than UB. Thus only if $F_e(e) \leq UB$, e is considered a feasible input for J. For each operation verifying the previous inequality, a node is created in the search tree, with a new partial selection $[PA + \{e\}]$. The complexity, for computing $F_e(e)$ is equal to $O(m.n_j)$, where n_j is the number of unselected operations.

We use a depth first search strategy: the last node created is the first node explored. The nodes are arranged in increasing order of the release dates of their input. Thus the first node explored corresponds to the input which has the earliest release date.

4.4 Simple adjustments of heads and tails

Adjustments are made to release dates (heads) and tails of the operations; local adjustments are made to the operations, followed by propagation of these adjustments to the other centers. This part of the method is extremely important. The efficiency of the lower bounds depends on the adjustments made to the release dates and tails. Moreover release dates and tails are also involved in determining the list of feasible inputs. Thus adjustments have a large impact on the quality of our branching scheme.

4.4.1 Adjustments to the current center

Immediate Adjustments deduced from the new input

The branching scheme has imposed a new input e, among the unselected set of operations J. An operation is an input if no other operation begins before it, thus:

$$\forall i \in J, \qquad r_i \leftarrow \max(r_i, r_e),$$
$$\forall k = 1, \ldots, m, \qquad R_k \leftarrow \max(R_k, r_e).$$

The simplest adjustments to the release dates of the unselected operations take into account the new availability time of the machines: an operation cannot start before a machine becomes available and R_1 is the first time a machine becomes available, thus:

$$\forall i \in J, \qquad r_i \leftarrow \max(r_i, R_1).$$

Adjustments based on the lower bound UB

An adjustment is made to the tail of the input e. If $t_{e\max}$ is the latest starting time of activity e, we have:

$$t_{e\max} = UB - (p_e + q_e),$$
$$\text{and} \quad t_{e\max} \leq UB - (1/m) \times (\Sigma_{i \in J+\{e\}} p_i + q_{j1} + \ldots + q_{jm}),$$

$(1/m) \times (\Sigma_{i \in J+\{e\}} p_i + q_{j1} + \ldots + q_{jm})$ is an evaluation of the work that has to be processed after e starts.

Then $q_e + p_e \geq (1/m) \times (\Sigma_{i \in J+\{e\}} p_i + q_{j1} + \ldots + q_{jm})$.

We can set $q_e \leftarrow \max(q_e, (1/m) \times [\Sigma_{i \in J+\{e\}} p_i + q_{j1} + q_{jm}] - p_e)$.

The complexity of this adjustment is $O(m.n_j)$, with $n_j = |J|$.

Another adjustment can be computed from the partial selection associated to a node. Let J be the set of the unselected operations, $G'(J)$ the lower bound associated with J, and UB the current upper bound. Let us denote δ, if it exists, the first integer verifying:

$$(1/m) \times [r_{i1} + \delta + r_{i2} + \ldots + r_{im-1} + r_{im} + \Sigma p_i + q_{j1} + \ldots + q_{jm}] > UB,$$

and $r_{im+1} - r_{i1} \geq \delta \geq 0$.

δ is the smallest integer that can be added to the release date of i_1, in order to verify $G'(J) > UB$. We can use $r_{i1} + \delta$ as the latest starting time of operation i_1. Thus we can deduce:

$t_{i1} < r_{i1} + \delta$, (otherwise the new value of $G'(J)$ will be strictly greater than UB),

$t_{i1} + p_{i1} < r_{i1} + \delta + p_{i1}$,

$C_{i1} < r_{i1} + \delta + p_{i1}$ (where C_i is the completion time of operation i),

$UB - C_{i1} > UB - (r_{i1} + \delta + p_{i1})$.

As we are interested in schedules having a makespan equal to UB, we can assume that $C_{i1} + q_{i1} = UB$. So we can set: $q_{i1} \leftarrow \max(q_{i1}, UB - (r_{i1} + \delta + p_{i1}) + 1)$.

δ exists if the evaluation is close to the upper bound. Thus, this adjustment is efficient only if the selection is near to be complete, or if the center is critical. If δ exists and if the new q_{i1} is greater than the previous one, the same deduction can be applied to q_{i2}. The complexity of these adjustments is the same as the complexity of the computation of $G'(J)$.

This kind of adjustment is also available for the release dates. We determine δ' as previously. Its correct value, and new adjustments to the r_{ij}, can be computed with the new value of $G'(J)$ corresponding to the adjusted tails. We can set r_{j1} equal to $\max(r_{j1}, UB - (q_{j1} + \delta' + p_{j1}) + 1)$.

