CHAPTER III
UPPER BOUNDS

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

LIST OF ACRONYMS

A1	Assignment rule 1
A2	Assignment rule 2
CCi	The i^{th} Class of HFS Configurations
CDS	Campbell, Dudek and Smith (1970) heuristic for the FS
CDSG	The adaptation of CDS idea to the HFS case by Guinet and Solomon (1996b)
CDSH	An adaptation of CDS idea to the HFS case proposed in this thesis
CDSV	An adaptation of CDS idea to the HFS case by Moursli (1997)
CH1	The first Class of Heuristics (Single stage based heuristics)
CH2	The second Class of Heuristics (Two-stage flowshop based heuristics)
CH3	The third Class of Heuristics (Priority rules based heuristics)
CH4	The fourth Class of Heuristics (Random scheduling)
CH5	The fifth Class of Heuristics (Local search techniques)
FS	Classical Flowshop
HFS	Hybrid Flow Shop
J-list	Johnson's list
LPT	Longest Processing Time dispatching rule
RS	A pure random scheduling heuristic (random sequencing and assignment)
RS1	A random scheduling heuristic using assignment rule A1
RS2	A random scheduling heuristic using assignment rule A2
SA	Simulated Annealing
SBP	Shifting Bottleneck Procedure
SBP_C	SBP using Carlier's branch and bound to solve the single stage subproblem
SBP_J	SBP using Jackson's heuristic to solve the single stage subproblem
SBR	Scheduling the HFS sequentially backward and rescheduling the critical stage
SBR_C	SBR using Carlier's BaB to solve the single stage subproblem
SBR_J	SBR using Jackson's heuristic to solve the single stage subproblem
SFR	Scheduling the HFS sequentially forward and rescheduling the critical stage
SFR_C	SFR using Carlier's branch and bound to solve the single stage subproblem
SFR_J	SFR using Jackson's heuristic to solve the single stage subproblem
SLST	Simple Local Search Technique
SPT	Shortest Processing Time dispatching rule
V1	The first variant of implementing priority rules for the HFS
V2	The second variant of implementing priority rules for the HFS
V3	The third variant of implementing priority rules for the HFS
V4	The fourth variant of implementing priority rules for the HFS

CHAPTER III
UPPER BOUNDS

Abstract

The hybrid flowshop (HFS) scheduling problem is NP-hard. Algorithms for finding optimal solutions in polynomial time are unlikely to exist. We therefore study heuristic solutions. The main objective is to design a set of heuristics that allow finding the best solutions possible. This chapter aims at studying a number of existing and new heuristics, by presenting and comparing them on different shop configurations. The studied heuristics use five different approaches. The first approach, by scheduling the HFS one stage at a time, uses different stage sequencing orders. The second and third approaches are mainly lists heuristics and use several policies to assign jobs to machines. To sequence the jobs, the second approach uses ideas derived from the multistage classical flowshop with a single machine per stage, while the third approach uses classical priority rules for jobshop scheduling problems. As opposed to the second and third approaches where specific sequencing and assignment rules are used, the fourth uses random scheduling so as to see whether elaborate scheduling solutions are worth developing or any randomly built solution suffices. The fifth approach uses local search techniques. For the latter, a neighborhood definition of an HFS schedule, based on its graph representation, is used for defining exchange procedures. All heuristics are tested on instances with different sizes in terms of number of jobs, machines and stages, and on several classes of shop configurations in terms of existence of initial release dates and final tails, and of bottleneck stages. Statistical analysis is carried out in order to compare the heuristics to one another and to eventually select the best subset of heuristics for each shop configuration.

1. INTRODUCTION

Minimizing the makespan of the hybrid flowshop (HFS) scheduling problem is NP-hard. Algorithms for finding optimal solutions in polynomial time are unlikely to exist. We therefore study heuristic solutions. The main objective is to design a set of heuristics that allow finding the best possible solutions. We presented, in Chapter II, a literature survey of the studied heuristics for the HFS problem. We also noticed that there are several ways for designing a heuristic, by using different stage sequencing orders and different job assignment and sequencing rules. In this chapter, five different heuristic classes using different scheduling approaches are considered. One of these approaches uses *random scheduling* and is used as a basis for comparing all the heuristics presented in this chapter so as to verify whether elaborate heuristics are worth developing or whether random heuristic should suffice to find good solutions. The heuristics using *random scheduling* are "*list heuristics*". Some of them will be tested with the non-random assignment rules used by the other *list heuristics.* Therefore, to simplify the presentation, these random heuristics are presented together with the other *list heuristics*. We present now the five approaches and their respective classes of heuristics.

- The first approach schedules the HFS a single stage at a time using different stage sequencing orders, and is derived from job shop solutions, see Adams et al. (1988).
- The second approach uses ideas derived from the classical multistage flowshop with a single machine per stage for sequencing the jobs, see Guinet and Solomon (1996b).
- The third one uses classical priority rules for sequencing the jobs and is derived from job shop solutions, see Hunsucker et al., (1989) and Hunsucker and Shah (1994).
- The fourth approach uses random scheduling.
- The fifth approach uses randomized local search techniques.

This chapter presents a number of existing and new heuristics and classifies them according to these five approaches. We report here on a comparative study of all the five classes of heuristics on the same instances in order to evaluate their performances, and to potentially allow the selection of the best set of heuristics for each problem configuration. We recall here briefly the main characteristics of these five classes of heuristics. Their detailed descriptions are given in subsequent sections.

- Our first class of heuristics uses *different stage sequencing orders*. It relaxes the HFS main problem into single stage subproblems with release dates and tails, and schedules the HFS one stage at a time using either a heuristic or an exact algorithm. Two sets of heuristics for building global HFS schedules are put forward. The first set schedules the HFS by giving a higher priority to bottleneck stages, i.e. shifting bottleneck procedure. The second set schedules the HFS sequentially starting by the first (the last) stage and going forward (backward) until the last (first) stage has been reached.
- Our second and third classes are mainly "*list heuristics*". They operate in two steps. The first step generates an ordered list of jobs (sequencing step). The second step schedules the HFS with respect to this list (assignment step). The second class of heuristics uses ideas derived from the two-stage classical flowshop heuristics so as to generate the ordered list, while the third one uses different priority-rules. The assignment of machines to the ordered list of the jobs is done by using several assignment rules for both classes of heuristics.
- As opposed to the second and third approaches where specific assignment and sequencing rules are used, the fourth approach uses several random sequencing and assignment rules. Three heuristics are designed. The first one is a pure random heuristic where job sequencing and machine assignment are done randomly. The second and the third heuristics build randomly an ordered list of jobs and, respectively, use two different assignment rules (the same as those used for the second and the third approaches).
- The fifth approach uses local search techniques. Two heuristics are designed. A simple local search technique where the search is done locally and randomly in the neighborhood of the seed solution (the current solution). The algorithm moves to a new solution only if the latter is better than the current one. The second heuristic uses simulated annealing technique. Here the algorithm accepts to move to a worse solution with some probability depending on some *cooling value*. The latter is reduced after the algorithm has explored a certain number of solutions.

Lower bounds are used to evaluate the heuristic performances and are based upon the relaxation of the main problem into single stage subproblems with release dates and tails. (See Chapter II, Section 5 and Chapter IV). We hereafter recall the studied HFS scheduling problem. The reader can refer to Chapter II for details.

