CHAPTER V
BRANCH AND BOUND METHODS

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

LIST OF ACRONYMS

ACT[S_s(A')]	Average Completion Time of S_s(A')
Best_UB	Best known Upper Bound
HFS	Hybrid flowshop
HSB	Hybrid Set Bound (used in SE)
HSSB	Heuristic for the subset bound
IM	Interval Method
IM_1	Interval Method algorithm using the first lower bounding strategy
IM_2	Interval Method algorithm using the second lower bounding strategy
IM_3	Interval Method algorithm using the third lower bounding strategy
JB	Job Bound
LBM	Machine based Lower bound
LJB	Job based Lower bound
MCT[S_s(A')]	Maximum Completion Time of S_s(A')
SB	Set Bound
SE	Sequence Enumeration method
S_s(A')	Partial schedule involving the schedule of all jobs in I from stage 1 through stage s-1 along with the schedule for the jobs in A'
SSB	Subset bound
STI_{is}	Starting Time Interval of job i at stage s

CHAPTER V
BRANCH AND BOUND METHODS

Abstract

Minimizing the makespan of the hybrid flowshop scheduling problem is NP-Hard. This chapter aims at introducing exact methods for this problem. Two branch and bound algorithms are studied. An improved version of the sequence enumeration algorithm that has been proposed by Brah and Hunsucker (1991), and a generalization of the interval method that has been introduced by Carlier (1987) as a branching scheme for the single stage problem with release dates and tails. We first recall the sequence enumeration algorithm and introduce improvements of the branching scheme, as well as of the bounding operations. We then generalize the interval method to the hybrid flowshop case and present the overall algorithm along with different lower and upper bounding strategies. Numerical tests on different classes of configuration are carried out so as to evaluate the performances of both methods. Thus, the improved sequencing enumeration method is first compared to the original one, and is then compared to the new interval method. The latter is tested with different bounding strategies. Numerical tests show that the interval method outperforms the sequence enumeration method.

1. INTRODUCTION

Minimizing the makespan for the hybrid flowshop (HFS) scheduling problem is NP-Hard. In Chapter III we have presented and compared several heuristics for solving the HFS scheduling problem. In the same way in Chapter IV we have studied several lower bounds. The main purpose of this chapter is the study of branch and bound algorithms as exact methods for solving the HFS scheduling problem when minimizing the makespan. We study two different branching schemes, and based on the main results found in Chapter III and Chapter IV we design complete branch and bound algorithms.

Since for large HFS problems, with large number of stages and jobs, the complexity of finding the optimal solution is very high, these branch and bound algorithms can always be used as heuristics or improvement methods when truncated after running a certain amount of time.

In Chapter II we have suggested the study of two branching schemes, that is the *"Sequence Enumeration method"* and the "*Interval Method"*. In order to schedule the HFS, Brah and Hunsucker (1991) have proposed a branch and bound algorithm based on the branching scheme introduced by Bratley et al., (1975) for scheduling the one stage parallel machine problem, "*sequence enumeration method"*. Section **2** presents Brah's original algorithm, as well as several improvements. Carlier (1987) has proposed the interval method as a branching scheme for the single stage subproblem with release dates and tails. This branching scheme can be generalized to the multistage case problem, i.e. HFS. The overall derived algorithm is presented in Section **3** along with several branching strategies.

Before presenting the two algorithms, we hereafter recall how a general branch and bound algorithm proceeds for finding the optimal solution and recall the HFS scheduling problem we are studying.

THE BRANCH AND BOUND ALGORITHM

The branch and bound algorithm is a search technique, generally, used as an exact method for finding the optimal solution of combinatorial problems, or as a heuristic or improvement method if truncated, i.e. stopped before the solution found is proven optimal. The branch and bound algorithm constructs the so-called branch and bound tree to explore the space of solutions. Such a tree is made up of nodes representing subproblems derived from the original problem represented by the root node of the tree. A separation operation is used to divide a subproblem represented by a node into two or several disjoint subproblems or nodes. While exploring or constructing the tree, the algorithm does not necessarily solve the subproblems, but uses bounding operations to check whether a node is worth exploring. Bounding operations can be used, at each node, to compute lower and upper bounds on the corresponding subproblem. In a minimization problem the lower bound allows one to discard the subproblem or the node if its lower bound is greater than the best known upper bound.

Technically speaking, the algorithm (when solving a minimization problem) consists in selecting a node N of the tree (the root node at the beginning of the algorithm), and if that node is worth exploring, i.e. the node lower bound (LB(N)) is smaller than the best known Upper Bound UB, separates N into K new nodes (subproblems), Nk, k=1,..., K with $K \geq 2$. If LB(N)>UB, node N is simply discarded and no new nodes are generated. The new nodes are created using a separation operation. All the nodes that are created and not selected yet for possible branching are called *outstanding nodes* of the tree and are stored in a list. The structure of the list determines the order in which the tree is explored. A *stack* results in a depth-first search whereas a *queue* results in a breadth-first search. This list can also be sorted in a certain order allowing first visiting nodes with a specific priority, for example, giving higher priority to nodes with smaller bounds (best bound).

Note also that, at the separation operation, the union of solution spaces represented by the newly created nodes must be equal to the space solution at the father node, to make sure that the algorithm does not miss the optimal solution. They also have to be disjoint so as to avoid exploring solution subspaces more than one time. Two very important steps are critical in the design of the branch and bound algorithm. (i) *The selection* of the next node N to explore and to branch on from the list of nodes, and (ii) *the separation operation* used to divide a problem represented by node N into two or more subproblems.

HFS SCHEDULING PROBLEM

The HFS consists of the classical multistage flowshop, each stage $s \in \{1,\ldots, K\}$ is being composed of M_s parallel machines, i.e. identical in our case. The task here lies in scheduling a set of jobs I, where a job consists of one operation for each stage. Those operations have to be realized sequentially. At each stage s, each job j has a given processing time p_{js}. Each job j may have an initial release date r_{j1} and a final tail q_{jK}. We need to build a schedule with a minimum makespan. A schedule consists in assigning a specific machine to each operation of every job, as well as sequencing all operations

assigned to each machine, so that successive operations of a job do not overlap, and so that each machine at most processes one job at a time. Job splitting and job preemption are not allowed.

2. SEQUENCE ENUMERATION METHOD

The branching scheme proposed by Brah and Hunsucker (1991) consists of the enumeration of all possible sequences of all the jobs over all machines at all stages. At each node of the enumeration tree, an augmented partial schedule (at the currently considered stage) is obtained by adding a new job among the non-yet-scheduled ones, either on the currently scheduled machine, or on a new one. The lower bounds used rely upon the partial schedule.

The remainder of this section is organized as follows. Subsection **2.1** recalls the sequence enumeration scheme as well as the lower bounds used in Brah and Hunsucker (1991). In subsection **2.2** we present some improvements of this branch and bound algorithm. The sequence enumeration running time complexity remains very high and Brah's original algorithm can tackle, in reasonable time, only very small problems. Tighter lower bounds and efficient implementation are then needed to reduce the enumeration space and time. In subsection **2.3** we show how some of the single stage based lower bounds, presented in Chapter IV, can be adapted for this sequence enumeration branching scheme. Subsection **2.4** presents the implementation of the chosen bounding strategies and subsection **2.5** presents the final algorithm. Subsection **2.6** gives numerical results obtained comparing the original algorithm of Brah and Hunsucher (1991) with our new version. Subsection **2.7** concludes the study of the sequence enumeration method.

2.1 THE ALGORITHM PROPOSED BY BRAH AND HUNSUCKER (1991)

2.1.1 BRANCHING SCHEME

The branching scheme used in Brah and Hunsucker (1991) is based upon the enumeration of all possible sequences of all jobs at all stages. It starts at the first stage and enumerates all job sequences over all the parallel machines composing that stage. It then moves forward, and schedules in the same way the subsequent stages, until the last stage has been reached and scheduled. The enumeration is realized by holding a partial schedule at each node and deciding to schedule a job, from the not-yet-scheduled ones, either on the currently scheduled machine, or on a new one. So each node of the tree represents the sequencing of one job. The computation of lower bounds, and of better feasible solutions or upper bounds allow one to discard paths, i.e. sequence of nodes, leading to solutions with higher makespan than the currently best known upper bound.

Sequence enumeration is carried out by building a tree made up of two types of nodes, the square and circle nodes. Nodes represent jobs, and a path linking the root node to one of the leaves of the branch and bound tree represents a complete schedule. If the path passes through a circle node, the corresponding job i is scheduled on the currently scheduled machine. However, if the path passes through a square node, the corresponding job i is scheduled on a new machine which becomes the current machine.

Once a (new) machine is selected for scheduling, it remains the only machine to which jobs are assigned until another (new) machine is selected. A machine is selected only once. Figure 5.1 gives a partial view at some stage s of an example of the sequence enumeration tree. Some of the created nodes will not be considered when branching so as to avoid constructing duplicate schedules, see later rules 5 and 6.

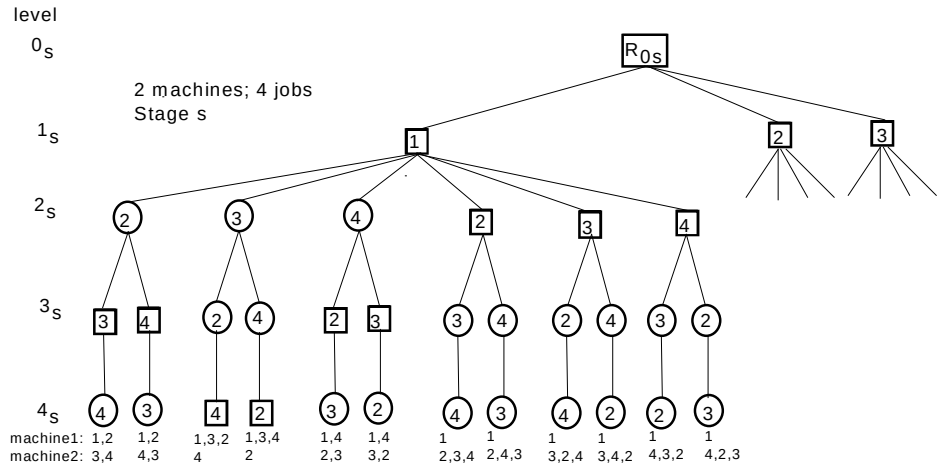

Figure 5.1: Sequence enumeration scheme
A partial schedule: possible sequences at stage s starting with job 1

The branch and bound tree is made up of n*K levels, one level for each job i=1, ..., n at each stage s=1, ..., K. On each specific path from root to leaf, each job is scheduled only once at each stage. Therefore, there are n*K levels that need to be explored so as to build a complete HFS schedule. The necessary rules, proposed by Brah and Hunsucker (1991), to develop the branch and bound tree, are as follows.

- *Rule 1.* Level 0_s contains the fictitious or root node R_{0s} at stage s = 1, ..., K of the HFS problem. R_{0s} represents the first fictitious operation that precedes all operations at stage s.
- *Rule 2.* Level 1_s contains the square nodes 1, 2, ..., x where $x = n - M_s + 1$, i.e. at each stage s we start assigning a first machine to some job. Note that the number of the node represents the job sequenced at that node.
- *Rule 3.* A path from level 0_s to level i_s, i=1, ..., n and s=1, ..., K, may be extended to level $(i+1)_s$ by any of square or circle nodes 1, 2, ..., n, respecting rule 4 through rule 7. In other words, at some level i_s, we either assign, at stage s, a not yet scheduled job j to the currently scheduled machine (circle node) or to a new machine (square node).
- *Rule 4.* In any path from level 0_s to level i_s, i=1, ..., n and s=1, ..., K, a job j may at most appear once as a square or circle node, i.e. a job is scheduled once, either as the first one on some machine or sequenced at the end of the sequence of jobs already assigned to the currently scheduled machine.
- *Rule 5.* A square node v may not be used to extend a path at level i_s, which already contains square node r with r>v. This rule avoids considering a sequence that has already been explored. Because there is no ordering between the identical machines, we can always impose that square nodes appear in increasing order of job number.
- *Rule 6.* No path may be extended in such a way that it contains more than M_s square nodes at each stage s, i.e. at stage s a maximum of M_s machines can be used.

- *Rule 7.* No path may terminate in such a way that it contains less than M_s square nodes at each stage s unless $n<M_s$, i.e. all the M_s machines must be used at stage s.

A complete sequence starting from level 0_1 to level n_K corresponds to a complete HFS schedule. This sequence is composed of $\sum_{s=1}^{K} M_s$ subsequences, M_s subsequences at each stage. Each subsequence starts with a square node corresponding to the first job on a machine. A complete schedule is obtained by assigning each subsequence to one machine at the corresponding stage. Also, the complete sequence suffices to compute job start and completion times. Indeed, each of the subsequences corresponds to a path of disjunctive arcs in the graph representation of the HFS schedule. (See Chapter II, Section 4).

2.1.2 THE LOWER BOUNDS

The following notation defines the state of the partial schedule obtained at some node in terms of the stage s currently being scheduled, the already scheduled and the not yet scheduled jobs as well as the completion times of the already scheduled machines at stage s.
Let

- $A \subset I$ be a subset of $I=\{1, \ldots, n\}$ corresponding to the jobs already sequenced at the current stage s.
- $A'=A \cup \{i\}$ be the set containing all jobs in A and a job $i \notin A$. So, i corresponds to the last sequenced job at stage s.
- I-A' be the set of the not yet scheduled jobs at the current stage s.
- $S_s(A)$ represent a partial schedule involving the schedule of all jobs in I from stage 1 through stage s-1 along with the schedule for the jobs in set A at stage s. In other words, $S_s(A)$ represents the path coming from the root node at stage 1 and going till the last node (at level $|A_s|$) of the sequence formed with those jobs in set A at stage s.
- $S_s(A')$ represent the schedule formed by appending job i to $S_s(A)$
- $C[S_s(A'), k]$ be the completion time of the partial schedule $S_s(A')$ on machine k, where k is one of the M_s parallel machines at stage s
- $q_{is}= \sum_{s'=s+1}^{K} p_{is'} + q_{ik}$ be the tail of job i at stage s defined as the remaining processing time after stage s plus the final tail.