4.4.2 Propagation of adjustments to other centers

The modification of the release dates of the operations of the current center K_c, are propagated to the following centers and the new tails are propagated to the previous centers.

$$\forall j > c, \qquad r_{i,j} = \max(r_{i,j}, r_{i,j-1} + p_{i,j-1}),$$
$$\forall k < c, \qquad q_{i,k} = \max(q_{i,k}, q_{i,k+1} + p_{i,k+1}).$$

Some modifications to the release dates of operations of a selected center may involve modification of the schedule associated with the selection. Thus, an earliest schedule is built using the strict algorithm, according to the selection and the modified release dates (see Sect. 2.1.1). The makespan of this new schedule must not be greater than the upper bound (UB). The release dates of the re-scheduled center are also propagated to its following centers.

The modification of the release dates and tails based on $G'(J)$ may also be computed for the operations of other centers (see Sect. 4.4.1). The adjusted values are then also propagated.

4.5 Local enumeration on a restricted m-machine problem

We now consider the m-machine problem, deduced from the set of the unselected operations of the current center. The method we propose produces adjustments to release dates and tails by enumerating all the schedules having a makespan smaller than UB for this restricted m-machine problem:

$$r_i \leftarrow \max(r_i, \min_{S \in SR}(t_i(S))),$$
$$q_i \leftarrow \max(q_i, \min_{S \in SR}(q_i(S))),$$

where $t_i(S)$ is the starting time of the operation i in the schedule S,

$q_i(S)$ is the computed tail of the operation i in the schedule S,

SR is the set of feasible schedules having a makespan smaller than or equal to UB.

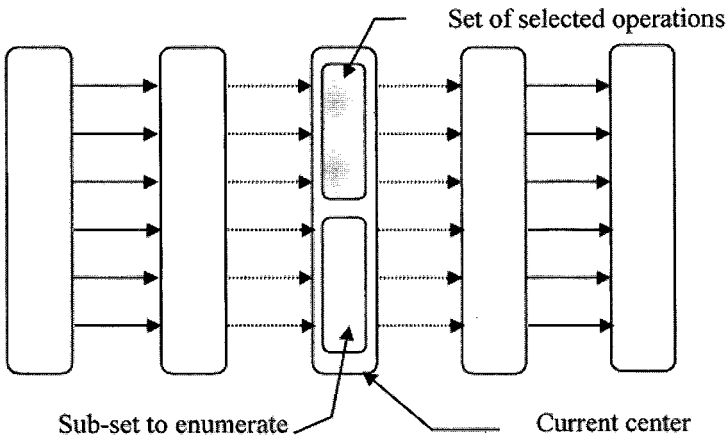

Figure 6. – Local enumeration on a m-machine problem.

The branch and bound method used to enumerate all the solutions must be efficient and simple, otherwise the time spent in exploring the search tree may be too high in relation to the deduced adjustments. Main modules of this branch and bound method are:

- *branching scheme*: at each node of this local enumeration, each unselected operation is considered as a feasible input if it satisfies $\max(t, r_i) + p_i + q_i \leq UB$. A selection is built when all operations are selected. A node is created in the local search tree for each feasible input. The number of nodes created is balanced by the speed of the treatments applied;
- *scheduling rules*: input e is scheduled at $t_e = \max(t, R_1, r_e)$, with R_1 the minimal machine release date, and r_e the release date of the operation e. $t \leftarrow t_e$, if necessary;
- *adjustments to the release dates*: each time a solution is found the minimum release dates are updated along the starting times of the operations, as computed in this solution;
- *adjustments to the tails*: each time a solution is found, a tail is computed for each successive input of the solution: $q_e \leftarrow \max\left(q_e, (1/m)\left[\sum_{i \in J+\{e\}} p_1 + q_{j1} + \ldots + q_{jm}\right]\right) + p_e$ where Σp_i is the sum of the processing times of the operations selected after e, $q_{j1}, \ldots, q_{jm}$ are the first m tails among the operations selected after e;
  Thus the minimal tails are updated with the value of the tails computed.