Problem definition. The HFS problem is a multistage flowshop, where each stage $s \in \{1,...,K\}$ is composed of M_s identical parallel machines. Each machine is able to process one job at a time. Our task lies in scheduling a set of jobs I={1,..., n}, where a job consists of one operation for each stage. Those operations have to be realized sequentially from the first to the last stage without overlapping between stages. Job preemption and job splitting are not allowed. Each job i has a given processing time p_{is} at each stage s, $i \in I$ and $s \in \{1, ..., K\}$. Each job may have an initial release date r_{i1} (at the first stage) and a final tail q_{iK} (at the last stage). For the release date and tail definitions see Chapter II. All data of the problem (i.e. p_{is}, r_{i1}, q_{iK}; $i \in I$, $s \in \{1, ..., K\}$) are integer. We need to build a schedule with a minimum makespan, where the makespan μ is defined by $\mu = \underset{i \in I}{Max}[t_{iK}+p_{iK}+q_{iK}-1]$, where $t_{iK} > r_{iK}$ is the starting time of the last (K^{th}) operation of job i (which is thus finished at time $(t_{iK}+p_{iK}-1)$). 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. The remainder of this chapter is organized as follows:

Section **2** briefly reminds the reader of the relaxation we introduced in Chapter II, and which is used to design the first class of heuristics. Section **3** introduces the latter. It presents how the shifting bottleneck procedure is used to schedule the HFS, using several sequencing algorithms for the single stage subproblem. Sections **4** and **5** respectively introduce the classes of heuristics based upon the classical flowshop and upon priority rules with different assignment policies. Sections **6** and **7** present, respectively, the random scheduling and local search classes of heuristics. Section **8** summarizes the above. Section **9** presents numerical tests and result analysis. We conclude this chapter with some comments on our results.

2. THE SINGLE STAGE SUBPROBLEM

In Chapter II, Section 5 we have shown how the HFS problem can be relaxed into a single stage subproblem, i.e. a parallel machine scheduling problem with release dates and tails for each stage. So for each choice of $s \in \{1, \ldots, K\}$ we can define a single stage relaxation whose corresponding processing times remain unchanged while their release dates and tails are computed as follows:

$$r_{is} = r_{i1} + \sum_{t=1}^{s-1} p_{it} \quad ; \qquad q_{is} = \sum_{t=s+1}^{K} p_{it} + q_{iK} \qquad i \in I \qquad (3.1)$$

By, respectively, adding r_{i1} and q_{iK} to the computation of the release date and tail of a job i we are only dealing with HFS scheduling problems with release dates at the first stage and tails at the last one. However, the heuristics we present here can easily be updated to deal with HFS problems with waiting time in-between stages. (See Chapter II, Section 5).

In this chapter, we call *single stage s subproblem*, for $s \in \{1, \ldots, K\}$, the problem of assigning jobs in the set I to M_s identical machines at stage s, and of sequencing their operations at stage s while taking into account operations release dates and tails. The data of this subproblem are r_{is}, p_{is}, q_{is}, for $i \in I$, and for $s \in \{1,\ldots,K\}$.

In the first class of heuristics, the single stage subproblem is solved as a subproblem, or a subroutine using either by Jackson's heuristic or Carlier's branch and bound algorithm (Carlier 1987). See Chapter II, Section 5 for a brief description of these methods.

3. HEURISTICS BASED UPON THE SINGLE STAGE SUBPROBLEM

3.1 SCHEDULING THE HFS ONE STAGE AT A TIME

In this class of heuristics, and for practical purposes, we use the graph representation of a schedule introduced in Chapter II, Section 4. (See Figure 3.1 for an illustration of an HFS schedule graph representation). This class of heuristics schedules and reschedules the stages *independently* and requires frequent release dates and tails updating. The graph representation allows us to easily carry out these updating operations and gives a good overview of the HFS scheduling procedure. Remember that, in this graph representation of a schedule, nodes represent jobs operations. Conjunctive arcs, linking two operations of the same job but at two consecutive stages, represent the precedence constraints between successive operations of a job. Disjunctive arcs are used to represent the sequence of jobs on machines. At the beginning of the algorithm, the graph contains only the conjunctive arcs.

The heuristics in this class build a global schedule, i.e. an HFS schedule, by scheduling *a single stage at a time*. They proceed iteratively and differ from each other in the order in which the stages are scheduled.

A new schedule for the one stage. Each time a schedule of a stage is obtained using a heuristic or an exact algorithm, corresponding job sequences on the machines at this stage are translated into their corresponding paths of disjunctive arcs on the graph, at their corresponding stage. This new added schedule, represented by paths of disjunctive arcs or, equivalently, by the corresponding assignment and sequencing decisions, imposes constraints on the other stages.

Release dates and tails. The current release date r_{is} of job i at stage s, $i \in I$ and $s \in \{1,\ldots,K\}$, is computed as the length of the longest path in the graph of the current solution from the fictitious operation (0) to operation (i, s). This longest path corresponds to the succession of operations that need to be finished before operation (i, s) can start (remember that the arcs represent precedence and sequencing constraints). The tail q_{is} of job i at stage s, $i \in I$ and $s \in \{1,\ldots,K\}$, is the length of the longest path from operation (i, s) to the last fictitious operation (f) minus the processing time p_{is} of operation (i, s). This longest path corresponds to the succession of operations that need to be processed, after processing of (i, s) has been completed until the makespan is reached.

$$r_{is} = \text{longest-path}(0, (i,s)) \qquad i \in I;\ s \in \{1, ..., K\} \qquad (3.2)$$

$$q_{is} = \text{longest-path}((i,s), f) - p_{is} \qquad i \in I;\ s \in \{1, ..., K\} \qquad (3.3)$$

Critical stages. Each time a stage s^* is scheduled, release dates of some jobs in the subsequent stages ($t > s^*$) and tails of some of jobs in the proceeding stages ($t < s^*$) may change. Any stage t that has been scheduled once, and where some of its jobs release dates r_{it} or tails q_{it} are changed due to the new added schedule for stage s^*, is designated as critical. It is called critical because its corresponding schedule may no more be *valid* vis-à-vis the newly computed release dates and tails. The critical stages may need then to be rescheduled to come up with a new better schedule using the new data on the release dates and tails.

This phenomenon is illustrated in Figure 3.1, taken from the graph representation example presented in Chapter II, Section 4.1. This graph is the same as the one in Figure 2.2, presented in Chapter II, except that the schedule of the middle stage has changed. Table 3.1 presents the release dates and tails computed before (Fig. 2.2) and after (Fig. 3.1) the schedule of the middle stage has changed. In this graph (Fig. 3.1) stage 1 and stage 3 are critical because q_{31}, q_{51}, r_{13}, r_{23}, r_{33} and r_{43} have changed. Note also that in this case the longest path has changed.