The average completion time $ACT[S_s(A')]$ and the maximum completion time $MCT[S_s(A')]$ given by the partial schedule $S_s(A')$ at stage s are defined as follows:

$$ACT[S_s(A')] = \sum_{k=1}^{M_s} C[S_s(A'), k]/M_s + \sum_{i \in I-A'} p_{is} / M_s \qquad (5.1)$$

$$MCT[S_s(A')] = \max_k C[S_s(A'), k] \qquad (5.2)$$

In fact, both $ACT[S_s(A')]$ and $MCT[S_s(A')]$ are lower bounds on the completion time at stage s, over all possible schedules obtained from the partial schedule $S_s(A')$. In their paper, Brah and Hunsucker (1991) have used two lower bounds on the global makespan, the machine (LBM) and job (LBJ) bounds.

- if $ACT[S_s(A')] \geq MCT[S_s(A')]$, $LBM[S_s(A')] = ACT[S_s(A')] + \min_{i \in I-A'} q_{is}$ (5.3)

  otherwise $LBM[S_s(A')] = MCT[S_s(A')] + \min_{i \in A'} q_{is}$ (5.4)
- $LBJ[S_s(A')] = \min_{k} C[S_s(A'), k] + \max_{i \in I-A'} (p_{is} + q_{is})$ (5.5)
- $LB[S_s(A')] = \max \{LBM[S_s(A')], LBJ[S_s(A')]\}$ (5.6)

LBM is easily understood using a graphical representation of the partial schedule $S_s(A')$ at stage s. Figure 5.2(a) shows the case where $ACT[S_s(A')] > MCT[S_s(A')]$. There exists at least one machine whose completion time at stage s is larger than or equal to $ACT[S_s(A')]$ and hence at the least the minimum tail in set I-A' can be added to $ACT[S_s(A')]$ so as to come up with a global lower bound. Indeed, in this case we are sure that one of the jobs in I-A' will be the last to be processed at stage s. Figure 5.2(b) shows the case where $ACT[S_s(A')] < MCT[S_s(A')]$ and the lower is obtained by adding the minimum tail over jobs in set A' to the $MCT[S_s(A')]$ since the last finishing job at stage s at time $MCT[S_s(A')]$ is in A'.

The minimum remaining processing time at stage s and the subsequent ones of job $i^* = \arg\max_{i \in I-A'} (p_{is}+q_{is})$ is equal to $\max_{i \in I-A'} (p_{is}+q_{is})$. Job i^* is assigned to some machine at stage s at time $\min_{k} C[S_s(A'), k]$ at the earliest. Hence $LBJ[S_s(A')]$ is a global lower bound.

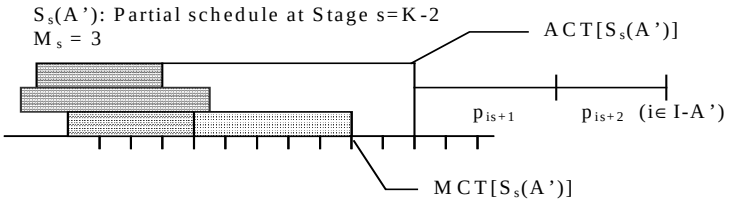

Figure 5.2:(a) Partial schedule at stage s=K-2, ACT > MCT.

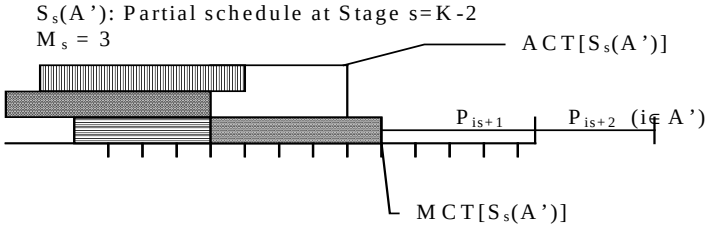

Figure 5.2: (b) Partial schedule at stage s=K-2, ACT < MCT.

Since the jobs in set I-A' will be scheduled only on the leftover machines (the current one and the not yet scheduled ones), Brah & Hunsucker (1991) have improved LBM. In equation (5.1), they suggest that the lower bound is more effective when dividing the sum of the remaining processing times at stage s of the jobs in I-A' by the number of the remaining machines instead of M_s.

2.2 IMPROVEMENTS AND ADAPTATIONS

In this section we present our improvements to the algorithm proposed by Brah and Hunsucker (1991). We first introduce some notations. For any partial schedule $S_s(A)$ let:

- ω be the currently scheduled machine

- {1,…, ω-1} be the set of the already scheduled machines in the partial schedule $S_s(A)$ at stage s
- m_s = {ω+1, …, M_s} be the set of the not yet scheduled machines
- V ={ω} ∪ m_s be the set of the leftover machines
- c_{is} be the completion time of job i at stage s
- r_{is} be the release date of job i at stage s

Actually, since we are scheduling the stages forward and because the partial schedule $S_{s-1}(I)$ is fixed, $r_{is}=c_{is-1}$ for s>1 and $r_{i1} \geq 0$ for all i∈I.

It is easy to see that c_{is} can be defined by $c_{is} = \max\{C[S_s(A), \omega], r_{is}\} + p_{is}$ where i∉A and job i is assigned next to machine ω at stage s, because job i must be released at stage s and is sequenced after the jobs in A.

2.2.1 A SYSTEMATIC IMPLEMENTATION OF RULE 5

Rule 5 states that no square node v may be extended at level i_s if a square node r>v has appeared in an upper level (j_s, j<i). Since we impose that square nodes appear in increasing order of job number, this rule avoids constructing previously considered schedules. Its implementation has not been specified in Brah's paper. We therefore propose a systematic implementation, which takes a look forward so as to avoid generating a node whose paths originating from it are going to be cut off by rule 5 or rule 6. Figure 5.3 shows that there is no need to create circle node 4 at level 2_s because all square nodes 1 and 2 at level 3_s and level 4_s will be discarded by rule 5. We next formalize this idea.

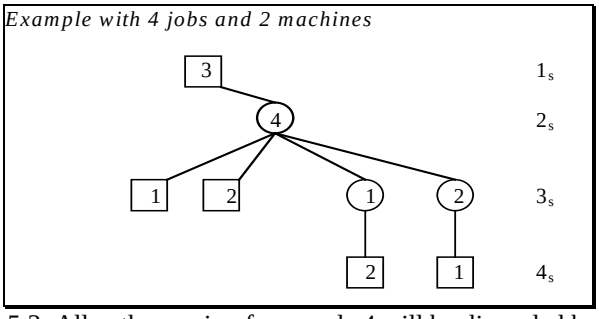

Figure 5.3: All paths coming from node 4 will be discarded by rule 5

Let e be the first (square) node of some path at stage s, e is the first job assigned at level 1_s to the first machine at stage s.

(i) If a circle node i is chosen to extend a path at stage s and $|\{j \in I\text{-}A', j>e\}| < |m_s|$, node i must be discarded. (A'=A∪{i})
(ii) If a square node i is chosen to extend a path at stage s and $|\{j \in I\text{-}A', j>e\}| < |m_s|-1$, node i must be discarded. (A'=A∪{i})

Rule 5 imposes that square nodes appear in increasing order of job number. Rules 6 and 7 impose that each complete subpath at stage s must contain M_s square nodes. If the number of machines in m_s ($|m_s|$) is larger than the number of jobs in I-A' with a number

larger than e ($|\{j \in I\text{-}A', j>e\}|$), it simply means that a least a square node (v) with a number smaller than e (v<e) will appear later on such a path.

2.2.2 LOWER BOUND ON THE COMPLETION TIME OF THE NOT YET SCHEDULED MACHINES

In the lower bounds presented in Brah and Hunsucker (1991), the completion times of the machines are computed as follows:

$$C[S_s(A'), k] \geq 0 \quad \text{for } k = 1,\ldots, \omega$$
$$C[S_s(A'), k] = 0 \quad \text{for all } k \in m_s = \{\omega+1,\ldots, M_s\}$$

The improvement of the computation of the completion times of the machines in set m_s might lead to the improvement of the lower bounds, because these values are used in the computation of the lower bounds LBM and LBJ.

For any partial schedule $S_s(A')$ we set the completion times (which are also the availability times or the earliest starting times) of the machines in set m_s, to the minimum release date of the not yet scheduled jobs at stage s, i.e. $C[S_s(A'), k] = \min_{i \in I-A'} r_{is}$; $k \in m_s$. Furthermore, let $r_{[ts]}$ be the t^{th} smallest release date in set I-A' at stage s.

$$C[S_s(A'), k] = r_{[ts]}, \text{ for } k = \omega + 1, \ldots, M_s; \text{ with } t = k - \omega, \ldots, |m_s| \qquad (5.7)$$

This equation comes from the following observations: (i) For any partial schedule $S_s(A')$ the release dates of the not yet scheduled jobs (jobs in I-A') are known. (ii) According to rule 7 no machine at any stage will be kept idle unless $n < M_s$. (iii) In order to start processing on each not yet scheduled machines, we have to wait the $|m_s|$ smallest release dates, at least, among the not yet scheduled jobs.

2.2.3 AT ANY STAGE THE LOWER BOUND OF ANY NODE IS AT LEAST EQUAL TO THE FATHER NODE LOWER BOUND

If at some node a lower bound LBM or LBJ becomes active, this lower bound might be possible lower than the parent node's lower bound. We hereafter present two cases where a node lower bound computed using the partial schedule $S_s(A')$, is smaller than its parent one, which is computed using the partial schedule $S_s(A)$, $A'=A\cup\{i\}$. We hereafter present two cases showing that $LB[S_s(A)] > LB[S_s(A')]$. Figures 5.4 (a) and (b) depict these two cases.

Case 1. Suppose the following:

$LB[S_s(A)] = ACT[S_s(A)] + \min_{i \in I-A} q_{is}$, i.e. the father node's lower bound

$LB[S_s(A')] = MCT[S_s(A')] + \min_{i \in A'} q_{is}$, i.e. the node's lower bound

(because $MCT[S_s(A')] > ACT[S_s(A')]$)

If $\min_{i \in I-A} q_{is} - \min_{i \in A'} q_{is} > MCT[S_s(A')] - ACT[S_s(A')]$

Then $LB[S_s(A)] > LB[S_s(A')]$

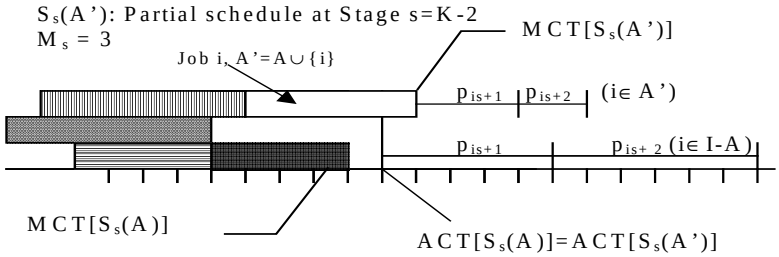

Figure 5.4: (a) $LB[S_s(A)] > LB[S_s(A')]$, Case 1

Case 2. Suppose the following:

$LB[S_s(A)] = \min_k (C[S_s(A), k]) + \max_{i \in I-A} (p_{is} + q_{is})$, i.e. the father node's lower bound

$LB[S_s(A')] = \min_k (C[S_s(A'), k]) + \max_{i \in I-A'} (p_{is} + q_{is})$, i.e. the node's lower bound

and $\min_k (C[S_s(A'), k]) - \min_k (C[S_s(A), k]) < \max_{i \in I-A} (p_{is} + q_{is}) - \max_{i \in I-A'} (p_{is} + q_{is})$

i.e. i^* has been assigned $A'=A \cup \{i^*\}$ with $i^* = \arg \max_{i \in I-A'} (p_{is} + q_{is})$

Then $LB[S_s(A)] > LB[S_s(A')]$

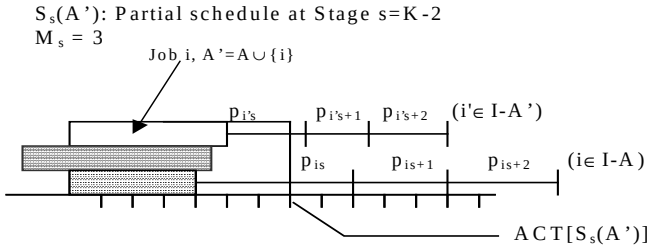

Figure 5.4: (b) $LB[S_s(A)] > LB[S_s(A')]$, Case 2

Hence, whatever the active lower bound, we impose that lower bound of any node must satisfy:

$$LB[S_s(A')] \geq LB[S_s(A)].$$

2.2.4 IMPROVEMENT OF LBJ $[S_s(A')]$ AND LBM$[S_s(A')]$

a. Since jobs in set I-A' are going to be scheduled only on the leftover machines in set V, $LBJ[S_s(A')]$ can be improved by computing the minimum completion time only over the machines in set V. Equation (5.5) is then replaced by equation (5.8).

$$LBJ[S_s(A')] = \min_{k \in V} C[S_j(A'), k] + \max_{i \in I-A'} (p_{is} + q_{is}) \qquad (5.8)$$

Note that this lower bound (5.8) is more effective than the one in (5.5) when the machines completion times are computed using equation (5.7).

b. One can see that:

- $\varphi = MCT[S_s(A')] + \min_{i \in A'} q_{is}$ is a lower bound since the tail of the last job defining $MCT[S_s(A')]$ is at least equal to $\min_{i \in A'} q_{is}$, whatever are the values of $ACT[S_s(A')]$ and $MCT[S_s(A')]$.