- *lower bound*: it is computed at each node in order to reduce the local search tree.

$$LB = (1/m)\,[\max\,(t, R_1) + \ldots + \max\,(t, R_m) + \Sigma p_i + q_{j1} + \ldots + q_{jm}]$$

where Σp_i is the sum of the processing times of the unscheduled operations,
R_k is the release date of the machine k,
$q_{j1}, \ldots, q_{jm}$ are the m minimal tails among the set of unscheduled operations.
Notice that the list of the m minimal tails can be computed incrementally at each node of the local search tree, in order to reduce the computation time;

- the special case of a *final leaf* (when a solution is built): the adjustments to release dates and tails are made. A list of the first inputs is also updated. This list contains the operations, which are the first inputs in any one of the solutions that have been built. When all the solutions having a makespan smaller than or equal to UB are explored, this list contains all the feasible inputs for the set of unselected operations. If an operation is a feasible input for the global m-machine problem ($F_e(e) \leq UB$) but does not belong to this list, the corresponding node in the global search tree will not be created.

This local enumeration can also be done to enumerate the feasible selections based on outputs for the m-machine problem. The adjustments are: $r_i \leftarrow \max\,(r_i, \min_{S \in SR}(r_i(S)))$, $q_i \leftarrow \max\,(q_i, \min_{S \in SR}(t_i(S) - p_i))$.

The local enumeration is based on the enumeration of the latest schedules having a makespan smaller than or equal to UB. These enumerations on restricted m-machine problems are performed at each node of the global tree, under some conditions on the three following parameters:

- *the number of operations to enumerate*, which is equal to the size of the restricted m-machine problem. If there are fewer operations to enumerate than the number of machines in the center, no adjustments will be made. But of course, the method would be more efficient if we considered a larger problem but it would used more computational time. Tests performed indicate that 5 or 6 operations is the optimal size of restricted m-machine problems required to obtain efficient enumerations;
- *the distance to the optimum to launch this method*: far from the optimum ($UB \gg LB$), deductions performed are rare; every partial selection

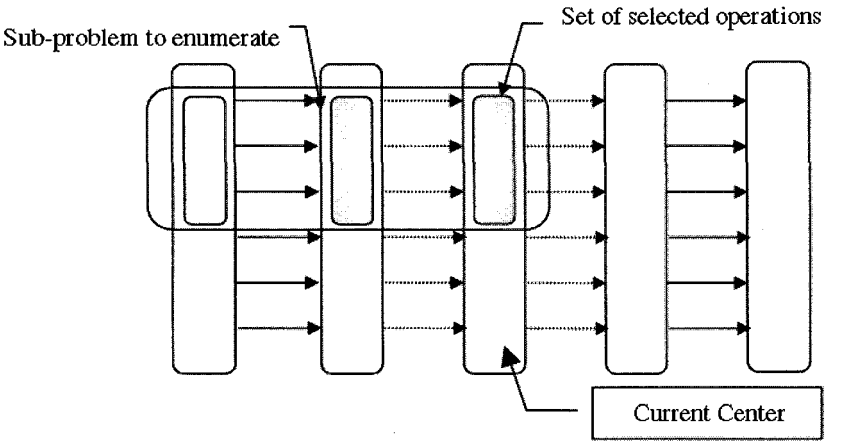

Figure 7. – Local enumeration on a restricted multi-processor flow-shop.

enumerated will be feasible. Thus the enumeration on a restricted m-machine problem is performed under the previous condition, and if $(UB - LB)/LB \leq D_{\text{call}}$. Experiments indicate 5% to be the best average distance providing a good trade-off between the time spent in this procedure and the adjustments deduced;

– *the way we use both enumeration methods* based on inputs and outputs. It seems that they are complementary, and the method is more efficient if both are used.

On some examples there exist better parameters, but these values give the best average results based on our benchmarks (see 5).

4.6 Local enumeration on a restricted multi-processor flow-shop

This local enumeration is applied during the construction of the selection of the most critical center, which is the first center selected. Since more than m operations are selected, a sub-problem can be isolated. This sub-problem is also a Multi-Processor Flow-Shop problem, the centers are the previous centers of the critical center, and the operations in each center are the ascendants of the selected operations in the critical center. This restricted Multi-Processor Flow-Shop is strongly constrained because of the selection in the critical center; according to this selection tails have been computed for selected operations, which have been propagated to the previous centers (see Sect. 4.4.2).