	Before rescheduling stage 2						After rescheduling stage 2					
Job	r_{i1}	q_{i1}	r_{i2}	q_{i2}	r_{i3}	q_{i3}	r_{i1}	q_{i1}	r_{i2}	q_{i2}	r_{i3}	q_{i3}
1	0	31	5	23	19	14	0	31	*5*	*17*	**21**	14
2	5	23	13	19	17	17	5	23	*15*	*19*	**19**	17
3	6	18	17	14	22	9	6	**15**	*12*	*14*	**24**	9
4	11	15	15	11	22	5	11	15	*19*	*11*	**23**	5
5	2	28	6	19	15	12	2	**32**	*6*	*23*	15	12

Table 3.1: Release dates and tails before and after rescheduling stage 2

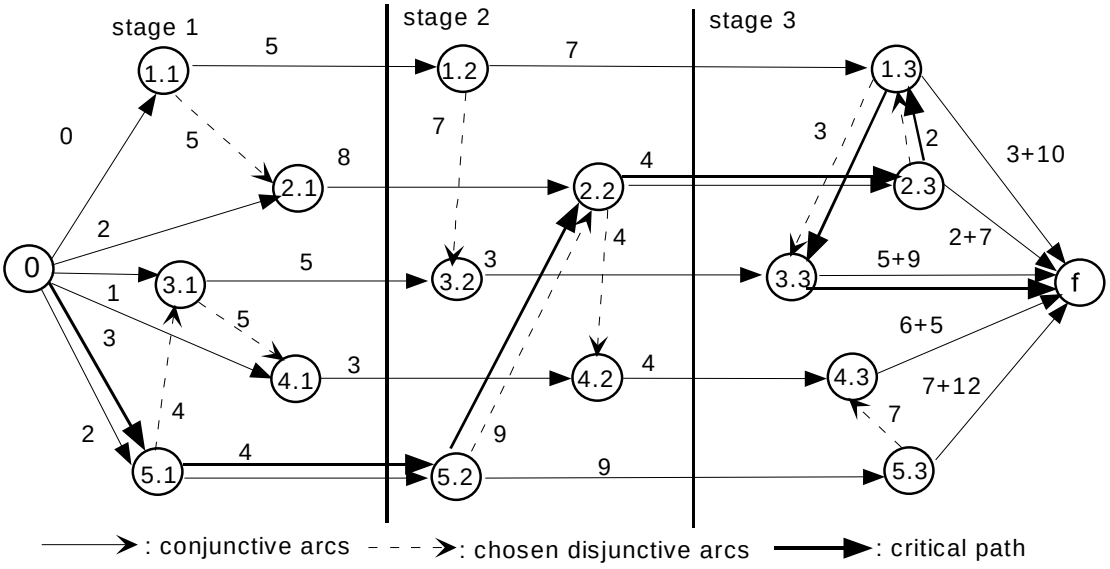

Figure 3.1: A graph representation of a three-stage HFS schedule

Scheduling and rescheduling the stages. Every time a new stage is scheduled or a critical stage is rescheduled, new disjunctive arcs are obtained. These arcs are then added

to, or replace, the disjunctive arcs of the corresponding stage. Release dates and tails are then updated to take the disjunctive arcs added to the graph into account, or equivalently to take the constraints imposed by the new scheduled or rescheduled stage into account.

A global feasible schedule. Once all the stages have been scheduled, and as soon as we recompute the release dates and tails at all stages, we obtain a global feasible schedule. The disjunctive arcs are directed arcs added for one stage at a time and do not create any cycle. Indeed, disjunctive arcs do not link the stages, and at each specific stage they consist of disjoint paths of disjunctive arcs that are never connected, i.e. each of these paths consist of a feasible sequence of jobs on one machine.

Computing the release dates and tails at all stages, once disjunctive arcs are added at all stages, or equivalently all the stages have been scheduled once, gives the longest path of the graph which is also the makespan of the corresponding schedule, i.e. time by which all jobs complete. This also means that once all stages have disjunctive arcs, and if by rescheduling we replace the disjunctive arcs at some stage, we do not always need to continue rescheduling the stages for obtaining a global feasible schedule. Recomputing release dates and tails suffice to obtain a global feasible schedule.

In the following we present two sets of heuristics. They differ in the order in which the stages are scheduled. In the first set, higher priority is given to bottleneck stages, i.e. Shifting Bottleneck Procedure (SBP). In the second set, the stages are sequentially scheduled starting with the first (last) stage and go forward (backward) until the last (first) stage is scheduled. Since we are building a global schedule by scheduling one single stage at a time, in all these heuristics a set S_0 is iteratively updated. S_0 contains the stages that have been scheduled at least once.

3.2 THE SHIFTING BOTTLENECK PROCEDURE

Adams et al., (1988) have introduced the Shifting Bottleneck Procedure (SBP) in order to schedule a jobshop with one machine at each stage. They have concluded that SBP produces very good results and is able to deal with large instances with up to 100 jobs.

The bottleneck stage. The main idea of SBP is to give a higher priority to the bottleneck machine, i.e. stage made up of parallel machines in our case. It works iteratively by scheduling the bottleneck stage s^* among the not yet scheduled stages, i.e. $s^* \in S \backslash S_0$, where S is the set of all stages. This stage is then added to the set of scheduled stages, i.e. $S_0 = S_0 \cup \{s^*\}$. In order to find the bottleneck stage s^* (i.e. to be scheduled next among those in set $S \backslash S_0$), we separately schedule each not yet scheduled single stage subproblem and select the one giving the maximum makespan, i.e. $s^* = \arg\max_{s \in S \backslash S_0} \text{makepsan}_s$ where makepsan_s is the makespan of the single stage s solution obtained by some scheduling algorithm.

Rescheduling. Every time a stage is scheduled for the first time, the already scheduled stages (i.e. stage in set S_0 including the last scheduled one) whose release dates and/or tails have changed, i.e. the critical stages, are rescheduled (once).

Figure 3.2 depicts the SBP as it is implemented in our implementation. Figure 3.3 provides an example of the order in which the stages are scheduled and rescheduled for a three-stage hybrid flowshop. Note that the stages are rescheduled only once and in the same order they were scheduled the first time. At the end of this example, we have a global feasible schedule.

When all stages have been scheduled once, because of chain reaction in step 2 of the algorithm, one can either stop iterating or keep on recommitting the r_{is} and q_{is} as well as rescheduling the critical stages. In our tests, the heuristic stops when an up-to-date global schedule is obtained, i.e. all stages are scheduled and the critical ones have been rescheduled once.

Let S be the set of all stages and S_0 be the set of the already scheduled stage
$S_0 = \varnothing$ at the start
While $S \backslash S0 \neq \varnothing$ do
Step 1. Identify a bottleneck stage s^* ($s^* = arg \max_{s \in S/S_0} \{makespan_s\}$)
$S_0 = S_0 \cup \{s^*\}$
Step 2. Reschedule each critical stage $s \in S_0$
($s \in S_0$ is critical if one of r_{is} or q_{is} changes $i \in I$)
End-while.

Figure 3.2: Shifting Bottleneck Procedure for the HFS

Stage 1	Stage 2	Stage 3
	[1]	
[2]		
(4)	(3)	
		[5]
(7)	(6)	(8)

[]: Scheduled as a bottleneck stage ( ): Rescheduled as a critical stage
Number = order in which the stages are scheduled and rescheduled

Figure 3.3: Scheduling order of a three-stage hybrid flowshop using SBP

For our tests, and from this procedure, i.e. SBP, two global heuristics for scheduling the HFS are then used, that is SBP_J and SBP_C. SBP_J uses Jackson's heuristic so as to schedule the single stage subproblem. SBP_C uses truncated (stopped after exploring 100 nodes) Carlier's branch and bound algorithm.

3.3 SCHEDULING THE HFS SEQUENTIALLY

The principle remains the same as in SBP but we now schedule the shop floor starting with the first (last) stage and go forward (backward) until we have scheduled the last (first) stage. After scheduling each stage we may reschedule some of the previously scheduled ones, i.e. stages in S_0. Therefore, these procedures are the same as SBP except for the scheduling of the bottleneck stages which is replaced by a sequential scheduling order (first to last stage, or last to first stage). We hereafter describe three sequential heuristics.