- $\gamma = ACT[S_s(A')] + \min_{i \in I} q_{is}$ is a lower bound if $ACT[S_s(A')] < MCT[S_s(A')]$, because at least one job of I finishes at stage s at or later than $ACT[S_s(A')]$, and its tail is at least $\min_{i \in I} q_{is}$.
- $\chi = ACT[S_s(A')] + \min_{i \in I-A'} q_{is}$ is also a lower bound if $ACT[S_s(A')] \geq MCT[S_s(A')]$, because at least one job of I-A' finishes at stage s at or later than $ACT[S_s(A')]$, and its tail is at least $\min_{i \in I-A'} q_{is}$.

Equations (5.3) and (5.4) can then be replaced by equations (5.9) and (5.10).

If $ACT[S_s(A')] < MCT[S_s(A')]$ $LBM[S_s(A')] = \max(\varphi, \gamma)$ (5.9)

Otherwise $= \max(\varphi, \chi)$ (5.10)

2.3 SINGLE STAGE BASED LOWER BOUNDS

In this subsection we show how the single stage lower bounds presented in Chapter II, can be adapted so as to be used in the sequence enumeration scheme. Three bounds will be derived, i.e. a Job Bound (JB), a Set Bound (SB) and a Hybrid Set Bound (HSB). Given a partial schedule $S_s(A')$ the following are known:

1. From stage 1 through stage s-1 all the jobs are scheduled. So, for all $i \in I$, c_{is-1} is known.
2. At stage s all jobs in set A' are scheduled and the following are known:
   - c_{is} ; for all $i \in A'$
   - $r_{is} = c_{is-1}$ for $s>1$ and $r_{i1} \geq 0$ for all $i \in I-A'$
   - $C[S_s(A'), k]$; for $k = 1, \ldots, \omega$
   - All machines in set $m_s = \{\omega+1, \ldots, M_s\}$ are idle and their earliest starting times are the $(M_s-\omega)$ smallest remaining release dates r_{is}, over all $i \in I-A'$ computed using equation (5.7).
3. The remaining jobs in set I-A' will be assigned only to those machines in set $V=\{\omega\} \cup m_s$

2.3.1 JOB BOUND (JB)

We know that each job i passes through stage s, spends some time p_{is} being processed and spends at least its tail q_{is} at the subsequent stages s'=s+1, ..., K. We therefore compute a lower bound on the completion time of any job i at the last stage K.

For the scheduled jobs ($i \in A'$): $JB1[S_s(A')] = \max_{i \in A'} (c_{is} + q_{is})$ (5.11)

For the not yet scheduled jobs ($i \in I-A'$) we first compute their Earliest Starting Times at stage s: $EST_{is} = \max\{c_{is-1}, \min_{k \in V} (C[S_s(A'), k])\}$ for all $i \in I-A'$, and then define the job bound: $JB2[S_s(A')] = \max_{i \in I-A'} (EST_{is} + p_{is} + q_{is})$ (5.12)

Hence for stage s: $JB[S_s(A')] = \max(JB1, JB2)$ (5.13)

Proposition 5.1: $JB[S_s(A')] \geq JB2[S_s(A')] \geq LBJ[S_s(A')]$

Proof. The first inequality is the definition of JB. For the second inequality using the redefinition (5.8) of LBJ we obtain

$$LBJ[S_s(A')] = \max_{i \in I-A'} (\min_{k \in V} C[S_j(A'), k] + p_{is} + q_{is})$$

$$\leq \max_{i \in I-A'} [\max\{c_{is-1}, \min_{k \in V} (C[S_s(A'), k])\} + p_{is} + q_{is}] = JB2[S_s(A')]$$

2.3.2 SET BOUND (SB)

Set bound on all machines and jobs in I-A' (SB1). According to the relaxation mentioned in Chapter II, Section 4, the Set Bound (SB) is a lower bound on the makespan of the single stage subproblem with release dates and tails. The partial schedule $S_s(A')$ can be taken into account in the definition of SB in such a way that SB can be used as a lower bound at any node of the branch and bound tree.

Let $\delta = \sum_{k=1}^{M_s} C[S_s(A'),k] + \sum_{i \in I-A'} p_{is}$

$QM_s(I) = \{[1_s], [2_s], ..., [M_s]\}$ be the set of the M_s jobs in set I having the smallest tails at stage s

$$SB1[S_s(A')] = \lceil (\delta + \sum_{i \in QM_s} q_{is}) / M_s \rceil \qquad (5.14)$$

δ is the equivalent part of the sum of release dates and remaining processing times of the set bound SB(I-A'). The completion times of the machines k = 1, …, ω, are known and can be regarded as the first ω minimum release dates we have to wait, before starting processing the jobs in I-A'. The completion times of machines k=ω+1, …, M_s, also regarded as release dates in I-A', are computed as in equation (5.7).

Set bound on the leftover machines and jobs in I-A' (SB2). Since jobs in set I-A' will be scheduled only on the machines in set V, we can consider our problem as made up only of this set of machines, i.e. set V. Actually, for the other machines, k = 1, …, ω-1, $JB1[S_j(A')]$ is a good lower bound on their completion times because their scheduling is fixed at stage s and will not change down the branch and bound tree.

Let $RM_s(I-A') = \{[1_s], [2_s], ... , [|m_s|_s]\}$ be the set of the $|m_s|$ jobs in set I-A', having the smallest release dates. Remember that $m_s=\{\omega+1,..., M_s\}$ is the set of the leftover machines excluding the currently scheduled one, machine ω.

$QM_s(J) = \{(1_s), (2_s), ..., (|m_s|+1)\}$ be the set of the $(|m_s|+1)=|V|$ jobs in set J, having the smallest tails at stage s, where set J = I-A'$\cup\{i^*\}$; i^* is the last job assigned to machine ω.

$$SB2[S_s(A')] = \lceil C[S_s(A'), \omega] + (\sum_{i \in RM_s(I-A')} r_{is} + \sum_{i \in I-A'} p_{is} + \sum_{i \in QM_s(J)} q_{is}) / |V| \rceil \qquad (5.15)$$

$C[S_s(A'), \omega]$ is considered as a release date on machine ω for any job in set I-A'. The minimum tail accounted for machine ω is the tail of the last job assigned to it, because we are sure that this tail appears in the set bound. Such job i^* is included in the set J.

$$SB[S_s(A')] = \max \{SB1[S_s(A')], SB2[S_s(A')]\} \qquad (6.16)$$

Proposition 5.2: $SB1[S_s(A')] \lesseqgtr LBM[S_s(A')]$
where $\lesseqgtr$ means that the left hand side of $\lesseqgtr$ could be larger or smaller than the right hand side. In Proposition 5.2, $\lesseqgtr$ means that no bound always dominates the other one.

Proof. $SB1[S_s(A')] = \lceil (\sum_{k}^{M_s} C[S_s(A'),k] + \sum_{i \in I-A'} p_{is} + \sum_{i \in QM_s} q_{is}) / M_s \rceil$

Case 1. $LBM[S_s(A')] = (\sum_{k}^{M_s} C[S_s(A'),k] + \sum_{i \in I-A'} p_{is}) / M_s + \min_{i \in I-A'} q_{is}$ (equation 5.3)

Since $\sum_{i \in QM_s} q_{is} / M_s \lesseqgtr \min_{i \in I-A'} q_{is}$ then $SB1[S_s(A')] \lesseqgtr LBM[S_s(A')]$

Case 2. $LBM[S_s(A')] = \max_{k} C[S_s(A'), k] + \min_{i \in A'} q_{is}$ (equation 5.4)

Since $\sum_{i \in QM_s} q_{is} / M_s \lesseqgtr \min_{i \in A'} q_{is}$

And $\lceil (\sum_{k} C[S_s(A'),k] + \sum_{i \in I-A'} p_{is}) / M_s \rceil \lesseqgtr \max_{k} C[S_s(A'), k]$

then $SB1[S_s(A')] \lesseqgtr LBM[S_s(A')]$

According to Proposition 5.2 we still need to compute $LBM[S_s(A')]$ because it might lead to a larger value than $SB1[S_s(A')]$ and conversely. We hereafter give an example where $LBM[S_s(A')] > SB1[S_s(A')]$.

From Example 5.1 and Figure 5.5 we compute the following:

$\sum_{k=1}^{M_s} C[S_s(A'),k] = 17, \quad \sum_{i \in I-A'} p_{is} = 12, \quad \sum_{i \in QM_s} q_{is} = 4$

$SB1[S_s(A')] = \lceil (17+12+4)/3 \rceil = 11 < LBM[S_s(A')] = 10 + 2 = 12$

Example 5.1: $M_s = 3$, $SB(I) = 11$

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

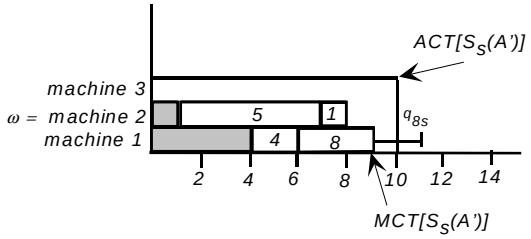

Figure 5.5: A partial schedule $S_s(A')$, where A'={4, 8, 5, 1}, I-A'={2, 3, 6, 7}

For the sake of comparisons we hereafter give two other examples where SB2>SB1 and JB1>JB2, respectively.

Example where SB1[$S_s(A')$] < SB2[$S_s(A')$]

From Example 5.1 and Figure 5.5, we compute the following:

$C[S_s(A'), \omega] = 8$, $\sum_{i \in RM_s(I-A')} r_{is} = 1$, $\sum_{i \in QM_s(J)} q_{is} = 4$

$SB1[S_s(A')] = 11 < SB2[S_s(A')] = \lceil (8+1+12+4)/2 \rceil = 13$

Example where JB1[$S_s(A')$] > SB2[$S_s(A')$]

From Example 5.1 and Figure 5.5, assume that $q_{8s} = 5$, which does not change the computation $SB2[S_s(A')]$.

$JB1[S_s(A')] = 14 > SB2[S_s(A')] = 13$

2.3.3 HYBRID SET BOUND (HSB)

Each time we augment the partial schedule $S_s(A)$ by assigning a job i to machine ω we can update its release date at the subsequent stages s'= s+1, ..., K as follows:

$$r_{is+1} = \max \{C[S_s(A'), \omega], r_{is+1}\}$$
$$r_{is'} = r_{is'-1} + p_{is'-1}; \text{ for } s' = s+2, \ldots, K$$

Once we have at our disposal the release dates of jobs at the subsequent stages s'=s+1,...,K, we can compute set bounds $SB_{s'}(I)$ for s'=s+1,...,K. By maximizing over the subsequent stages we obtain a global lower bound (HSB) on the partial schedule $S_s(A')$ constructed.

$$HSB[S_s(A')] = \max_{s'=s+1,\ldots,S} SB_{s'}(I) \qquad (5.17)$$

Although SB is not time consuming, i.e. O(n), computing HSB at the upper levels of the branch and bound tree is time consuming since each time we augment the partial schedule $S_s(A)$ with one job, we compute (K–s) lower bounds, i.e. $HSB[S_s(A')]$. However, HSB has the big advantage of detecting a possible bottleneck among the not yet scheduled stages, and in which case it provides a tighter lower bound. In our implementation we do not compute HSB at each node, see next subsection for our lower bounding strategy.

2.4 IMPLEMENTATION

LOWER BOUNDING STRATEGY. As far as the lower bounds are concerned, we end up with four of them, i.e. the improved LBM, JB, SB and HSB. LBJ has been ignored because it is dominated by JB2. None of these four lower bounds is dominating the others. In order to reduce the computational complexity of the branch and bound algorithm, at each node the lower bounds are computed in the non-decreasing order of their running time complexities. Actually, LBM, JB, SB1 and SB2 are computed in O(n) where n is the number of jobs in I. However, SB1 and SB2 will take a bit more running time because we need to find the M_s and |V| smallest release dates and tails, respectively. HSB is computed in O(nK), where K is the number of stages. We first compute LBM and JB, followed by SB1, SB2 and finally we compute HSB.

A lower bound is computed only if the preceding one fails to discard the evaluated node. A computational effort is then allowed only if it is really needed. Furthermore, since

computing HSB at each level of the tree is time consuming, we decide to use it only at those levels where the schedule of a stage is completely constructed, i.e. at levels 0_s, s=1,...,K. HSB is certainly much effective when all jobs are assigned at stage s. Indeed, if the complete schedule of a stage s is fixed, real release times of all jobs at subsequent stage s+1 are known, and allow to compute effective lower bounds. Doing so, HSB is computed only K times instead of K*n times, along an entire path from the root node to a final leaf.

UPPER BOUNDING STRATEGY. At the root node we use the best known upper bound, i.e. Best selected heuristics in Chapter III, that is CDSH, CDSV, CDSG, SF, SFR and SBP. For a detailed description of this heuristics see Chapter III. In order to come up with an upper bound at each node of the branch and bound, we need to use a "*building*" heuristic so as to complete the partial schedule at the corresponding schedule. Based on the conclusions of Chapter III and after many preliminary tests, we decide to use SF_J, so as to complete the partial schedule $S_s(A')$ and come up with an upper bound only at nodes corresponding to level 0_s, for all s=2,...,K.

SEARCH STRATEGY. As a search strategy we use the same as the one used in Brah and Hunsucker (1991), the deepest node with the least lower bound rule. The next node to explore is the node at the deepest level of the tree, and if there is more than one node at this level we choose the node with the smallest lower bound.

THE GENERAL ALGORITHM.

Let Ni be any square or cirle node i created from node N.