We propose a branch and bound method to solve this restricted Multi-Processor Flow-Shop:

- *schedules are built chronologically*: an operation e is eligible, if its predecessor, is scheduled and if it satisfies $r_e + p_e + q_e \leq UB$. This operation is scheduled at $t_e = \max(r_e, R_1, t)$ (R_1 is the first time a machine is available). Then t is increased to t_e.
  Inputs of the critical center are scheduled according to the selection already computed for this center; an operation of the critical center can be scheduled only if all the operations before it in the selection have already been scheduled. A node is created for each operation that can be scheduled.
  Each time a final leaf is reached the list of the minimal release dates is updated with the value of the starting times of the operations;
- *adjustments on release dates*: if e is scheduled at time t_e, no other operation can start before t_e. Then the release dates of the unscheduled operations are adjusted to t_e. These adjustments are propagated to the next centers. If the release date of an operation of the critical center is modified, a schedule is computed in the critical center according to the selection and the adjusted release date. A backtrack occurs if the makespan is greater than UB;
- *adjustments to tails*: as it is reported in Section 4.5, adjustments of tails are computed when a final leaf of the local search tree is reached: $q_e \leftarrow \max\left(q_e, (1/m) \times \left[\sum_{i \in J_{1e}} p_1 + q_{j1} + \ldots + q_{jm}\right] - p_e\right)$. Then the minimum tails are updated with the values of the tails computed;
- *the left-shift rule*: if an operation e, scheduled at time t_e, can be left shifted without violating either the precedence constraints (its predecessor p has started before $t_e - p_p$), or the resource constraints ($R_1 \leq t_e$), then the corresponding schedule will be dominated, as it has been proved by Demeulemeester and Herroelen [Demeulemeester, 92];
- *lower bound*: each node corresponds with an operation e, which is the latest operation scheduled. The lower bound is computed on the center to which e belongs. This lower bound is computed on the set of unselected operations of this center. It is important to note that the whole center is considered (not only the operations of the center of the restricted problem). The release dates and tails of the operations which belong to the restricted problem are those which have been determined at the node of the local tree. Due to the number of machines available

in the initial problem, the resource constraints will be loose if only the operations of the restricted problem are considered.

If we use the same notation as in Section 4.5, the adjustments made to the release dates and tails are:

$$r_i \leftarrow \max(r_i, \min_{S \in Sr}(t_i(S))), \qquad q_i \leftarrow \max(q_i, \min_{S \in Sr}(q_i(S))).$$

The present method involves explicit enumeration of the solutions for the restricted problem. But this explicit enumeration may cost a lot of computational time. The method we use is based on a truncated exploration of the search tree: the restricted problem is not solved explicitly, but we stop the traversing of the search tree since a given number of operations are scheduled. We consider that a final leaf in the tree is reached since $Nb_{\text{operation}}$ operations are scheduled, where $Nb_{\text{operation}}$ is a fixed number.

Experiments have been performed with several parameters. Efficient use of this method requires consideration of the same 3 parameters as those used in the previous section (see Sect. 3.5):

- *the number of operations to enumerate*: 10 operations;
- *the size of the restricted flowshop*: 4 jobs;
- *the distance to the optimum to launch this method*: 5%.

These three parameters may improve the efficiency of the method by decreasing the time search. The values above are the ones which give the best results based on our benchmarks. Of course these values are a trade-off between the time spent per node and the number of nodes explored.

4.7 Construction of a schedule during the exploration

Solutions are generated during exploration of the search tree. Schedules are built each time a selection is completed in a center, before the choice of the next current center. If the newly built schedule has a makespan smaller than or equal to UB, a solution is found. The algorithm used to build it is the strict algorithm based either on the selection, if the center is completely selected, or on the maximum tail priority rule.

Partial solution algorithm:

$c = 1$
While $(c \leq q)$ **do**
  If (center c is selected)
    $L =$ selection of the center c
  Else
    $L = (i_1, i_2, ..., i_n)$ with $q_{i1} \geq q_{i2} \geq ... \geq q_{in}$

End If
Build the strict schedule according to L
For each operation $o_{i,c}$ **Do**
Propagate the starting time of $o_{i,c}$ to the next center:
$r_{i,c+1} = r_{i,c} + p_{i,c}$
End Do
$c = c + 1$
End Do

This mechanism is useful because of the dichotomizing search: some iterations correspond to an upper bound far from the optimum, in which case a solution can be found quickly, even if all centers are not selected.