Scheduling Forward (SF). This heuristic schedules the shop, stage by stage, starting from the first stage and goes *sequentially forward* until the last stage has been reached. The completion time of each job at a scheduled stage s is taken as the release date of that job at stage s+1. The rescheduling process is not considered in this heuristic.

Scheduling Forward and Rescheduling the critical stages (SFR). This heuristic behaves in the same way as SF but every time a stage is scheduled for the first time, the already scheduled stages whose tails have changed are rescheduled once.

Scheduling Backward and Rescheduling the critical stages (SBR). This heuristic behaves similarly to SBP but the scheduling of the bottleneck stages is replaced by a backward sequentially scheduling order. It starts by the last stage and goes *sequentially backward* until the first stage has been reached. Every time a stage is scheduled for the first time, the already scheduled stages whose release dates have changed are rescheduled once.

For our tests, and from these three procedures (SF, SFR and SBR), six global heuristics for scheduling the HFS are used, that is SF_J, SF_C SFR_J, SFR_C, SBR_J, SBR_C. 'X_J' and 'X_C', respectively, stand for the procedure 'X' using Jackson's heuristic and truncated Carlier's branch and bound algorithm so as to schedule the single stage subproblem.

4. HEURISTICS BASED UPON THE TWO-STAGE FLOWSHOP

These heuristics schedule the HFS by slightly adapting or directly using heuristics that have been developed to solve classical flowshop scheduling problems. Guinet and Solomon (1996b) were the first to use the classical flowshop (FS) heuristics, i.e. the K-stage flowshop with one machine per stage, to schedule the HFS. They have adapted different classical flowshop heuristics to schedule the HFS. In their paper, from all the FS adapted heuristics, (Campbell, Dudek and Smith (1970) (CDS), Nawaz et al., (1983) and Townsend (1977)), for the makespan criterion and for non-bottleneck configurations (the hardest configurations, see later numerical test conclusions), *CDS* outperforms all the other heuristics.

In this class of heuristics, and according to Guinet's paper, we decided to use only CDS idea to schedule the HFS. We have added some new extensions to CDS so as to schedule the K-stage HFS. We hereafter recall CDS heuristic, as it has been introduced for the first time by Campbell, Dudek and Smith (1970) to schedule the K-stage FS.

CDS for the K-stage FS (i.e. M_s=1 for s=1,…,K). CDS, by aggregating stages, creates K-1 fictitious two-stage FS problems with one machine per stage. The processing times of the K-1 two-stage problems are defined as follows, where t represents the index of the two-stage problem, indices A and B are used to represent the two fictitious stages:

$$\text{For } t = 1, ..., K-1 \quad p_{iA}^{t} = \sum_{s=1}^{t} p_{is} \; ; \quad p_{iB}^{t} = \sum_{s=K-t+1}^{K} p_{is} \qquad \text{for all } i \in I \qquad (3.4)$$

Applying Johnson's algorithm (see hereafter) to the K-1 makespan minimization two-stage FS problems, gives K-1 permutation lists. Each list is used to build a permutation schedule for the K-stage FS. The list giving the shortest schedule and its corresponding makespan are selected by CDS as a solution for the K-stage FS (Campbell et al. 1970).

A Johnson's list (J-list) is built as follows. The jobs are partitioned into two sets, with set 1 containing all jobs with $p^t_{iA} < p^t_{iB}$ and set 2 all jobs with $p^t_{iA} > p^t_{iB}$. Jobs with $p^t_{iA} = p^t_{iB}$ may be in either set. The jobs in set 1 go first in J-list, and they go in increasing order of p^t_{iA}; the jobs in set 2 follow in decreasing order of p^t_{iB}. Ties are broken arbitrarily.

We hereafter recall how Guinet and Solomon (1996b) used CDS to schedule the HFS (CDSG). We also introduce two new CDS variants for scheduling the HFS (CDSH and CDSV). All these heuristics operate in two steps. The first step builds an ordered list of the jobs. The second step uses that list to assign machines to jobs and come up with a global HFS schedule. Since the second step is the same for all three heuristics, we will only present the first step for each of the three heuristics, by highlighting their main differences. These differences will also be depicted graphically. We then present the assignment step.

4.1 CONSTRUCTION OF AN ORDERED LIST OF THE JOBS

CDSG. In their paper Guinet and Solomon (1996b) turn the K-stage HFS into a K-stage classical flowshop (FS) with one machine at each stage. To do so, they divide the processing time of each job by the number of machines at the corresponding stage, and consider the K-stage HFS as a K-stage FS. The obtained K-stage FS is scheduled using the single permutation list constructed with CDS heuristic (Campbell et al., 1970). Once this single list is known, Guinet and Solomon transform it into a K-stage HFS schedule using the first assignment rule presented in step 2 (see Section 4.2). In CDSG heuristic, the processing times of the K-1 two-stage FS problems are defined as follows:

$$\text{For } t = 1, ..., K\text{-}1 \qquad p^t_{iA} = \sum_{s=1}^{t} (p_{is} / M_s);\ p^t_{iB} = \sum_{s=K-t+1}^{K} (p_{is} / M_s) \qquad \text{for all } i \in I \qquad (3.5)$$

Figure 3.4 shows graphically how CDSG builds a schedule using CDS heuristic in a 5-stage HFS.

CDSH. In Moursli (1997) we used another adaptation of **CDS** to the **H**FS. CDSH differs from CDSG in three points. (i) The original processing times are used and the K-1 permutation lists are built from the K-1 two-stage FS problems using Johnson's algorithm. (ii) Each of the K-1 lists is used to schedule the whole HFS, instead of scheduling the K-stage FS only as in CDSG. In other words, the permutation list coming from each two-stage fictitious problem is transformed in step 2 into an HFS feasible schedule. (iii) The best feasible schedule is then chosen among the K-1 HFS schedules.

Because the K-1 lists are directly used to schedule the K-1 *K-stage HFS*, instead of scheduling *K-1 K-stage FS* as in CDSG, the *processing times* used for defining the K-1 two-stage FS problems are the *original ones*. The two-stage FS problems are then defined as follows:

$$\text{For } t = 1, ..., K\text{-}1 \qquad p^t_{iA} = \sum_{s=1}^{t} p_{is} \;;\; p^t_{iB} = \sum_{s=K-t+1}^{K} p_{is} \quad \text{for all } i \in I \qquad (3.6)$$

The graphical representation of CDSH is the same as in Figure 3.4 except that the t^{th} Johnson's list (J-list t) is build using equation (3.6) and is used to build the t^{th} K-stage HFS schedule, (i.e. *K-HFSSt)*, instead of the t^{th} K-stage FS schedule. The best of K-1 K-stage HFS schedules, (i.e. K-FSSt, t=1, …, 4), is selected as the final schedule. Table 3.2 presents the main differences between CDSG and CDSH.