Sequence Enumeration branch and bound algorithm (SE)

Read the data of the HFS problem;
Best_UB = UB(Root) = best makespan; {CDSH, CDSV, CDSG SF, SFR, SBP}
Add Root node to List-Of-Node;
While List-Of-Node is not empty
    Remove node N at the deepest level with LB(N) minimal from List-Of-Node;
    If level is equal to 0_s
        terminate the schedule of node N and compute UB(N)
        If UB(N) < Best_UB ➔ *Best_UB = UB(N)*
    End-if;
    If LB(N) < Best_UB ➔
        Construct branches by creating nodes Ni, i=1,2,…
            (square and circle nodes)
        If the created nodes are at level n_K
            $Best\_UB = \min\{Best\_UB, \min_i (UB(Ni))\}$
        Else Compute nodes lower bounds; add Ni, i=1,2,… to List-Of-Node;
    End-if;
End-while;

2.5 NUMERICAL TESTS

The original sequence enumeration algorithm and the improved one have been programmed using an object-oriented language (C++). The tests have been carried out on a Pentium 120. All the jobs have zero release dates and tails at the first and the last stage, respectively. The processing times were generated uniformly with p_{is} = [1..100], i∈I, s=1,...,K. Different configurations, in terms of number of jobs, number of stages and number of machines at each stage, have been designed. For each specific configuration, 20 instances have been tested. The original algorithm of Brah and Hunsucker (1991) has been compared with our new version on the same sets of problems. The two algorithms have also been tested on mid-size problems so as to appreciate the robustness of the sequence enumeration method. All tested problems have been stopped after 10 minutes of running time.

Let us first recall some results obtained by other methods for scheduling the HFS. The largest instances tested by Brah and Hunsucker (1991), had 6 jobs, 5 stages, 3 machines at stage 3 and 2 machines everywhere else. They found the optimal solution after 12 hours running time on PC XT. For the case where *no bottleneck stage exists* (all stages have the same number of machines and data range or distribution is the same at all stages) Table 5.1 gives a summary of some known results. It shows the largest size problem (i.e. the number of jobs and the number of machines at each stage), the percentage of deviation from the lower bound. *Where there is a bottleneck stage* the problems are much easier and strongly depend on the way the bottleneck stage is determined. Here, we do not report on bottleneck configuration results.

Authors	Method	n	Machines	% to the LB	Optimized criterion
Guinet 1996a	Primal dual heuristic	30	5-5-5-5-5	18.01 %	Makespan
Guinet *et al.*, 1996b	CDS based	30	5-5-5	18.93 %	Makespan
Moursli and Pochet 1996	SBP based	100	8-8-8-8-8	16.00 %	Makespan
Vignier *et al.*, 1996b	BaB (Brah & al 91)	15	2-2-2	Not Optimal after 2H	Sum of completion time

Table 5.1: Known results obtained by other methods for non-bottleneck configurations

						% of average deviation from the best LB			Average running **Time** in seconds for 20 instances	
n	M1	M2	M3	M4	M5	RootUB	BaB	Brah	BaB	Brah
4	2	2				9.85	0	0	0.02	0.02
4	2	2	2			5.93	0	0	0.04	0.07
4	2	2	3	2	2	6.60	0	0	1.30	2.22
6	2	2				7.30	0	0	0.18	1.13
6	2	2	2			10.30	0	0.4	4.2	105
8	2	2				4.80	0	0.8	7.7	230
10	2	2				3.07	0	1.9	57	550
15	2	2				1.56	1	1.6	430	481
8	2	2	2			10.90	0	8.3	120	581
6	2	2	3	2	2	8.38	1.7	6.2	190	444
8	2	2	3	2	2	15.70	15.0	15.9	600	601
10	2	2	2			12.50	5.1	12.0	490	601

Table 5.2: Comparing Brah's algorithm (*Brah*) with the improved one (*BaB*)

In Table 5.2 we give the results of the tests that were carried out. Column n gives the number of jobs. Column Ms gives the number of parallel machines at stage s, s=1, …, 5. The first set of columns gives the percentage of the average deviation of the corresponding algorithm from the best known lower bound found by both algorithms. The second set of columns gives the average running time in seconds over 20 instances

of the corresponding algorithm. Column 'RootUB' gives the percentage of average deviation of the best known upper bound at the root node from the best lower bound found. Column 'Brah' and Column 'BaB', respectively, give the results for Brah's algorithm and our's (average deviation and time).

2.6 CONCLUSION

Although the problems remain hard to solve for the new algorithm, numerical tests show that it has drastically reduced the number of the explored nodes as well as the running time needed to reach an optimal solution when compared to the original algorithm from Brah and Hunsucker (1991). The extra running time needed for the new lower bounds has been largely compensated by the reduction in the number of nodes explored.

In the following section we present a different branching scheme for the HFS problem, that is the *Interval Method*.

3. INTERVAL METHOD

3.1 INTRODUCTION

Carlier (1987) has proposed the interval method as a branching scheme for the single stage subproblem with release dates and tails. See Chapter II Section, 5.3 for a brief description of this algorithm. In this section we generalize the interval method to the HFS problem.

The interval method is a branching scheme completely different from the sequence enumeration method. The main idea is to assign to each job i at each stage s its possible Starting Time Interval STI_{is} according to some known or Best Upper Bound (Best_UB). $STI_{is}=[r_{is}, d_{is}]$ with d_{is} = Best_UB-q_{is}-1, where d_{is} can be seen as the due date of job i at stage s so that its processing does not exceed (Best_UB - 1). Note that we are only interested in values strictly smaller than Best_UB, hence the reduction of Best_UB by one.

As a separation operation one can split STI_{is} into several subintervals for creating subproblems or nodes in the search tree. One can also split STI_{is} into two equal or non-equal parts. Actually, in some optimal solution only one starting time $t_{is} \in STI_{is}$ is assigned to job i and we do not have any information on which part of STI_{is} this starting time might take place. One appropriate strategy as branching scheme is to use *dichotomous search* and split STI_{is} into two equal parts every time job i is selected for branching.

The effectiveness of the interval method in a branch and bound framework relies on the effectiveness of the bounds used. Indeed, an improved upper bound reduces STI_{is} and therefore accelerates the search of the optimal solution. Different heuristics (see Chapter III) and lower bounds (see Chapter IV) for the HFS scheduling problem with diverse effectiveness are at our disposal. The remainder of this chapter is organized as follows. Subsection **3.2** presents different upper and lower bounding strategies. The separation

operation concerns one branching operation at some branching stage. Subsection **3.3** presents our generalization of the interval method to the HFS case. In Chapter IV we have selected the set bound (SB) and the heuristic for subset bound (HSSB) for use in our branch and bound algorithm. In our branching scheme, the selection of the branching job and stage requires the computation of several lower bounds, which is time consuming when using HSSB. Subsection **3.4** presents how we implemented the different proposed lower bounding strategies. Also, in order to reduce run time subsection **3.5** presents how to compute the potential increase of the lower bound when branching on some job at some stage without recomputing HSSB. The final algorithm is presented in subsection **3.6.** Several versions of the interval method, using different bounding strategies are compared to the improved version of the sequence enumeration method (presented in Section **2**) on different HFS configurations. Section **4** presents the numerical tests and comments. Section **5** concludes this chapter.

3.2 UPPER AND LOWER BOUNDING STRATEGIES

Lower and upper bounding strategies are required so as to design an effective branch and bound algorithm and to accelerate the search of the optimal solution or to improve the quality of first solutions found in the branch and bound tree.

In Chapters III and IV, we have studied several heuristics and lower bounds for the HFS scheduling problem. Based on the conclusions made in these chapters, we hereafter present different upper and lower bounding strategies that can be used in our branch and bound algorithm.

UPPER BOUNDING STRATEGY. After several preliminary tests of the branch and bound algorithm the heuristics CDSG, CDSH, CDSV, SF_J, SFR_J and SBP_J were selected. At the root node, all these heuristics are used to compute the best possible upper bound. While at each node in order to come up with an upper bound only the single stage based heuristics (SF_J, SFR_J and SBP_J) are used. CDSG, CDSH and CDSV are permutation heuristics, where only the processing times are used to build Johnson's lists (see how these heuristics build an HFS schedule in Chapter III). However, the single stage based heuristics take into account the release dates and tails of the jobs at each stage which vary from node to node in our branching scheme, and therefore may consequently produce different solutions at different nodes.

SF heuristic (scheduling the HFS forward) is the least time consuming heuristic. Numerical tests have shown that, for a large number of problems, SF is dominated by, i.e. giving a higher makespan than, SFR and SBP. We therefore propose the following three strategies for computing an upper bound at each node.

1. *The first strategy chooses, a priori, one heuristic (among SF_J, SFR_J and SBP_J) and keeps using it at each node.*
2. *The second strategy selects the heuristic giving the best makespan at the root node, and uses it at each node. This strategy is based upon the assumption that each instance may have a special structure and one of the three heuristics, SF_J, SFR_J or SBP_J, might be more effective to cope with it.*
3. *The third strategy consists of choosing at each node the best schedule among the ones constructed by the three heuristics, SF_J, SFR_J and SBP_J.*

LOWER BOUNDING STRATEGY. At the root node, the Dual Heuristic Bound (DHB) is used to compute a lower bound. According to the conclusions made in Chapter IV, we can use the Set Bound (SB) or the heuristic for the subset bound (HSSB) to compute lower bounds at each node of the branch and bound tree.

The algorithm chooses a job to branch on in such a way that, after branching, the lower bound is maximized. Hence, we need to decide which lower bound to use (i) at the branching node as well as (ii) to appreciate the possible lower bound improvement obtained by each potential branching job. SB and HSSB seem to have different computational complexities leading to different qualities (see Chapter IV). Since a priori choice of one of these two bounding procedures is not obvious (quality versus time), we need to compare the performance of different lower bounding strategies. We hereafter present three of them.

1. *The first strategy uses HSSB (i) to evaluate each node as well as (ii) for choosing the branching job.*
2. *The second strategy uses SB (i) to evaluate each node and (ii) for choosing the branching job.*
3. *The third strategy uses HSSB (i) to evaluate each node and (ii) SB for choosing the branching job.*

These strategies account for the branch and bound dilemma. Should we spend more time on computing the bounds and on selecting the branching operation or should we spend more time on exploring many more nodes? The first strategy uses the subset bound HSSB with computation complexity of O(nM) for computing the single stage subproblem lower bound. It devotes much more time to computing a lower bound and finding a good branching job, i.e. giving a high increase in lower bound (see later Section 3.3), whereas the second one uses the Set Bound (SB) with complexity O(n) to compute the lower bound and finding a good branching job. Since HSSB may give a better bound than SB, it is interesting to test both strategies within a fixed running time. The first strategy devotes much more time to computing the bounds while the second one explores many more nodes. The third strategy is a mix of the first two. It uses HSSB to evaluate nodes in the tree (used once at each node) and SB for finding a good branching job, which is done roughly n times at each node and stage where n is the number of jobs.

The implementation of these lower bounding strategies is given in Subsection 3.4. We hereafter give our generalization of the interval method for the HFS problem.

3.3 THE BRANCHING SCHEME

In Chapter II, Section 4, we have presented the single stage relaxation of the HFS problem. In this branching scheme and at the root *node,* release dates and tails of all jobs at all stages are computed as for the single stage relaxation.

$$r_{is} = r_{i1} + \sum_{t=1}^{s-1} p_{it} \quad ; \qquad q_{is} = \sum_{t=s+1}^{K} p_{it} + q_{iK} \qquad \text{for all } i \in I,\ s \in \{1,..., K\} \qquad (5.18)$$

For node N of the branch and bound tree and each job i, let $r_{is}(N)$ and $q_{is}(N)$ be, respectively, the release date and tail of job i at stage s. Let Best_UB be the best known makespan so far. Note that since we already have an upper bound on the solution, in this branching scheme we are only interested in solutions better (strictly smaller) than Best_UB, so we can assign to each job i at each stage s a due date $d_{is}(N)$ as follows:

$$d_{is}(N) = \text{Best\_UB} - q_{is}(N) - 1, \quad i \in I, \quad s \in \{1, ..., K\} \qquad (5.19)$$

Equation (5.19) is easy to understand. The tail q_{is} is the minimum time a job has to spend in the shop between the completion time at stage s and the makespan. Hence, for any solution better than Best_UB, with makespan ≤ (Best_UB-1), job i has to be completed at stage s at time d_{is}=Best_UB-q_{is}-1 at the latest. Therefore, in order to meet the makespan (Best_UB – 1), job i at stage s can only start processing within the time interval STI_{is} = $[r_{is}(N)+1, d_{is}(N)-p_{is}+1]$.

Let $Margin_{is}(N)=d_{is}(N)-r_{is}(N)-p_{is}$ be the margin of job i where $Margin_{is}(N)+1$ represents the remaining flexibility in terms of possible number of start times for job i at stage s at node N. The branching scheme consists of choosing the branching node *N*, a branching stage t^* as well as a branching job j^*. The branching node *N* is selected among the active nodes as the one with the best (least) lower bound. At the end of this section and in Subsection 3.4 we present how t^* and j^* are selected.

As we have already noticed, because we do not have any information on which part of STI_{is} the starting time of i at stage s might take place in the optimal solution one appropriate strategy is to use *dichotomous search* and split STI_{is} into two equal parts every time job i is selected for branching.

When t* and j* are selected, the branching scheme replaces the node N problem by two subproblems N1 and N2 obtained by splitting the possible start dates of job j* at stage t* in two non overlapping sets. This is achieved by delaying the release date of job j^* at stage t^* at node N1 and bringing forward in time its due date at stage t^* at node N2.

The data (p_{is}, r_{is}, q_{is}, d_{is}, i∈I, s=1, ..., K) for both nodes N1 and N2 remain the same as for node N, except for the release dates r_{j^*s} and due dates d_{j^*s} of the branching job j^*, which are set as follows:

at node N1

$r_{j^*s}(N1) = r_{j^*s}(N)$	$s = 1, \ldots, t^*-1$
$r_{j^*t^*}(N1) = r_{j^*t^*}(N) + \lceil Margin_{j^*t^*} / 2 \rceil$	stage t^*
$r_{j^*s}(N1) = \max [r_{j^*s}(N) , r_{j^*s-1}(N1) + p_{j^*s-1}]$	$s = t^*+1, ..., K$
$d_{j^*s}(N1) = d_{j^*s}(N)$	$s = 1, ..., K$

at node N2

$d_{j^*s}(N2) = d_{j^*s}(N)$	$s = t^*+1, \ldots, K$
$d_{j^*t^*}(N2) = d_{j^*t^*}(N) - \lfloor Margin_{j^*t^*} / 2 \rfloor - 1$	stage t^*
$d_{j^*s}(N2) = \min [d_{j^*s}(N), d_{j^*s+1}(N2) - p_{j^*s+1}]$	$s = t^*-1, t^*-2, ..., 2, 1$
$r_{j^*s}(N2) = r_{j^*s}(N)$	$s = 1, ..., K$

Note that the due dates computation should be in the above given order, i.e. for $s=t^*+1, \ldots,K$, then for stage t^* and then for $s = t^*-1, t^*-2, ..., 2, 1$.