5. EXPERIMENTATION

The branch and bound method has been tested on some benchmarks. Our method is efficient on many kinds of problems involving 10 or 15 jobs and 5 or 10 centers. For a given number of jobs and a given number of centers there exist several machine configurations. A machine configuration is a list of digits representing the number of machines available in each center. An example of a test is (10 × 5 [33331]) where there are 10 jobs, 5 centers and 3 machines available in each center, except in the last center where only one machine is available.

The computer used is a SPARC ENTERPRISE 40000 (SOLARIS). The method was coded in C language.

If no solution has been found after 1300 seconds, the search is stopped. Each set consists of 6 instances with the same number of jobs, the same number of centers and the same machine configuration. For all the problems that we have generated, processing times were randomly chosen.

The number of operations in each center is one of the main parameters; with the same number of operations, the problem which has fewer jobs and more centers will be easier to solve. For example a 10 center and 5 job problem will be easier than a 10 job and 5 center problem.

Since the centers are equilibrated (no center is a bottleneck center), the problems become harder even if there are only 10 jobs and 5 centers. For smaller problems, computational results are not reported, however the method finds the optimal solution even if there is the same number of available machines in all centers.

Tables 1 and 2 present a comparison with the results obtained by [Vignier, 97] on benchmarks proposed by [Portmann *et al.*, 98]. This comparison is interesting because of the difference in the branching scheme used. The

method proposed by Vignier is an extension of the branch and bound method proposed by [Brah and Hunsucker, 91]. The branching scheme used is based on the construction of active schedules. Machines are taken into account independently.

It is important to notice that Vignier's results were obtained on a PC486DX33. It is difficult to compare the computation time search because of the difference between the two computers used, but the low number of nodes generated by our method in solving these instances, provides information about its efficiency. It therefore seems that our method is more efficient, on this kind of instances.

TABLE 1
Comparison with Vignier's method [configuration 13323].

	Vignier *et al.*	Carlier and Néron		
Configuration: 13323	% Solve	% Solve	Av. Nodes	Av. TIme
5 Jobs 5 centers	100%	100%	32	0.01 s
10 Jobs 5 centers	33%	100%	4649	15.56 s
15 Jobs 5 centers	33%	100%	110	0.15 s

TABLE 2
Comparison with Vignier's method [configuration 33231].

	Vignier *et al.*	Carlier and Néron		
Configuration: 33231	% Solve	% Solve	Av. Nodes	Av. Time
5 Jobs 5 centers	0%	100%	30	0.01 s
10 Jobs 5 centers	0%	100%	11	0.01 s
15 Jobs 5 centers	0%	83%	23	0.05 s

These tests do not prove the total efficiency of the method:

- most of the problems are very easy to solve (less than 0.1 s to get the optimum);
- those which are not solved immediately seem to be extremely difficult.

So tests have to be performed on randomly generated problems:

- average time search, computed on each set of problems is reported. It takes into account only the time search for the problems solved optimally.

For example in Table 4 (line 2), the average time increases if the local enumeration methods are used, but the method is more efficient because of the larger number of problems solved optimally;

- if the problem is not solved optimally, the distance to the optimum gives information on the efficiency of the method. The column "distance" indicates, for the unresolved problems an estimation of the distance to the optimum: $(UB_d - LB_d)/LB_d$, where LB_d and UB_d are the best upper bound and the best lower bound determined by the dichotomizing search;
- this information is reported for a branch and bound method without local enumeration (basic method) and for a branch and bound method which uses the local enumeration methods with the parameters described in Sections 4.5 and 4.6.

Tables 3 and 4 show that for this kind of "not too large problems" the method is very efficient, since there is a bottleneck center (less than 1s to solve them optimally). Additionally the local enumeration methods are useful in reducing the computation time even if the number of problems which are optimally solved stays equal for the 15 jobs and 5 centers instances.

TABLE 3
10 job and 5 center problems with a bottleneck center.

[Jobs-Centers]	Basic Method			Local Enumerations		
[10-5]	% Solved	Av. Time	Distance	% Solved	Av. Time	Distance
33133	100%	0.3 s	–	100%	0.14 s	–
13333	100%	2.07 s	–	100%	1.05 s	–
33233	86%	332.1 s	5.4%	100%	232.3 s	–
33333	33%	266.1 s	10.1%	50%	731.2 s	7.2%

TABLE 4
15 job and 5 center problems with a bottleneck center.