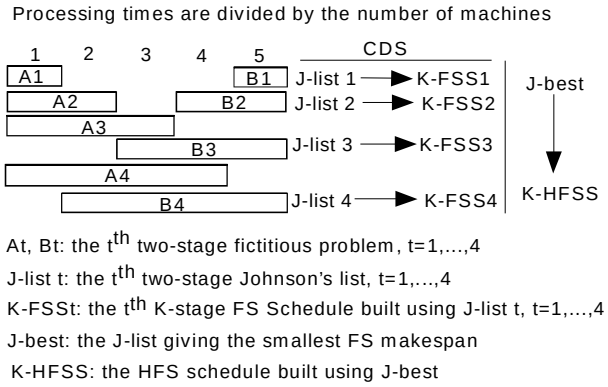

Figure 3.4: Example of a 5-stage HFS scheduled using CDSG

	CDSG	*CDSH*
Used processing times in J-lists	p_{is}/M_s	p_{is}
Shops scheduled with J-lists	**K-1 K-stage *FS***	**K-1 K-stage *HFS***
J-best	K-stage ***FS*** with min makespan	K-stage ***HFS*** with min makespan

Table 3.2: Main differences between CDSG and CDSH

CDSV. This heuristic is a new **V**ariant. It works in the same way as CDSH but the K-1 permutation lists are built using the following processing times:

$$\text{For } t = 1, ..., K\text{-}1 \qquad p^t_{iA} = \sum_{s=1}^{t} p_{is} \;;\; p^t_{iB} = \sum_{s=t+1}^{K} p_{is} \quad \text{for all } i \in I \qquad (3.7)$$

Figure 3.5 graphically depicts how CDSV builds an HFS schedule by modifying CDS idea.

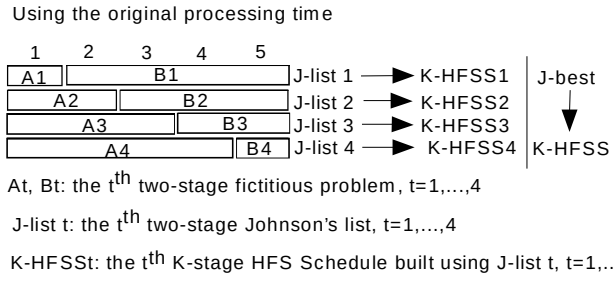

Figure 3.5: Example of a 5-stage HFS scheduled using CDSV

The main difference between CDSV and the first other two CDS's variants (CDSG and CDSH) lies in the way the K-1 fictitious flowshop problems are constructed. Because here we need to relax the multistage into a 2-stage problem so as to obtain an ordering of the jobs using Johnson's idea, CDSG and CDSH build several 2-stage relaxed problems. For each problem they take into account the processing time of each job at a set of successive machines as representing the processing time of that job on one of the two fictitious machines. For the 2-stage fictitious FS problems with $t>\lfloor K/2 \rfloor$, processing times at stages s=K-t+1, …, t are counted twice in the processing times (on machines *At* and *Bt*), while for the FS problem with $t<\lfloor K/2 \rfloor$ processing times at stages s=t+1,…, K-t are not taken into account.

One intuitive explanation of these choices could be that in CDSG and CDSH, the algorithms give similar *processing weight* to both machines *At* and *Bt*. In particular, when the processing times follow the same uniform distribution at all stages, the average processing times on the two fictitious machines are identical. In the case of CDSV the processing times at each stage s = 1, …, K are used only once either on *At* or *Bt* for t=1,…,K-1.

The drawback in CDSV is that the processing times on the two fictitious machines are sometimes unbalanced and, therefore, Johnson's idea is not really applied. For instance, for A1 and B1 in Figure 3.5, CDSV would probably order the jobs in increasing values of processing times at stage 1 only.

4.2 SCHEDULING THE HFS WITH AN ORDERED LIST OF THE JOBS

In this step, the ordered list of jobs is transformed into an HFS schedule by proceeding forward in time, and by scheduling stage by stage, from stage 1 to stage K. The completion time of a job at stage s is taken as its release date at stage s+1. The schedule of each stage is obtained by assigning the machines to the jobs and sequencing the jobs on each machine according to the predefined order obtained in step 1. To do so, one can use different assignment rules. We hereafter define two of them.

Assignment rule 1 (A1). This rule schedules the HFS one stage at a time, sequentially from the first to the last. It starts assigning the jobs according to their respective order in the list. The list is started from the beginning at each stage. The first (with respect to time) available machine is assigned to the next job of the list. The assignment order of the jobs is the same at any stage and is the one given by the list built in step 1. Note that, since we are respecting the order of the list at each stage and because we have parallel machines, though some jobs are ready, some machines may remain idle waiting for the next job of the list to be released from the previous stage.

This assignment rule produces a (hybrid) permutation schedule, in the sense that the so-called non-overtaking rule is respected in the assignment process or for the jobs start times, i.e. the jobs are assigned with respect to the order of the list and this order is maintained at all stages. The non-overtaking rule states that if job i is scheduled before job j at stage s, it is also the case at stage s+1, for s = 1, ..., K-1. Note that in the case of the classical flowshop with one machine at each stage, the majority of the exact methods

and the heuristics produce a permutation schedule. It is then interesting to test the non-overtaking rule for scheduling the HFS.

Assignment rule 2 (A2). The second rule, when assigning the jobs to the machines, respects the order of the list at the first stage only. For the subsequent stages, the first available machine is assigned to the ready job with the largest tail defined by (3.1). Apart from the first stage, this heuristic behaves similarly to Jackson's heuristic (See Chapter II) because jobs with largest tails are given higher priority. If there is more than one job with the largest tail, the order of the list is used to break ties.

As opposed to A1, the non-overtaking rule is not respected here. Indeed, because some jobs have smaller processing times than others and may be assigned, at the same stage, to different machines, it is *logical* that the initial assignment order of the jobs at the first stage may not be respected at the subsequent ones. And, using the same permutation list at each stage (i.e. rule A1) may not lead to good solutions like in the FS case. Note that this rule does not leave a machine idle unless no job is ready.

5. HEURISTICS BASED ON PRIORITY RULES

Similarly to the previous class, the heuristics in this class schedule the HFS one stage at a time, sequentially from the first to the last stage, and proceed forward in time at each stage. The completion time of a job at each stage is taken as its release date at the subsequent stage.

In this class, two priority rules, i.e. Shortest (SPT) and Longest Processing Time (LPT) rules are considered. These two rules are the most frequently used in the HFS scheduling literature. These two rules are the most frequently used in the HFS scheduling literature and seem to outperform (dominate) other priority rules with respect to the makespan criterion. See Tsubone et al., (1996), Kadipasaoglu et al., (1997), Hunsucker et al., (1989) and Hunsucker and Shah (1994). It is well known that SPT compared to LPT does not perform well for the makespan criterion, however we will see later that it is not the case for the HFS. This result seems counterintuitive to what is known in the single stage parallel machine case. Because here (in the HFS case) we have serial parallel machines, starting with jobs with smaller processing times at some stage reduces machine idle times at the subsequent stage. And, SPT might give better results then LPT. For instance, in a two-stage flowshop with processing times at stage 1 significantly smaller than processing times at stage 2, Johnson's algorithm would sequence the jobs nearly in SPT order.

Since there may be different ways of implementing these two priority rules, we have developed different variants. Only four of these variants dominate all the others and were retained in our final tests and analysis.

Variants 1 and 2, respectively, give higher priority to *local and remaining processing times*, while variants 3 and 4, respectively, give higher priority to processing times at the *first stage only* and to *total processing times* (at all stages). Variant 3 makes more sense

when used with assignment rule 2, since the ordered list is respected only at the first stage and the highest-tail-rule is used for subsequent stages.