In other words, the starting time interval of job j^* at stage t* $[r_{j^*t^*}(N)+1, d_{j^*t^*}(N)-p_{j^*t^*}+1]$ is split into two non-overlapping parts:

At node N1, job j^* at stage t* can start in the interval:

$$[r_{j^*t^*}(N) + \lceil margin_{j^*t^*}/2 \rceil + 1,\ d_{j^*t^*}(N) - p_{j^*t^*} + 1]$$

At node N2 job j^* at stage t* can start in the interval:

$$[r_{j^*t^*}(N) + 1,\ d_{j^*t^*}(N) - \lfloor margin_{j^*t^*}/2 \rfloor - p_{j^*t^*}].$$

Furthermore, we can increase the release dates of j^* at the subsequent stages (s>t*), because its release date has increased at the branching stage t*. Similarly we can decrease its due dates at the preceding stages (s < t*) because its due date has been reduced at the branching stage t*. However, the release dates of j^* at the precedent stages (s<t*) can not be modified at node N1. In the same way, at node N2, the due dates of j* at the subsequent stages (s > t*) do not change.

Table 5.3 and Figures 5.6(a, b, c) depict an example of a 3-stage HFS problem with Best_UB(N)=40; $t^*=2$ and $Margin_{j^*2} = 15$.

	p_{j^*1}	p_{j^*2}	p_{j^*3}	r_{j^*1}	r_{j^*2}	r_{j^*3}	d_{j^*1}	d_{j^*2}	d_{j^*3}	q_{j^*1}	q_{j^*2}	q_{j^*3}
N	5	6	4	5	10	16	25	31	35	15	9	5
N1	5	6	4	*5*	***18***	***24***	25	31	35	15	9	5
N2	5	6	4	5	10	16	***17***	***23***	*35*	15	9	5

Table 5.3: Branching operation

A node N1, delaying the release date of j* at stage 2 by 8 units of time from $r_{j^*2} = 10$ to $r_{j^*2}=18$ does delay its release date at stage 3 by 8 units of time from $r_{j3} = 16$ to $r_{j3}=24$. However, this job can still start at any time between $r_{j^*1}+1 = 6$ and $(d_{j1}-p_{j1}+1) =21$, at stage 1. Doing so, the possible starting time intervals of job j* at stage 2 and stage 3 have shrunk, but did not change at stage 1. At node N2, the same reasoning can be made for the due dates. The possible starting time intervals of job j* at stages 1 and 2 have shrunk and did not change at stage 3.

Note also that at node N2, j* has to start at time $d_{j^*2}(N2)-p_{j^*2}+1=18$ at the latest so as to meet $d_{j^*2}(N2)=23$ and that at node N1, job j* starts at time $r_{j^*2}(N1)+1=19$ at the earliest, which make N1 and N2 representing two disjoint subproblems of node N.

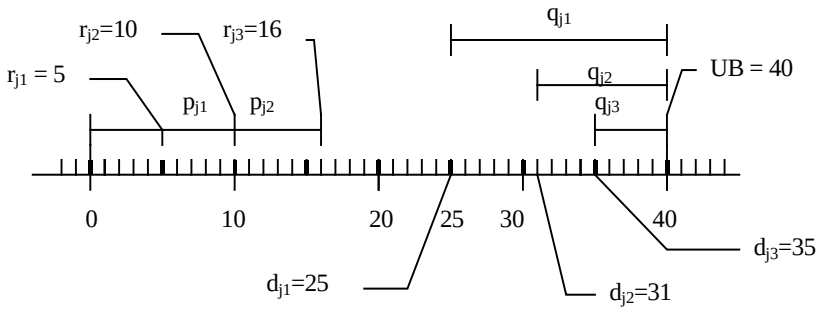

Figure 5.6: (a) Release dates and due dates at all stages of the branching job at node N

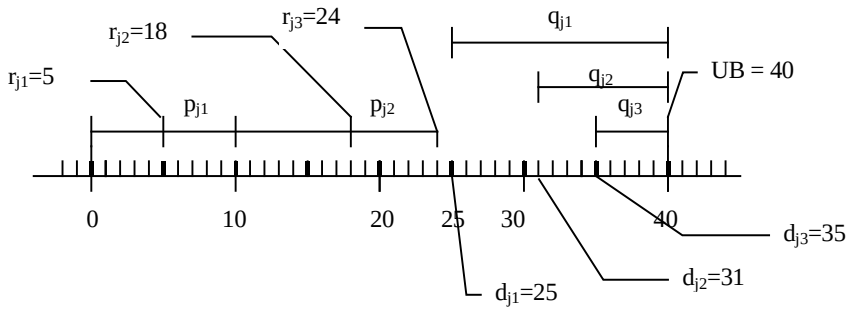

Figure 5.6: (b) Release dates and due dates at all stages of the branching job at node N1 stage 2 is the branching stage

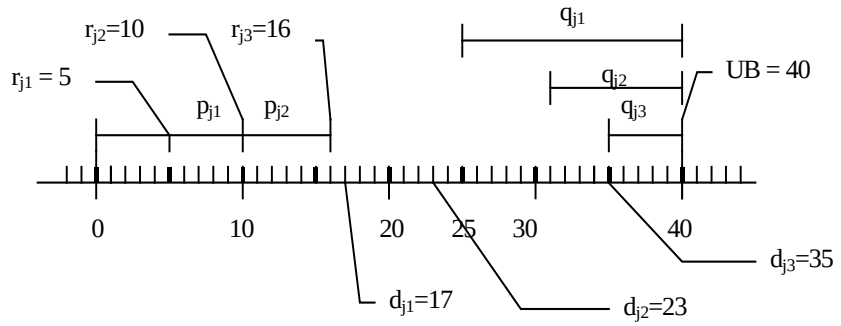

Figure 5.6: (c) Release dates and due dates at all stages of the branching job at node N2 stage 2 is the branching stage

PREPROCESSING OF DUE DATES AND TAILS

Each time a node N is selected for branching from the list of the outstanding nodes, and before computing a tighter lower bound, the tails and due dates of the jobs at node N can be improved as follows, see Carlier (1987):

$$q_{is} = \max \{q_{is}(N),\ LB(N) - d_{is}(N)\} \qquad \forall i \in I, s \in \{1, ..., K\} \qquad (5.20)$$

$$d_{is} = \min \{d_{is}(N),\ Best\_UB - 1 - q_{is}(N)\} \qquad \forall i \in I, s \in \{1, ..., K\} \qquad (5.21)$$

As LB(N) is a lower bound on the optimal makespan, when $q_{is} < LB(N) - d_{is}(N)$ we can always increase q_{is} in (5.20) to $q_{is} = LB(N) - d_{is}(N)$ because job i at stage s must be completed by time d_{is}. Equation (5.21) is similar to (5.19) and used to update d_{is} when a new tail q_{is} or/and a new upper bound have been found. Equations (5.20) and (5.21) take advantage of the improvement of the lower and upper bounds to tighten the problem solution space by reducing the due dates of the jobs.

In the preceding example, the job j* at stage 2 with a reduced due date at node N2 has its tail increased.

If LB(N2) = 36, $q_{j*2}(N2) = \max(q_{j*2}(N2), LB(N2) - d_{j*2}(N2)) = \max(9, 36 - 23) = 13$.

THE BRANCHING OPERATION

Let α_{jt}, β_{jt} be, respectively, the increase of the lower bound at N1 and N2 if we branch on job j at stage t, $\alpha_{jt} = LB(N1) - LB(N)$, $\beta_{jt} = LB(N2) - LB(N)$. The branching job j^* and stage t^* can be chosen so as to maximize the increase $(\alpha_{jt} + \beta_{jt})$.

$$(j^*, t^*) = \arg \max_{(j,t) s\in I^*\{1,...,K\}} (\alpha_{jt} + \beta_{jt}) \qquad (5.22)$$

Since this branching job and branching stage selection is time consuming (for each job we need to compute n*K lower bounds) we propose in the next section different branching job and branching stage selections depending on the lower bounding strategy used.

3.4 LOWER BOUNDING STRATEGIES IMPLEMENTATION

Since we have a choice among the three lower bounding strategies (proposed in Section 3.2) the computation time needed to compute the increase ($\alpha_{jt} + \beta_{jt}$) depends on the strategy we use. We hereafter recall the three lower bounding strategies and how they are implemented in our algorithms.

In the three strategies the stages are sorted in decreasing order of their corresponding lower bounds. The stages with smaller lower bounds are considered candidates for branching only if no job at the stage with higher lower bound has produced a strictly positive increase in lower bound. In other words, we always consider the stage with larger lower bound first, and choose the first stage that produces a positive increase of the lower bound, when branching on some job at that stage. This is done in order to save run time, instead of always evaluating (5.22) exactly.

As we compute at each node N the best lower bound ($SB_s(I)$ or $HSSB_s(I)$) over all possible stages s=1, ..., K, let r denote the stage giving the best bound.

$$(r) = \arg \max_{s\in\{1,...,K\}} SB_s(I) \text{ when using SB} \qquad (5.23)$$

$$(r) = \arg \max_{s\in\{1,...,K\}} HSSB_s(I) \text{ when using HSSB} \qquad (5.24)$$

Furthermore, let $U \subseteq I$ be the best subset of jobs found so far by HSSB at stage *r* such that $SB_r(U) = \max_{s\in\{1,...,K\}} HSSB_s(I)$.

Strategy 1. This strategy uses HSSB to evaluate each node and HSSB for selecting the branching job/stage.

Because LB(N) has already been computed when node N is selected for branching, r and U are known. So in order to select j* and t* we proceed as follows. We compute the increase in lower bound ($\alpha_{jr} + \beta_{jr}$) over only jobs in set *U* at stage *r* using HSSB. So the branching job j* is selected only among those in set U as the job maximizing ($\alpha_{jr}+ \beta_{jr}$) and stage r is taken as the branching stage, i.e t*=r.

If $\alpha_{jr} + \beta_{jr} = 0$ for all jobs in U, jobs in I/U at stage r become candidates for branching and the one with largest $\alpha_{jr} + \beta_{jr}$ is selected. If $\alpha_{jr} + \beta_{jr} = 0$ for all jobs in I, stage r is no longer chosen as the branching stage and all jobs at all stages $s \neq r$ become candidates for branching. Note that the other stages are considered in a non-increasing order of their corresponding lower bounds, i.e $HSSB_s(I)$.

Applying the HSSB firstly to set U and then to set I/U at stage r, before doing it over all stages might drastically reduce the computation time needed to find j^*. Also, stage r can be regarded as the bottleneck stage and hence branching on it might give a large increase in lower bound.

Strategy 2. This strategy uses SB to evaluate each node and SB for selecting the branching job/stage.

Here, only the stage r minimizing $SB_s(I)$ over all stages is selected when computing LB(N). So, the idea is the same as in the first strategy except that we use SB (instead of HSSB), and start by considering all jobs in set I at stage r as potential branching operation. If $\alpha_{jr} + \beta_{jr} = 0$ for all jobs in I we consider the other stages in a non-increasing order of their corresponding lower bounds, i.e $SB_s(I)$**.**

Strategy 3. This strategy uses HSSB to evaluate each node and SB for selecting the branching job/stage.

Here the idea remains the same as in the first strategy but we use the SB for selecting the branching job instead of the HSSB. So, we look first for an increase in lower bound by branching at stage r on jobs in set U, than on jobs in set I\U if $\alpha_{jr}+\beta_{jr}=0$ for all jobs in U. And, if $\alpha_{jr}+\beta_{jr} = 0$ for all jobs in I the other stages are considered in a non-increasing order of their corresponding lower bounds, i.e $SB_s(I)$**.**

Whatever the strategy, if no job has provided a strictly positive increase we then choose the job with the largest margin. Choosing the job with the largest margin does not provide a better lower bound at the newly created nodes N1 and N2. However, the probability of getting better lower bounds at nodes emanating from N1 and N2 is greater than the one if we branch on a job with smaller margin. Indeed, at node N, branching on a job with the largest margin leads to a larger increase in its release date at N1 and possibly in its tail at N2. Doing so for all jobs at different levels of the search tree, will lead to a large increase of the jobs release dates and/or tails, at some nodes, which will provide higher lower bounds. Remember that the lower bounds use the smallest release dates and the smallest tails.

Note also that only jobs with a strictly positive margin at stage s have the potential of increasing the lower bound. Jobs with null margin are simply ignored when selecting the branching job. Actually, these jobs have their starting times and completion times fixed according to the best known schedule.

SELECTING THE BRANCHING JOB/STAGE USING HSSB AND SB

In strategy **1** we use HSSB for selecting the branching job and stage, which means that we may need to compute HSSB for each potential job and stage, which is time consuming. In Section 3.5 we show how to approximate the increase in HSSB without recomputing it. In strategies **2** and **3** we use the set bound SB for selecting the branching job and stage. We recall how SB at stage s is computed.

$$SB(I)=\lceil (r_{[1s]} + r_{[2s]} + \ldots + r_{[Ms]} + q_{[1s]} + q_{[2s]} + \ldots + q_{[Ms]} + \sum_{j \in I} p_{js})/M_s \rceil$$

where $r_{[1s]}, r_{[2s]}, ..., r_{[Ms]}$ and $q_{[1s]}, q_{[2s]}, ..., q_{[Ms]}$ are the M_s smallest r_{js} and q_{js} over all jobs in I.