[Jobs-Centers]	Basic Method			Local Enumerations		
[15-5]	% Solved	Av. Time	Distance	% Solved	Av. Time	Distance
33133	100%	0.1 s	–	100%	0.2 s	–
13333	100%	0.7 s	–	100%	0.2 s	–
33233	17%	468.3 s	7.5%	17%	260.2 s	7.5%

TABLE 5
10 job and 10 center problems.

[Jobs-Centers]	Basic Method			Local Enumerations		
[15-10]	% Solved	Av. Time	Distance	% Solved	Av. Time	Distance
3333133333	100%	0.4 s	–	100%	1.3 s	–
1333333333	100%	0.7 s	–	100%	0.5 s	–
3333233333	0%	–	11.4%	0%	–	10.4%

TABLE 6
15 job and 10 center problems.

[Jobs-Centers]	Basic Method			Local Enumerations		
[10-10]	% Solved	Av. Time	Distance	% Solved	Av. Time	Distance
3333133333	86%	1.0 s	1.0%	83%	0.8 s	1.0%
1333333333	66%	0.1 s	3.2%	83%	9.6	3.4%

On larger problems it becomes hard to reach the optimum, even if some 10 job and 10 center instances which have a bottleneck center are optimally solved.

6. CONCLUSION

We have proposed a new method to solve exactly the Multi-Processor Flow-Shop. This method uses some properties of the m-machine problem. The branching scheme generalizes the notion of selection to the cumulative case. Its main advantage is the production of adjustments to heads and tails of operations of the current center, or of operations of the other centers. We have reported our first computational results, which indicate the method seems efficient. For instance we can solve larger problems if a dominant center exists. Experimentation has also been done to prove the efficiency of the method on equilibrated problems, which are extremely difficult to solve.

Many improvements could be made, especially in regard to the lower bounds. At present we are working on improving the method to obtain more efficient lower bounds and adjustments, by using the works of Baptiste *et al.* [Baptiste, 98], and Carlier and Pinson [Carlier and Pinson, 98] on the m-machine problems.

ACKNOWLEDGEMENTS

The authors would like to thank Philippe Baptiste for many enlightening discussions on Multi-Processor Flow-Shop. We would also like to thank Antony Vignier for his complete benchmarks, results and helpful remarks. Finally we are grateful to the referees for their constructive comments.

REFERENCES

[Baptiste *et al.*, 98] P. Baptiste, C. Le Pape and W. Nuijten, *Satisfiability tests and time-bound adjustments for cumulative scheduling problems*, Technical Report 98-97, Université de Technologie de Compiègne, to appear in Ann. Oper. Res.

[Brah and Hunsucker, 91] S. A. Brah and J. L. Hunsucker, *Branch and bound method for the flow shop with multiple processors*. European J. Oper. Res., 1991, *51*, p. 88-89.

[Carlier, 84] J. Carlier, *Problèmes d'ordonnancement à contraintes de ressources: algorithmes et complexité*, Thèse d'État, MASI. 1984.

[Carlier and Pinson, 98] J. Carlier and E. Pinson, *Jackson pseudo preemptive schedule for the Pm/ri,qi/Cmax problem*, Ann. Oper. Res., 1998, *83*, p. 41-58.

[Demeulemeester and Herroelen, 92] E. L. Demeulemeester and W. S. Herroelen, *A Branch and Bound procedure for the multiple resource-constrained project scheduling problem*, Management Science, 1992, *38*, p. 1803-1818.

[Gupta, 88] J. N. D. Gupta, *Two stages hybrid flowshop scheduling problem*, Operational Research Society, 1988, *4*, p. 359-364.

[Perregaard, 95] M. Perregaard, *Branch and bound methods for the multiprocessor jobshop and flowshop scheduling problems*, Datalogisk Institut Kobenhavns Universitet - Master Thesis, 1995.

[Portmann *et al.*, 98] M. C. Portmann, A. Vignier, D. Dardilhac and D. Dezalay, *Branch and bound crossed with G.A to solve hybrid flowshops*, European J. Oper. Res., 1998, *107*, p. 339-400.

[Sprecher, 94] A. Sprecher, *Resource-Constrained Project Scheduling - Exact Methods for the Multi-Mode case*, Lecture Notes in Economics and Mathematical Systems, Springer Verlag, Berlin, 1994.

[Vandevelde, 94] A. Vandevelde, *Minimizing the makespan in a multiprocessor flow shop*, Eindhoven University of Technology - Master Thesis, 1994.

[Vignier, 97] A. Vignier, *Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachines ("flow-shop hybride")*, PhD Thesis, 1997.