Variant 1 (V1). This variant implements the priority rules in the classical way by scheduling the HFS stage by stage using at each stage s the corresponding processing time (p_{is}, $i \in I$). The first (in time) available machine is assigned to the job with the shortest/longest processing time, among the ready ones, in the case of SPT/LPT. We can see that this variant takes into consideration only local processing times (at the stage being scheduled) of ready jobs in order to decide which job to assign next.

Variant 2 (V2). This variant behaves in the same way as Variant 1 but, at each stage s, in order to decide which job to assign next among the ready ones, uses the sum of the *remaining processing times* (i.e. $\sum_{t=s}^{K} p_{it}$, $i \in I$) instead of local processing times at stage s. Note that Jackson's heuristic is different from this variant, because it considers only the remaining processing times at subsequent stages (excluding the being scheduled one).

As opposed to selection rule A1 used in Section 4, we consider in variants V1 and V2 only the ready jobs. This is because the selection lists are local (i.e. different from stage to stage) in V1 and V2, and therefore using A1 here would imply very long waiting or idle times. This observation was confirmed with numerical tests that we do not give here.

Variant 3 (V3). The third variant builds one ordered list of the jobs using the (shortest/longest) processing times of jobs at the first stage only, i.e. p_{i1}, $i \in I$. The ordered list is therefore the same at all stages. The selection of the next job is performed according to one of the assignment rules A1 and A2 presented in Section 4.

Variant 4 (V4). The fourth variant builds one ordered list of the jobs using the sum of the (shortest/longest) processing times of jobs over all stages, i.e. $\sum_{s=1}^{K} p_{is}$, $i \in I$. The ordered list is therefore the same at all stages. The selection of the next job is performed according to one of the assignment rules A1 and A2.

Other variants have also been developed and tested. Because they gave very large deviations from the lower bound they were discarded from our analysis and are not presented here.

We eventually get twelve heuristics. Two heuristics (SPT and LPT) for V1 and two heuristics for V2. Four heuristics (i.e. SPT and LPT with two assignments rules A1 and A2) for V3 and four heuristics for V4.

6. HEURISTICS BASED ON RANDOM SCHEDULING

As opposed to the above classes of heuristics where specific assignment and sequencing rules are defined, this class of heuristics uses different random sequencing and

assignment rules. These heuristics are fast and easy to implement. They are considered here as a basis for comparing all the heuristics presented in this thesis and to verify whether they (heuristics in other classes) are worth developing or whether any random heuristic suffices to find a good solution.

Like the heuristics in the second and third classes, those in this class are list heuristics. They also proceed in two steps, namely the construction of an ordered list of jobs, and the assignment of the machines to the jobs. They schedule the HFS stage by stage sequentially and proceed forward in time at each stage. The completion time of a job at each stage is taken as its release date at the subsequent stage. This class is composed of three heuristics. They differ only by the assignment rule used.

Random sequencing. All heuristic in this class use a randomly generated ordered list of jobs,. The ordered list is generated by assigning a random value to each job, and by sorting the jobs in a non-decreasing order with respect to their corresponding random values.

Assignment rules. The first heuristic (**RS**) randomly assigns machines to jobs. This heuristic is the same as assignment rule A1, presented in Section 4, except that, instead of assigning the first (in time) available machine to the next job in the list, we assign a randomly selected machine to the next job in the list, i.e. each of the M_s parallel machines is assigned a random value and the one with the higher one is selected. The second and the third heuristics use assignment rules A1 and A2, respectively, i.e. (**RS1**) and (**RS2**).

7. HEURISTICS BASED ON LOCAL SEARCH TECHNIQUES

Local search techniques are improvement methods. They proceed by exploring the neighborhood of a known solution for the sake of finding a better one. They do not guarantee finding the optimal solution but have the potential of improving an initial given one.

This class is composed of two local search heuristics. The first one is a Simple Local Search Technique (**SLST**) where the search is performed locally in the neighborhood of the seed solution, i.e. the current solution. The algorithm moves to a new seed solution only if the latter is better (i.e. its makespan is lower) than the current one. The second heuristic (**SA**) is a simulated annealing algorithm. Here, in order not to remain blocked in local optima the algorithm agrees to move to a worse solution with some probability depending on some cooling value. This cooling value is reduced, at each step, after the algorithm has explored a certain number of solutions and, consequently, reduces the probability of accepting worse solutions.

Both SLST and SA need a good starting solution given by a good and fast heuristic. SF_J has been chosen so as to provide this starting seed solution. In order to compare the heuristics in the other classes to those in this class, the latter are given a maximum amount of running time. Note also that these heuristics stop as soon as an optimal solution is found, i.e. equal to the global lower bound.

Both SLST and SA need a neighborhood definition of the HFS schedule solution so as to explore the *surrounding area* (neighborhood) of the seed solution. Before introducing these heuristics, we first present a neighborhood definition of an HFS schedule solution. The latter is based on the HFS graph representation introduced in Chapter II, Section 4.

7.1 NEIGHBORHOOD DEFINITION OF THE HFS SCHEDULE

Some observations regarding the longest path in the graph representation of the HFS schedule can be inferred. See an example of the graph representation of the HFS schedule in Chapter II, Section 4, Figure 2.2 and Figure 3.1 in this chapter.

1. The longest path is composed of K subpaths, sp_s, s=1, ..., K one for each stage.
2. Each subpath sp_s is composed of disjunctive arcs only, and hence belongs to one stage. It also has a cardinality (number of nodes) of 1 at least, i.e. the subpath passes through one operation at least, at each stage. In Figure 3.1 the subpaths are {(1,1); (2,1)} at stage 1, {(2,2)} at stage 2 and {(2,3); (1,3); (3,3)} at stage 3.
3. Each subpath corresponds to the complete, or a part of the, path representing the sequence of operations on one machine at the corresponding. Indeed, two paths at the same stage, corresponding to two different machines, are never connected.
4. Each subpath sp_s starts with some operation (i, s) and finishes with another operation (j, s), in the same stage. Since two successive operations of one job belong to two different stages, operation (i, s) and operation (j, s) have conjunctive incoming and out-coming arcs, respectively, linking this subpath to the remaining of the longest path, i.e. the subpath are linked to each other with conjunctive arcs.

Therefore, in order to reduce the makespan of a given schedule, one needs to reduce the corresponding longest path in the graph representation of that schedule, (see Matsuo et al., (1988) and Van Laarhoven et al., (1988)). Or in particular one can reduce the length of one of these subpaths.

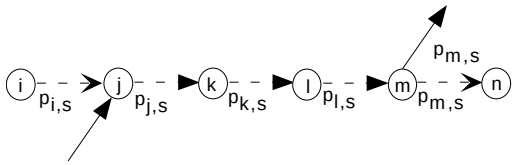

Figure 3.6: Path of disjunctive arcs (job sequence on a machine)

Figure 3.6 gives an example of a path of disjunctive arcs or a sequence of jobs (i, j, k, l, m, n) on one machine at stage s. Its corresponding subpath sp_s belonging to the longest path, starts with job j and ends with job m, sp_s = {j, k, l, m}. It has a cardinality of four (passes through 4 jobs, $|sp_s|$=4) and a length defined as of (p_{js}+ p_{ks} + p_{ls} + p_{ms}). The two conjunctive incoming and outcoming arcs (plain arrows) show the subpath as part of the whole longest path of the graph.