The increase in the numerator of SB, when branching on (j, t) can be computed as follows (see Carlier (1987)):

$$\alpha_{jt} = [\min \{ r_{[M_t+1]t}(N),\ r_{jt}(N1) \} - r_{jt}(N)]^+ \qquad (5.25)$$

$$\beta_{jt} = [\min \{ q_{[M_t+1]t}(N),\ q_{jt}' \} - q_{jt}(N)]^+ \qquad (5.26)$$

$$q_{jt}' = \max \{ q_{jt}(N), LB(N) - d_{jt}(N2) \}$$

Where $r_{[M_t+1]t}(N)$ and $q_{[M_t+1]t}(N)$ are the $(M_t+1)^{th}$ smallest release date and tail at stage t for the subproblem at node N. q_{jt}' is the new value of $q_{jt}(N)$ at node N2 computed using the new due date of job j at node 2, $d_{jt}(N2)$, see Equation (5.20).

For computing α_{jt} we check only the jobs with release dates smaller than $r_{[M_t+1]t}(N)$. In the same way for computing β_{jt} we check only the jobs with release dates smaller than $q_{[M_t+1]t}(N)$. Indeed, only such jobs have the potential of increasing the lower bound because the used set bound is computed using the M_t smallest release dates and M_t smallest tails. Therefore, only the increase of one of these release dates or tails might increase SB(I) and hence finding j* at some stage t when using SB is of $O(2M_t)$.

Recall that computing HSSB is of O(nM). Hence, finding the best branching job at some stage s is of $O(n^2M_s)$ when suing HSSB. We, therefore, propose in the next subsection how to approximate the potential increase in lower bound ($\alpha_{js}+\beta_{js}$) without recomputing HSSB.

3.5 COMPUTING THE POTENTIAL INCREASE OF THE SUBSET BOUND

In the first lower bounding strategy, for selecting the branching job and stage (j*, t*) we need to evaluate the increase of the lower bound, i.e. $\alpha_{jt}+\beta_{jt}$, for each potential job using HSSB. The idea behind HSSB is to find the subset $U \subseteq I$ producing the largest possible lower bound. We hereafter present a Search Procedure (SP), which seeks, with the subset U_t, or U for short, producing the lower bound at stage t at node N, whether branching on some job produces a positive increase of the lower bound. So we do no more need to compute HSSB for checking whether branching on some job might increase the lower bound.

The idea here is to check with the subset U giving the lower bound at node N, LB(N)=HSSB(I) = SB(U), whether or not branching on some job belonging to U or to I\U might increase the subset bound. Such Search Procedure (SP) should be less time consuming than computing HSSB so as to be more effective. Furthermore, SP needs to consider only those subsets of jobs having, potentially, larger lower bounds than HSSB(I). Numerical tests will be used to compare the use of SP to the use of HSSB for selecting the branching job/stage.

For a clear notation, we will ignore the symbol N in the following. Every time we refer to the set I or to the subset U, we assume that we refer to the set I and the subset U at stage t, at node N.

For checking whether some potential branching job i leads to a positive increase in lower bound ($\alpha_{it}+\beta_{it}$) we can define four operations depending on whether job i belongs or not to the subset bound U. The last two operations are carried out only when set I/U is not empty.

a) Branch on some job $i \in U$ and keep U (do not add or remove any job to or from U)
b) Branch on some job $i \in U$ and remove some job j from U with $j \neq i$
c) Branch on some job $i \in U$ and add some job $j \in I/U$ in U
d) Branch on some job $i \in I/U$ which is added in U

In the following we will show that operation (a), (b) and (d) may provide a positive increase of the lower bound, and prove that operation (c) never provides a positive increase of the lower bound and consequently is simply ignored.

In order to carry out these four operations, we maintain three sets of jobs:
- R(U) is the set of jobs where only the release dates are used in the computation of SB(U)
- Q(U) is the set of jobs where only the tails are used in the computation of SB(U)
- RQ(U) is the set of jobs where both release dates and tails are used in the computation of SB(U)

Let S = R(U) $\cup$ Q(U) $\cup$ RQ(U). For the following, these observations concerning the subset U at node N are needed.

Let γ(N) be the increase at stage t of HSSB(I) obtained by removing a job $j \in U$ from U. Also, let λ(N) be the increase at stage t of HSSB(I) obtained by adding a job $j \in I \backslash U$ in U. γ(N) and λ(N) are computed as follows:

$$\gamma(N) = \max\{0,\ r_{[M_t+1]}(N) - r_{jt}(N)\} + \max\{0, q_{[M_t+1]}(N) - q_{jt}(N)\} - p_{jt}(N)$$

$$\lambda(N) = p_{jt}(N) - \max\{0,\ r_{[M_t]}(N) - r_{jt}(N)\} - \max\{0,\ q_{[M_t]}(N) - q_{jt}(N)\}$$

Where $r_{[M_t+1]}$ and $q_{[M_t+1]}$, are the $(M_{t+1})^{th}$ release date and tail among jobs in U, respectively

Recall that the subset U is considered here as the optimal subset and removing a job j from U *at node N* does not provide a better lower bound than HSSB(I), (see how HSSB is computed in Chapter IV, Section 8), hence $\gamma(N) \leq 0$. Also, adding a job $i \in I/U$ in U *at node N* does not provide a better lower bound than HSSB(I), hence $\lambda(N) \leq 0$. These two observations ($\gamma(N) \leq 0$ and $\lambda(N) \leq 0$) are used in the following.

a. *Branch on some job $i \in U$ and keep U (do not add or remove any job to or from U)*
The idea here is to check, without modifying the set U, if branching on some job $i \in U$ leads to a positive increase of the lower bound. This operation is the same as the one when using the set bound SB but here we use the subset U instead of I. $\alpha_{it}(a)$ and $\beta_{it}(a)$

are the increases giving by operation a if we branch on job $i \in U$ at stage t. A maximum of $2*M_t$ jobs are considered in the checking process, i.e. jobs in set S.

$$\alpha_{it}(a) = \min \{ r_{[M_t+1]}(N),\ r_{it}(N1) \} - r_{it}(N) \qquad (5.24)$$

$$\beta_{it}(a) = \min \{ q_{[M_t+1]}(N),\ q_{it}' \} - q_{it}(N) \qquad (5.25)$$

where $q_{it}' = \max \{ q_{it}(N), LB(N) - d_{it}(N2) \}$

Example 5.2(a) and data in Table 5.4(a) show that when branching on job $3 \in U$, the lower bound at node N1 increases and the subset U at node N1 remains the same as at node N, i.e. U(N)=U(N1).

Example 5.2(a):
U(N)={1, 3, 4, 5}; HSSB(I) = SB(U(N))=$\lceil [(2 + 7) + (8 + 8) + 55] / 2 \rceil = 40$;
$r_{[M_t+1]}(N) = 8$; $q_{[M_t+1]}(N) = 12$; Let Best_UB = 44, $d_3 = 32$
$margin_3 = 16 \rightarrow r_3(N1) = 2 + 8 = 10$; $d_3(N2) = 32–8–1= 23$; $q_3' = 40–23=17$
$\alpha_{3t}(a) = \min \{8, 10\} - 2 = 6$; $\beta_{3t}(a) = \min \{12, 17\} - 12 = 0$
And, HSSB(N1) *2 = $\lceil [(15 + 14 + 6 + 20) + (10 + 7) + (8 + 8)]/ 2 \rceil = 43$

i	r_i	p_i	q_i
1	12	15	12
2	2	8	3
3	2	14	12
4	8	6	8
5	7	20	8

Table 5.4: (a) A two-machine problem data at node N, processing SP (a)

b. *Branch on some job $i \in U$ and remove some job j from U with $j \neq i$*
Because of the increase of $r_{jt}(N)$ and the possible increase of $q_{jt}(N)$ at nodes N1 and N2 respectively, the subset U at N1 and N2 might no longer be the optimal subset. And, removing some job j from U might lead to a positive increase of the lower bound. In this operation, two parts compose the increase of the lower bound. The first part is given when branching on $i \in U$ (see above operation a). The second part comes from the removal of some job $j \in U$, $j \neq i$. Let $r_{[M_t+1]}'(N1)$ and $q_{[M_t+1]}'(N2)$ be the new values of $r_{[M_t+1]}(N)$ and $q_{[M_t+1]}(N)$, respectively, obtained from the modification of $r_{it}(N)$ and $q_{it}(N)$ when branching on job i. The total increase is computed as follows:

$$\alpha_{it}(b) = \alpha_{it}(a) + \max \{0,\ r_{[M_t+1]}'(N1) - r_{jt}(N)\} + \max \{0, q_{[M_t+1]}(N) - q_{jt}(N)\} - p_{jt}(N)$$

$$\beta_{it}(b) = \beta_{it}(a) + \max \{0,\ r_{[M_t+1]}(N) - r_{jt}(N)\} + \max \{0,\ q_{[M_t+1]}'(N2) - q_{jt}(N)\} - p_{jt}(N)$$

Example 5.2(b) using the data from Table 5.4(b) shows that when branching on job $3 \in U$ and job 4 is removed from the subset U the lower bound, at node N1 increases by 4. Table 5.4(b) also gives the detail calculations before and after modifying the release date and tail of job 3 as well as after removing job 4, at node N1 and at node N2. Note that $r_{[M_t+1]}(N)$ has changed at node N1.

		i	$p_i(N)$	$r_i(N)$	$q_i(N)$	$r_i(N1)$	$q_i(N2)$
		1	*15*	*12*	*12*	*12*	*12*
		3	*16*	*2*	*12*	***12***	***13***
		~~4~~	~~4~~	~~8~~	~~8~~	~~8~~	~~8~~
		5	*20*	*7*	*8*	*7*	*8*
U={1,3,4,5}	$SB(N)*M_t$ =	80 =	55+	2+7	8+8		
	$SB(N1)*M_t$ =	86 =	55+		8+8	8+7	
	$SB(N2)*M_t$ =	80 =	55+	2+7			8+8
U={1,3,5}	*$SB(N)*M_t$ =*	*80 =*	*51+*	*2+7*	*12+8*		
	$SB(N1)*M_t$ =	*90 =*	*51+*		*12+8*	*12+7*	
	$SB(N2)*M_t$ =	*80 =*	*51+*	*2+7*			*12+8*

Table 5.4: (b) A two-machine problem data at node N, processing SP (b)

Example 5.2(b):

U(N)={1, 3, 4, 5}; HSSB(I) = SB(U(N)) = 40;

Let Best_UB = 50, $margin_3 = 20 \rightarrow r_3(N1)=12$; $d_3(N2) = 27$; $q_3' = 40-27=13$;

$r_{[M_t+1]} = 8$; $r_{[M_t+1]}'(N1) = 12$; $q_{[M_t+1]}(N) = q_{[M_t+1]}'(N2) = 12$;

$\alpha_{3t}(a) = \min\{8, 12\} - 2 = 6$; $\beta_{3t}(a) = \min\{12, 13\} - 12 = 0$

$\alpha_{3t}(b) = \alpha_{3t}(a) + \max\{0, 12 - 8\} + \max\{0, 12 - 8\} - 4 = 10$

$\beta_{3t}(b) = \beta_{3t}(a) + \max\{0, 8 - 8\} + \max\{0, 12 - 8\} - 4 = 0$

c. *Branch on some job $i \in U$ and add some job $j \in I/U$ in U*

Because of the increase of $r_{jt}(N)$ and the possible increase of $q_{jt}(N)$ at nodes N1 and N2, respectively, $r_{[M_t]}$ and $q_{[M_t]}$ might change.

Proposition 5.4: Adding some job $j \in I/U$ in U after branching on some job i in U always gives a negative increase.

Proof. Let $r_{[M_t]}'(N1)$ be the new value of $r_{[M_t]}(N)$ obtained at node N1 from the modifications of $r_{it}(N)$.

Note that $r_{[M_t]}' \geq r_{[M_t]}$ only if $r_{it}(N1) \geq r_{[M_t]}(N)$, otherwise $r_{[M_t]}' = r_{[M_t]}$, hence $r_{[M_t]}' \geq r_{[M_t]}$. Let $\lambda(N)$ and $\lambda(N1)$ be, respectively, the increase gained by adding a job $j \in I/U$ in set U at node N and node N1.

$\lambda(N) = p_{jt}(N) - \max\{0, r_{[M_t]}(N) - r_{jt}(N)\} - \max\{0, q_{[M_t]}(N) - q_{jt}(N)\} \leq 0$

$\lambda(N1) = p_{jt}(N) - \max\{0, r_{[M_t]}'(N) - r_{jt}(N)\} - \max\{0, q_{[M_t]}(N) - q_{jt}(N)\}$

We can see that $\lambda(N1) > \lambda(N)$ only if $r_{[M_t]}' < r_{[M_t]}$ which never happens.

The proof is the same when considering tails instead of release dates.

d. *Branch on some job $i \in I/U$ which is added in U*

Branching on some job $i \in I/U$ increases $r_{it}(N)$ and possibly increases $q_{it}(N)$ at node N1 and N2, respectively. And, reintroducing this job i in the subset U, at N1 and N2 might increase the lower bound HSSB(I), i.e. $SB(U \cup \{j\})$ might be larger than SB(U). This operation is the same as *the improvement step* presented in Chapter IV, Section 8.5.

The increase produced by this operation is computed as follows:

$$\alpha_{it}(d) = p_{it}(N) - \max\{0,\ r_{[M_t]} - r_{it}(N1)\} - \max\{0,\ q_{[M_t]} - q_{it}(N)\}$$
$$\beta_{it}(d) = p_{it}(N) - \max\{0,\ r_{[M_t]} - r_{it}(N)\} - \max\{0,\ q_{[M_t]} - q_{it}'\}$$

Example 5.3 shows that when branching on job $4 \in I/U$, the lower bound, at node N1, increases when adding this job to the subset U, i.e. $U(N1) = U(N) \cup \{4\}$.