Neighborhood definition. As a neighborhood of an HFS schedule we decided to choose to visit only solutions which have the potential of reducing the longest path. Any modification made outside the longest path would never reduce the latter. A schedule in the neighborhood of a given schedule is defined by *moving one operation of one job*,

belonging to the longest path, i.e. changing its sequencing position in the graph representation of a schedule (see later how this job move is carried out). This *move*, consequently, occurs at one stage (or at one of the K subpaths). Only those subpaths having cardinality larger than 1 are considered. Indeed, a subpath composed of one operation could never be reduced. In order to reduce the length of a chosen subpath sp_s', we need to remove one node from such a subpath.

***A longest path reduction*.** Since a graph may have more than one longest path, for a *specific operation move*, we arbitrarily choose one of these longest paths and reduce it by reducing one of its composing subpaths, i.e. sp_s'. Reducing the length of sp_s' definitely reduces the length of the longest path to which it belongs. However, this longest path reduction does not necessarily reduce the other longest paths, if any. Furthermore, it might lead to the creation of a new longest path, which in turn could be longer than the one that has just been reduced. Figure 3.7 gives the general structure of a subpath sp_s at some stage s (*subscript s is ignored in the graph*).

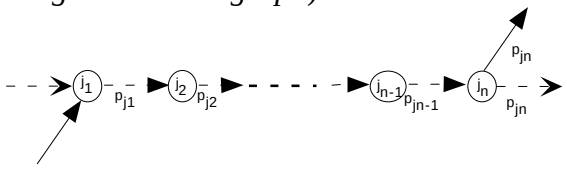

Figure 3.7: General structure of a subpath sp_s'

Let sp_s' = {j_t /t=1, ..., n} with cardinality n and length $\sum_{t=1}^{n} p_{j_{t,s}}$, be a *specific subpath* belonging to a *specific longest path*. One can observe that sp_s' can be reduced as follows:

1. Sp_s' is reduced by $\sum_{t=2,...,v} p_{j_{t,s}}$ when j_1 is moved after job j_v for v = 2, ..., n-1
2. Sp_s' is reduced by $\sum_{t=v,...,n-1} p_{j_{t,s}}$ when j_n is moved before job j_v for v = 2, ..., n-1
3. Sp_s' is broken into two singleton and non-connected subpaths when j_1 is moved after job j_n or when j_n is moved before j_1, i.e. {j_1} and {j_n} which length p_{j1} and p_{jn}, respectively.
4. Sp_s' is reduced by some p_{jv} when job j_v is simply removed from this path of disjunctive arcs (or from the machine to which it is assigned to), and put on some randomly-selected other machine.

Figure 3.8(a) gives an example of a subpath sp_s made up of 5 jobs, j_t, t=1, …, 5, with a length of 19 time units.

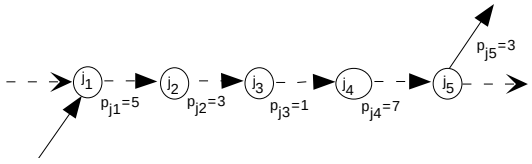

Figure 3.8 : (a) An example of a subpath sp_s

The following four examples show the new length of the sp_s after each of the four above mentioned moving operations is performed.

- Sp_s is reduced by 4 when j_1 is moved after job j_3, Figure 3.8(b).
- Sp_s is reduced by 11 when j_5 is moved before job j_2, Figure 3.8(c).
- Sp_s is broken into two singleton $\{j_5\}$ and $\{j_1\}$ which length 3 and 5, respectively, when j_1 is moved after job j_5, Figure 3.8(d).
- Sp_s is reduced by 7 when job j_4 is simply removed from sp_s, Figure 3.8(e).

One can also notice that moving any job j_t, t = 2, …, n-1, after or before any other job $j_{t'}$, t'=2,...,n-1, where t≠t', will not reduce sp_s'. Indeed, after such a move, the sequencing of operations changes but sp_s' still starts with job j_1 and finishes with job j_n. It also still contains the same set of operations, each of which has the same length as before the moving operation.

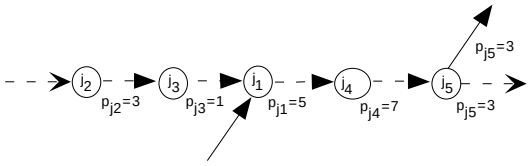

Figure 3.8(b)

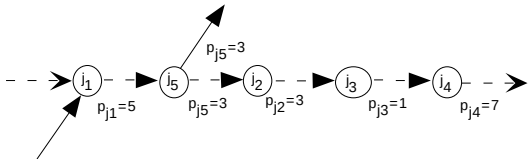

Figure 3.8(c)

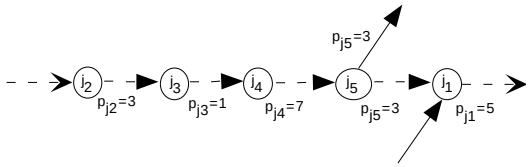

Figure 3.8(d)

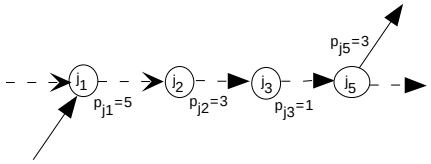

Figure 3.8(e)

Neighborhood exploration. In order to explore the neighborhood of the seed solution, we have chosen to process one of the above mentioned four operations. These operations have the potential of reducing the longest path in the seed solution. Note however that after processing one of those operations, this *specific longest path* is reduced and that others may still exist or may be created due to the processed operation. In order to perform one of these operations we proceed as follows:

- *Subpath selection.* A subpath sp_{s*} with cardinality greater than one is randomly selected, i.e. a stage s^* is selected.
- *Step selection.* Within sp_{s*}, a step v=1, ..., n, is randomly selected, where n=$|sp_{s*}|$
- *Moving selection*. One of the three following operations is randomly selected.
  1. Moving job j_1 after job j_v with v≠1
  2. Moving job j_n before job j_v with v≠n
  3. Moving job j_v after the t^{th} operation assigned to machine m, where t and m are randomly selected.

This neighborhood definition of an HFS schedule is used to design two local search techniques, i.e. SLST and SA, which we present in the following subsections.

7.2 SIMPLE LOCAL SEARCH TECHNIQUE

In the pure local search technique, the neighborhood of a current solution is completely explored and the heuristic moves to the best solution found afterwards, which becomes

the current solution. Since exploring the complete neighborhood of an HFS schedule solution may be time consuming, we decided to use a slightly modified version which we call *Simple Local Search Technique* (SLST). In this heuristic we explore randomly the neighborhood of an HFS schedule and move to any new better solution found, which becomes the current solution. Figure 3.9 shows the local search algorithm as implemented in our tests.

Let i be an initial seed solution of an HFS schedule generated by a heuristic
Let S_i be the makespan of solution i
While the running time is smaller than the maximum running time do
    Select solution j in the neighborhood of i
    If $S_j < S_i$ replace solution i with solution j
End-while.

Figure 3.9: Simple Local Search Technique (SLST)

7.3 SIMULATED ANNEALING

Simulated annealing is a randomized local search technique. It differs from other local search methods in that it accepts to move to a worse solution in the neighborhood of the seed solution, with some probability depending on some cooling value. The latter is decreasing after a certain number of solutions have been explored.

The algorithm runs in several steps. At each step t=1, …, T, the algorithm visits a certain number of solutions in the neighborhood of the seed solution. The cooling value c_t is decreased at the beginning of each step t.