Example 5.3: $U(N) = \{1, 3, 5\}$; $HSSB(I) = SB(U(N)) = 40$

i	r_i	p_i	q_i
1	10	15	12
2	2	8	3
3	8	14	8
4	2	6	6
5	7	20	8

Table 5.5: A two-machine problem data at node N, processing SP (c)

Let Best_UB = 42; $d_4(N) = 36$; $r_{[M_t]} = 8$, $q_{[M_t]} = 8$;
$margin_4 = 28 \rightarrow r_4(N1) = 16$; $d_4(N2) = 21$; $q_4' = 19$;
$\alpha_{it} = 6 - \max\{0, 8 - 16\} - \max\{0, 8 - 6\} = 4$
$\beta_{it} = 6 - \max\{0, 8 - 2\} - \max\{0, 8 - 19\} = 0$
$U(N1) = U(N) \cup \{4\} = \{1, 3, 4, 5\}$;
$HSSB(N1) = [(15 + 14 + 6 + 20) + (8 + 7) + (8 + 6)] / 2 = 42$

Proposition 5.5: (i) if $r_{it}(N) \geq r_{[M_t]}$ then $\alpha_{it}(d) = \lambda(N) \leq 0$
(ii) if $q_{it}(N) \geq q_{[M_t]}(N)$ then $\beta_{it}(d) = \lambda(N) \leq 0$

Proof. (i) Let $\lambda(N)$ be the increase gained by adding a job $j \in I/U$ in set U at node N.
$\lambda(N) = p_{it}(N) - \max\{0,\ r_{[M_t]} - r_{it}(N)\} - \max\{0,\ q_{[M_t]} - q_{it}(N)\}$
$\alpha_{it}(d) = p_{it}(N) - \max\{0,\ r_{[M_t]} - r_{it}(N1)\} - \max\{0,\ q_{[M_t]} - q_{it}(N)\}$
We know that $\lambda(N) \leq 0$
if $r_{it}(N) \geq r_{[M_t]}$ then $p_{it}(N) - \max\{0,\ q_{[M_t]} - q_{it}(N)\} \leq 0$
and $\alpha_{it}(d) = p_{it}(N) - \max\{0,\ q_{[M_t]} - q_{it}(N)\}$ because $r_{it}(N1) > r_{it}(N) \geq r_{[M_t]}$
Hence $\alpha_{it}(d) = \lambda(N) \leq 0$ if $r_{it}(N) \geq r_{[M_t]}$
(ii) The proof is the same as for (i) by considering tails instead of release dates.

We now give the final algorithm.

3.6 THE ALGORITHM

Read the data of the HFS problem;
Best_UB = The best makespan; {CDSG, CDSH, CDSV, SF, SFR and SBP}
Compute r_{is}, q_{is}, *and* d_{is} $i \in I$, $s=1, ..., K$; {equations (5.18) and (5.19)}
LB(Root) = HSSB(I)
Add Root node to List-Of-Node;
While List-Of-Node is not empty
    Remove node N with LB(N) minimal from List-Of-Node;
    If LB(N) $\geq$ *Best_UB* ➔ *Erase all the nodes in List-Of-Node;*
    Adjust q_{is} *and* d_{is} $\forall i \in I$, $s=1, ..., K$; {equations (20) and (21)}
    UB(N) = The best makespan; { SF_J, SFR_J and SBP_J }
    If UB(N) < Best_UB ➔ *Best_UB = UB(N);*
    Compute a better lower bound LB(N);
    If LB(N) < Best_UB ➔
        Choose j^* *and* t^*;
        Introduce N1 and N2 ;
        Compute LB(N1) and LB(N2);
        Add N1 and N2 to List-Of-Node;
    End-if;
End-while;

4. NUMERICAL TESTS

By these numerical tests we are first interested in comparing the improved Sequence Enumeration branch and bound algorithm (SE) (presented in Section 2) with the Interval Method algorithm (IM). We are also interested in comparing different versions of the interval method algorithm (presented in Section 3) using different lower and upper bounding strategies.

Root node upper and lower bounds. For both algorithms (SE and IM), at the root node we retain the best upper bound provided by CDSG, CDSH, CDSV, SF, SFR and SBP (selected in Chapter III as the best heuristics). Also, for both algorithms (SE and IM), at the root we compute the best possible lower bound using the Dual Heuristic based lower Bound (DHB) (selected in Chapter IV as the best lower bound).

For the sequence enumeration method, we use the algorithm proposed in Section 2 along with the proposed lower and upper bounding strategies (used at each node of the search tree). These upper bounding strategies were selected from different versions as the best-suited ones for the sequence enumeration algorithm. *For the interval method*, we hereafter recall the lower and upper bounding strategies presented in Subsection 3.2 and Subsection 3.4.

Interval method upper bounding strategies
Strategy 1 chooses a priori heuristic among SF_J, SFR_J and SBP_J and keeps using it at each node.

Strategy 2 selects the heuristic giving the best makespan at the root node and uses it at each node.
Strategy 3 chooses at each node the best schedule among the ones constructed by the three heuristics, SF_J, SFR_J and SBP_J.

Interval method lower bounding strategies
Strategy 1 uses HSSB to evaluate each node and for choosing the branching job/stage
Strategy 2 uses SB to evaluate each node and for choosing the branching job/stage
Strategy 3 uses HSSB to evaluate each node and SB for choosing the branching job/stage

Also, recall that for selecting the branching job/stage in the case of the first lower bounding strategy we suggested in Subsection 3.5 a Search Procedure (SP) to approximate the increase of the lower bound for each potential branching job instead of using HSSB.

Preliminary tests
For the interval method and during the preliminary tests, which we do not report here, we have noticed the following remarks and have accordingly taken some decisions for final test purposes.

- The third upper bounding strategy always gives better solutions in a fixed running time although the number of the explored nodes had been reduced. This result was expected. Indeed, the efficiency of the interval method relies heavily on the quality of the upper bounds found. Every time a better upper bound is obtained, the starting time intervals of all jobs are tightened, reducing the space of solutions that still needs to be explored. Computing the best upper bound at each node accelerates then the convergence of the algorithm. Note also that the used heuristics were selected as the best and least time consuming ones as well as complimentary heuristics. See Chapter 3, Section 9. We therefore decided to use the third upper bounding strategy for all the tests reported here.
- As expected, in a fixed running time using the Search Procedure (SP), defined in Subsection 3.5, for selecting the branching job and stage provides better solutions than HSSB. So we decided to use SP for selecting the branching job/stage at the first lower bounding strategy.

4.1 TESTED ALGORITHMS AND CONFIGURATIONS

In the following we compare four algorithms. The improved Sequence Enumeration algorithm (SE) (presented in Section 2) and three versions of the Interval Method (IM) (presented in Section 3). Each version of the interval method uses one of the three above recalled lower bounding strategies, i.e. IM_1, IM_2 and IM_3, where IM_i stands for the interval method branch and bound algorithm using lower bounding strategy i. Remember that all algorithms IM_i, i=1, 2, 3 use the third upper bounding strategy, that is choosing at each node the best upper bound among the ones given by the three heuristics SF_J, SFR_J and SBP_J (presented in Chapter III).

We have carried out tests on both algorithms (SE and IM) on a Pentium 350MHz with 64 MB RAM. All algorithms have been programmed using an object-oriented language

(C++) and run under Windows NT 4.0. The processing times, release dates and tails were generated uniformly ($p_{is} \in [1, 100]$, $r_{i1} \in [1, 100]$ and $q_{iK} \in [1, 100]$). The release dates and tails were generated for the first and the last stage, respectively. Configurations with and without release dates and tails as well as with and without a bottleneck stage, have been considered.

Bottleneck versus non-bottleneck configurations

Configurations with the same number of machines at each stage are considered non-bottleneck configurations, $M_s=M$ for s=1, …, K. For generating bottleneck configurations we maintain the same number of machines at all stages except at some stage t where the number of machines is reduced by 1 to create an artificial bottleneck stage, $M_s=M$ for all $s \neq t$ and $M_t = M-1$. As it has been noticed in Chapter III, the smaller is M_t (compared to M) the larger is the relative capacity reduction at the bottleneck stage t and the easiest is the HFS corresponding problem, i.e. our algorithms (heuristics and branch and bound algorithms) find good solutions. This observation has been noticed in Chapter III and in our numerical tests here. Also, combining all types of configurations in a single experimentation does not reflect the real performance of the algorithm with respect to hard problems, i.e. the non-bottleneck problems.

Three classes of configurations are considered. The first class **C1** (Table 5.6) is composed of non-bottleneck configurations with jobs having zero initial release dates and zero final tails. The second class **C2** (Table 5.7) is composed of non-bottleneck configurations with jobs having nonzero initial release dates and nonzero final tails. The third class **C3** (Table 5.8) is composed of bottleneck configurations with jobs having nonzero initial release dates and nonzero final tails.

In each class, different HFS configurations, in terms of number of stages, number of machines and number of jobs, have been designed. For each configuration, 20 instances have been tested. All the branch and bound run have been stopped after a fixed running time of 10 minutes.

Tables 5.6, 5.7 and 5.8 (*Given at the end of this chapter*) present the numerical tests we have carried out. Column n gives the number of jobs. Columns Ms (s = 1, ..., 5) give the number of parallel machines at stage s.

The percentage of the average deviation from the best lower bound found

The first set of columns in Tables 5.6, 5.7 and 5.8 gives for the root node (RootUB) and for each algorithm tested (SE, IM_1, IM_2 and IM_3) the percentage (over the 20 tested instances) of the average deviation from the best lower bound found so far. Actually, each time a branch bound algorithm stops either because the branch and bound finishes or because it has spent 10 minutes running, we retain the last lower bound found. For each tested instance, the best lower bound found over the four tested algorithms is used to compute the average deviations presented in Tables 5.6, 5.7 and 5.8. For example in Table 5.6 we can see that for the configuration with 30 jobs and 5 stages with 5 machines per stage, the four algorithms on average start at the root node with an average deviation of the upper bound from the lower bound (*the initial gap*) equal to 16.2%. And, after 10 minutes running time, this gap has been reduced by SE to 14.3%, however it has been reduced to about 9% by the other algorithms, (IM_i,i=1, 2, 3).

The percentage of the optimal solutions found

The second set of columns in Tables 5.6, 5.7 and 5.8 gives the percentage of the optimal solutions found by each algorithm, i.e. branch and bound finished. For example in Table 5.6, we can see that for the configuration with 30 jobs and 5 stages with 5 machines per stage, for the 20 tested instances SE has never found the optimal solution. However, for 55% of tested instances IM_1 has found the optimal solution, versus 30% for IM_2 and 50% for IM_3.

Table 5.6 **(C1)** compares non-bottleneck configurations with zero initial release dates and zero final tails, i.e. $r_{j1}=0$, $q_{jK}=0$; $j \in I$. Table 5.7 **(C2)** compares non-bottleneck configurations with initial release dates and final tails strictly positive, i.e. $r_{j1}>0$, $q_{jK}>0$; $j \in I$. Table 5.8 **(C3)** compares bottleneck configurations with $r_{j1}>0$ and $q_{jK}>0$; $j \in I$.

4.2 RESULT ANALYSIS

COMPARING THE SEQUENCE ENUMERATION WITH THE INTERVAL METHOD

Table 5.9 summarizes Tables 5.6 (C1), 5.7(C2) and 5.8(C3) (Rows C1, C2 and C3, respectively, corresponds to 320, 320 and 600 tested problems), Row C1 responds to 320 problems and . The numerical tests show, after a few minutes of running time (10 minutes), that even for the hardest (i.e. without a bottleneck stage) and mid-size problems (i.e. 15 to 30 jobs), the branch and bound algorithms on average have improved effectively the initial solution. Indeed, the algorithms have reduced the gap between the best known upper bound at the root node and the best known lower bound at the end of the algorithms.

	% of the average deviation from the best LB					% of the optimal (proven) solutions found			
	ROOTUB	SE	IM_1	IM_2	IM_3	SE	IM_1	IM_2	IM_3
C1	***10.6***	***7.3***	***3.9***	***4.1***	***4.0***	***23***	***63***	***43***	***65***
C2	***10,3***	***7,2***	***3,9***	***4,2***	***4,0***	***20***	***66***	***40***	***65***
C3	***7.3***	***5.0***	***2.3***	***2.4***	***2.2***	***20***	***62***	***49***	***73***

Table 5.9: Summary of Tables 5.6, 5.7 and 5.8

For non-bottleneck configurations (C1 and C2) the gap has been reduced by 31% (from 10.6% to 7.3%) by the sequence enumeration (SE) method and by about 63% (from 10.6% to 4%) by the interval method (IM_1). For the bottleneck configurations (C3) the gap has been reduced by only 31% (from 7.3% to 5%) by the sequence enumeration (SE) method and by about more than 69% (from 7.3% to 2,2%) by the interval method (IM_3).

In the case of C1 and C2 both algorithms (SE and IM) start with a higher gap (about 10%) than in the case of C3 (about 7%). In the former case, the interval method converges more rapidly than in the latter case. For IM and for C1 and C2 the gap has shrunk by about 70% versus 63% for C3. However, for the sequence enumeration method the convergence seems to be the same for C1, C2 and C3.

Note that the Interval Method (IM) outperforms the Sequence Enumeration (SE) method. Indeed, in general, IM converges rapidly, i.e. its gap reduction is twice the gap reduction of SE. Also, IM provides three times the number of the optimal solutions provided by

SE. For C3, for 73% of the tested instances IM_3 has found the optimal solution versus 20% for SE.

Also, we can see that IM is more robust than SE in the sense that IM finds a large number of optimal solutions even for HFS problems with large sizes. For instances, in the case of the 5-stage configurations, IM_1 (see Table 5.7) has found the optimal solution for 65% of tested instances versus 2% for SE.

COMPARING THE DIFFERENT BOUNDING STRATEGIES IN THE INTERVAL METHOD

In general, when comparing the gap (between the root upper bound and the best known lower bound) reduction, no major differences can be noticed among the three lower bounding strategies (less than 1%). However, when we compare the percentage of the optimal solutions found, IM_2 seems to be outperformed by the two other lower bounding strategies. On average, IM_1 finds 20 to 30% more optimal solutions than IM_2. For the bottleneck configurations, IM_3 finds 17% more optimal solutions than IM_1.

Based on these observations we conclude that the second lower bound strategy (using the set bound to evaluate the node and to select the branching job/stage) using a less tight and time consuming lower bound does not perform well compared to the other strategies. And hence, the use of tighter lower bound (HSSB) although more time consuming provides better solutions. The time spent in computing lower bounds has largely been compensated by the reduction of the number of nodes to explore.

Also, since IM_1 and IM_3, which differ by the method used for the selection of the branching job/stage, provide almost the same results, we can conclude that the selection of the branching job/stage is very important in the interval method. Indeed, if this were not the case, IM_3, which spends less time in finding the good branching job/stage, would provide better results than IM_1, because the saved running time would permit us to explore more nodes and hence would allow IM_3 to find better solutions. Here also we conclude that the use of tighter lower bounds is important to the interval method.

4.3 NUMERICAL TEST CONCLUSIONS

The interval method outperforms the sequence enumeration method. The sequence enumeration method, which has largely been used in other scheduling contexts (flowshop, parallel machines, see Bratley et al., 1975, Townsend W., (1977) and Pinedo 1995) does not perform well in the HFS context, i.e. it has a higher running time complexity in the HFS case. This result was expected because of the combinatorial explosion of the number of the sequences, which need to be enumerated in the case of the multistage flowshop with parallel machines, i.e. HFS.

The generalization of the interval method to the HFS problem provides good solutions when compared to the sequence enumeration method. Also, the interval method is more robust than the sequence enumeration method. It gives a large number of optimal solutions even for HFS problems with large sizes (15 to 30 jobs and 3 to 5 stages).

The interval method is more flexible than the sequence enumeration method in the sense that we were able to better fine tune the final algorithm. Also, the branching operation in the interval method is more powerful than the sequence enumeration one's in the sense that splitting in two parts the possible starting time of a job, modifies strongly the structure of the subproblem or node. In the case of the sequence enumeration method, fixing the position of a job in the sequence, at some node, creates a large number of new nodes to evaluate and explore, and consists simply in checking one solution within the large space of solutions.

For the interval method one should pay careful attention to the lower bounding strategy used. Based on the comparison of the three tested lower bounding strategies, we conclude that tighter lower bounds with improved running time are required for better solving the HFS scheduling problem when using branch and bound algorithms based on the interval method.

5. CONCLUSION

In this chapter we have studied the branch and bound technique for solving the hybrid flowshop scheduling problem. We tackled the makespan minimization case. Two branching schemes were studied. The Sequence Enumeration (SE) method proposed by Brah and Hunsucker (1991) for solving the HFS problem and the generalization of the interval method introduced by Carlier (1987) for solving the parallel machine problem with release dates and tails.

Based on the conclusions made in Chapter III and IV concerning upper and lower bounds we designed two branch and bound algorithms. We first proposed several improvements to Brah's algorithm and showed by numerical tests that the new obtained algorithm outperforms the original one. We then generalized the interval method to the HFS context and proposed several bounding strategies.

The new sequence enumeration algorithm and the interval method algorithm along with the different bounding strategies have been tested on different HFS configurations with reasonable sizes (15 to 30 jobs) compared to those found in the literature. Based on these numerical tests we concluded that:

- The complexity of the sequence enumeration method in the HFS context is very high and largely outperformed by the interval method.
- The interval method is more robust than the sequence enumeration method. It gives a large number of optimal solutions even for HFS problems with large sizes (15 to 30 jobs and 3 and 5 stages).
- After several minutes of running time (viz., 10 minutes) the interval method has reduced the gap between the root upper bound and the best-known lower bound by more the 60%.
- For the interval method, the use of tighter lower bound (HSSB) though more time consuming provides better solutions. One then should pay careful attention to the lower bounding strategy used.

- Tighter lower bounds with improved running time are then required for better solving the HFS scheduling problem when using the interval method.

Therefore, based on the conclusions made in Chapter III and Chapter IV dealing with upper and lower bounds, and based on the study carried out in this chapter, we designed a new branching scheme to deal with the HFS scheduling problem. This algorithm seems to be a promising method that may be enhanced in further investigations.

Extensive study and design of several scheduling algorithms in this project has led us to implement a large variety of scheduling algorithms. For efficiency reasons and in order to reduce the complexity and time needed for computer implementation, object oriented modeling has been used so as to design a global Object Model for Scheduling Algorithm Implementations (OMSAI). This model is presented in the appendix of this thesis.

REFERENCES

1. Brah, S. A., J.L. Hunsucker, (1991). Branch and bound algorithm for the flow shop with multiprocessors, European Journal of Operational Research, 51, 88-89, 1991.
2. Bratley, P., M. Florian, P. Robillard, (1975). Scheduling with earliest start and due date constraints on multiple machines, Naval Research Logistics Quaterly, vol. 22, n°1, 165-173, 1975.
3. Carlier, J., (1987). Scheduling jobs with release dates and tails on identical machines to minimize the Makespan, European Journal of Operations Research, 29, 298-306.
4. Guinet, A., (1996a). Integrated and Sequential Approaches to schedule Hybrid Flowshops in make to stock Environment, Workshop on production planning and control (Proceedings), FUCAM (Belgium), 299-302, September 1996.
5. Guinet, A., M. Solomon, (1996b). Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time, Int. J. Prod. Res., 34, 1643-1654.
6. Hoovegen, J. A., J. K. Lenstra, B. Veltman, (1994). Minimizing makespan in a multiprocessor flowshop is strongly NP-Hard, European Journal of Operations Research to appear.
7. Moursli, O., Y. Pochet, (1996). Heuristics for scheduling a Multiprocessor flowshop. Advances in Industrial Engineering Applications and Practice I. Houston, Texas. Conference proceeding 456-463.
8. Pinedo, M., (1995). Scheduling: Theory, Algorithms, and Systems, Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1995.
9. Townsend W., (1977). Sequencing n jobs on m machines to minimize tardiness: a branch and bound solution, Management Science, vol. 23, 9, 1016-1019, 1977.
10. Vandevelde, A., H. Hoogeveen, C. Hurkens, J. K. Lenstra, (1995). Lower bounds for the multiprocessor flow shop, Working paper, Eindhoven University of Technology, The Netherlands.
11. Vignier A., D. Dardilhac, D. Dezalay, C. Proust, (1996b). An optimal approach to solve a k-stage hybrid flowshop. Workshop on production planning and control (Proceedings), FUCAM (Belgium), 315-319, September.

						% of the average deviation from the best LB					% of the optimal (proven) solutions found			
n	M1	M2	M3	M4	M5	ROOTUB	SE	IM_1	IM_2	IM_3	SE	IM_1	IM_2	IM_3
6	2	2				2.6	0.0	0.0	0.0	0.0	100	100	95	100
8	2	2				5.5	0.0	0.1	0.1	0.1	100	75	80	75
10	2	2				5.0	1.8	0.8	1.0	1.0	40	80	40	80
15	3	3				10.6	7.7	2.0	1.7	2.3	0	65	40	60
20	4	4				10.8	7.6	2.8	3.3	2.2	10	35	40	65
30	5	5				10.7	8.8	3.2	3.2	3.1	0	55	35	65
8	2	2	2			8.4	3.8	2.6	2.8	2.7	40	60	50	60
10	2	2	2			8.6	4.4	3.2	3.2	3.4	25	50	60	55
15	3	3	3			12.2	8.6	4.9	5.9	4.9	5	55	25	65
20	4	4	4			11.2	9.0	4.3	5.1	4.2	0	45	15	80
30	5	5	5			13.8	11.2	5.5	5.6	5.5	0	55	30	60
8	2	2	2	2	2	10.6	6.6	4.0	4.2	4.0	25	70	45	65
10	2	2	2	2	2	13.3	9.4	6.3	6.5	7.4	20	75	45	50
15	3	3	3	3	3	14.9	11.9	7.0	7.4	7.4	0	60	25	50
20	4	4	4	4	4	15.3	12.1	7.0	7.4	7.1	0	75	40	55
30	5	5	5	5	5	16.2	14.3	8.9	9.0	8.9	0	55	30	50
Average						***10.6***	***7.3***	***3.9***	***4.1***	***4.0***	***23***	***63***	***43***	***65***

Table 5.6: (C1) Non-bottleneck configurations with zero release dates and tails.
Comparing the Interval Method (IM) with the improved Sequence Enumeration (SE) method, and comparing the interval method IM_i using different lower bounding strategies i=1, 2, 3.

						% of the average deviation from the best LB					% of the optimal (proven) solutions found			
n	M1	M2	M3	M4	M5	ROOTUB	SE	IM_1	IM_2	IM_3	SE	IM_1	IM_2	IM_3
6	2	2				4	0	0	0	0	100	100	60	95
8	2	2				4.5	0.1	0.1	0.3	0.2	100	100	80	90
10	2	2				5.0	1.8	0.8	1.0	1.0	40	80	40	80
15	3	3				5.2	3.4	0.3	0.3	0.1	0	75	80	95
20	4	4				7.9	5.1	1.4	1.7	0.9	5	40	20	85
30	5	5				5.8	4.5	1.3	1.2	0.6	20	45	45	90
8	2	2	2			9.5	4.9	3.4	3.8	3.7	35	65	45	45
10	2	2	2			9.3	4.6	2.3	2.3	2.6	10	65	55	55
15	3	3	3			11.7	8.4	3.8	3.9	4.3	0	70	25	45
20	4	4	4			11.2	9.0	4.3	5.1	4.2	0	45	15	80
30	5	5	5			11.6	9.1	4.7	4.6	4.3	0	40	40	65
8	2	2	2	2	2	14.4	10.2	5.1	5.8	6.4	5	80	40	35
10	2	2	2	2	2	14.7	12.6	8.2	8.5	8.3	5	70	30	35
15	3	3	3	3	3	16.9	14.2	8.3	8.7	8.7	0	50	15	45
20	4	4	4	4	4	15.3	12.1	7.0	7.4	7.1	0	75	40	55
30	5	5	5	5	5	19.4	16.0	11.4	11.9	11.5	0	50	15	40
Average						***10.3***	***7.2***	***3.9***	***4.2***	***4.0***	***20***	***66***	***40***	***65***

Table 5.7: (C2) Non-bottleneck configurations with nonzero initial release dates and nonzero final tails
Comparing the Interval Method (IM) with the improved Sequence Enumeration (SE) method, and comparing the interval method IM_i using different lower bounding strategies i=1, 2, 3.

						% of the average deviation from the best LB					% of the optimal (proven) solutions found			
n	M1	M2	M3	M4	M5	ROOTUB	SE	IM_1	IM_2	IM_3	SE	IM_1	IM_2	IM_3
10	1	2				0.8	0.0	0.0	0.0	0.0	100	100	100	100
10	2	1				2.0	0.8	0.3	0.3	0.3	65	95	85	100
15	2	3				4.9	3.4	0.7	0.8	1.0	5	75	45	60
15	3	2				6.6	4.6	0.5	0.5	0.5	0	65	30	80
20	3	4				7.5	4.8	1.2	1.3	0.8	0	45	30	60
20	4	3				6.7	4.8	1.2	1.5	1.3	0	70	10	65
30	4	5				8.2	6.6	1.9	2.1	1.5	0	20	45	65
30	5	4				6.4	4.3	2.1	2.3	1.9	5	50	20	55
10	1	2	2			2.5	1.4	0.6	0.6	0.7	55	90	90	90
10	2	1	2			2.8	0.7	0.2	0.2	0.3	75	100	85	95
10	2	2	1			4.9	1.7	0.6	0.6	0.5	50	85	85	100
15	2	3	3			5.5	3.2	0.8	0.9	0.8	5	70	35	65
15	3	2	3			6.5	4.7	1.7	1.7	1.6	0	60	40	70
15	3	3	2			8.0	5.1	2.0	2.3	2.2	0	75	25	60
20	3	4	4			10.9	6.8	2.1	2.2	1.9	0	45	25	60
20	4	3	4			10.9	8.7	4.5	5.1	4.1	5	50	30	65
20	4	4	2			4.1	1.9	0.5	0.6	0.3	0	65	40	90
30	4	5	5			11.5	8.9	3.2	3.1	2.5	0	20	45	65
30	5	4	5			11.0	9.1	5.0	5.3	4.5	5	30	20	90
30	5	5	4			8.0	6.3	3.2	3.4	2.9	0	45	40	55
8	1	2	2	2	2	5.8	2.8	1.6	1.6	1.3	50	75	85	85
8	2	2	1	2	2	4.9	2.7	1.7	1.8	1.7	60	90	60	90
10	1	2	2	2	2	4.9	2.3	0.8	0.8	0.7	50	85	85	95
10	2	2	1	2	2	4.6	2.6	1.3	1.5	1.3	50	80	75	85
15	2	3	3	3	3	7.7	5.4	2.8	3.0	3.0	0	40	55	55
15	3	3	2	3	3	10.5	7.7	3.9	4.0	3.7	0	35	45	60
15	3	3	3	3	2	10.3	7.7	4.0	4.4	4.8	15	55	40	45
20	3	4	4	4	4	14.1	11.1	7.6	7.6	7.7	5	35	35	55
20	4	4	3	4	4	13.7	10.7	6.7	7.3	6.2	0	55	25	65
20	4	4	4	4	3	12.5	9.5	5.8	5.7	5.6	5	45	45	70
Average						***7.3***	***5.0***	***2.3***	***2.4***	***2.2***	***20***	***62***	***49***	***73***

Table 5.8: (C3) Bottleneck configurations with non-zero initial release dates and nonzero final tails *Comparing the Interval Method (IM) with the improved Sequence Enumeration (SE) method, and comparing the interval method IM_i using different lower bounding strategies i=1, 2, 3.*