Let i denote the seed solution and j denotes some other solution within the neighborhood of i. Also, let S_i and S_j denote the respective objective functions or makespans of these solutions. If $S_j < S_i$, solution j becomes the seed solution otherwise j is accepted with some probability AP_{ij}. Metropolis (1953) used the following acceptance probability, which we decided to use in our heuristic:

$$AP_{ij} = \min \{1, \exp(-(S_j-S_i))/ c_t\} \qquad (3.8)$$

We also decided to use the following decreasing function that has been introduced by Van Laarhoven et al., (1988), for a simulated annealing algorithm for the makespan minimization job shop scheduling problem, where σ_t is the standard deviation of the solution values obtained at the step t, and where δ is some positive control parameter.

$$c_{t+1} = \frac{c_t}{1+\frac{c_t.\ln(1+\delta)}{3\sigma_t}} \qquad (3.9)$$

When applied to several problems in combinatorial optimization, Van Laarhoven and Aarts (1987) have found simulated annealing extremely robust (final solutions within 2% of the global optimum), when δ is chosen sufficiently low (0.1 or smaller).

Let T be the number of steps and V be the number of solutions to be visited at each step. In order to stop this heuristic after a running time slightly equivalent to the running time taken by other heuristics, V was fixed with respect to the size of the configuration tested in terms of number of jobs and number of stages. Figure 3.10 gives the simulated annealing algorithm as implemented in our tests.

Let i be an initial seed solution of an HFS schedule generated by a heuristic
For t = 1, ..., T (steps)
    For v = 1, ..., V (number of visited solutions at step t)
        Select solution j in the neighborhood of i
        If $S_j < S_i$ *replace solution i with solution j*
        otherwise
            Generate a random value a between 0 and 1
            If $a < AP_{ij}$ *replace solution i by solution j*
    End-for
    Compute c_{t+1} *using equation (3.9)*
End-for

Figure 3.10: Simulated Annealing algorithm (SA)

8. GLOBAL HEURISTICS

We end up with five classes made up of 31 heuristics, which we hereafter summarize.

Class 1. Single stage based heuristics (CH1). This class contains two sets of heuristics (Section 3).

- The Shifting Bottleneck Procedure SBP using either Jackson's heuristic, SBP_J, or a truncated (stopped after 100 nodes) Carlier's branch and bound algorithm, SBP_C, for scheduling the single stage subproblems.
- The sequential heuristics denoted by SF_J, SF_C, SFR_J, SFR_C, SBR_J and SBR_C. S stands for sequential heuristics, F and B for Forward and Backward, R for the Rescheduling process, which is carried out once for each critical stage. J and C stand for scheduling the single stage subproblem using Jackson's heuristic and a truncated (stopped after 100 nodes) Carlier's branch and bound algorithm, respectively.

Class 2. Two-stage flowshop based heuristics (CH2). This class contains three sets of heuristics (Section 4).

- CDSG1 and CDSG2 use CDS idea by dividing the processing time at each stage by the corresponding number of machines. 1 and 2 stand for assignment rules A1 and A2, respectively.
- CDSH1 and CDSH2 are also derived from CDS idea using the original processing times. Each of the K-1 Johnson's lists is transformed into an HFS schedule. 1 and 2 stand for assignment rules A1 and A2, respectively.
- CDSV1 and CDSV2 use a new variant to construct permutation lists and use assignment rules A1 and A2, respectively.

Class 3. Priority rules based heuristics (CH3). This class contains twelve heuristics using four variants and two assignment rules (Section 5). SPT1, LPT1, SPT2, LPT2,

SPT31, SPT32, LPT31, LPT32, SPT41, SPT42, LPT41, and LPT42. SPT and LPT, respectively, stand for short and long processing time. The first number, in the heuristic name indicates the variant (1, 2, 3 and 4) while the second, if any, indicates the assignment rule number (A1 and A2) from Section 4. For variants 1 and 2, the permutation list is local to each stage and a new assignment rule is defined. For variants 3 and 4, the permutation list is global and the assignment rules A1 and A2 are used.

Class 4. Random scheduling (CH4). This class contains three heuristics using three different assignment rules (Section 6). They all use a randomly generated ordered list of jobs. The first heuristic (**RS**) randomly assigns machines to the ordered list of jobs. The second and the third heuristics use assignment rules A1 and A2, respectively, i.e. (**RS1**) and (**RS2**).

Class 5. Local search techniques (CH5). This class contains two heuristics (Section 7). The first one (**SLST**) is a Simple Local Search Technique. The second one (**SA**) is a Simulated Annealing technique.

9. NUMERICAL TESTS

The heuristics have been programmed using an object oriented language (C++) and run under Windows NT 4.0 on a Pentium II 350 MHz with 64 MB RAM. The processing times, the release dates and tails were generated uniformly ($p_{is} \in [1, 100]$, $r_{i1} \in [1, 100]$ and $q_{iK} \in [1, 100]$, $i \in I$ and $s \in \{1, \ldots, K\}$).

Lower bounds are used so as to evaluate the heuristics performances. See Chapter II, Section 5 for a brief description and Chapter IV for a detailed description of the lower bounds.

9.1 TESTED CONFIGURATIONS

The five classes of heuristics are tested on several classes of shop floor configurations. We considered five classes of configurations, **CCi, i=1,...,5**. **CC1 and CC2** are called non-bottleneck configurations because the number of machines and data ranges are the same at all stages. In **CC1** the jobs have zero initial release dates and zero final tails. In **CCi, i=2,…,5.** the jobs have non-zero initial release dates and non-zero final tails. The release dates and tails were generated for the first and the last stage, respectively. **CC3** is a special class where the number of machines is different from stage to stage, but the processing times are generated so as to equilibrate the load at all stages. The configurations in this class are also called non-bottleneck configurations. Let $s_* = \arg \max_{s=1,\ldots,K} M_s$ be the stage with the largest number of machines. Let lb_{s*} and ub_{s*} be the lower and upper limits on the processing time range used at stage s^*. For **CC3** the processing time ranges are updated at each stage s as follows:

$$lb_s = \lceil lb_{s*} * M_s / M_{s*} \rceil \qquad ub_s = \lceil ub_{s*} * M_s / M_{s*} \rceil$$

CC4 and CC5 are composed of what is called the bottleneck configurations. In CC4, at some stage the number of machines has been reduced by 1 so as to create a bottleneck stage by the number of machines. In CC5, the number of machines is the same at all stages but at one stage the lower and upper bound limits on the processing time range have been increased by 20 time units, i.e. $p_{is} \in [20, 120]$, $i \in I$ and $s \in \{1,\ldots,K\}$, so as to create a bottleneck stage by processing times.

We tested these configurations (non-bottleneck and bottleneck configurations) separately because bottleneck configurations seem much easier to solve, i.e. giving smaller deviations from the lower bound.

Since in industrial contexts HFS configurations are so diverse we tested, separately, the three types of non-bottleneck configurations and the two types of bottleneck configurations. For the non-bottleneck configurations, in CC1 and CC2 the number of machines and the processing times range are the same at all stages, while in CC3, the number of machines differs at some stage but the load is equilibrated by adjusting the processing times range at this stage. For bottleneck configurations, in CC4 one bottleneck stage is generated by a smaller number of machines according to the remaining stages while in CC5 a bottleneck stage is generated by increasing the processing times range at this stage and by keeping the same number of machines at all stages.

Actually, combining all types of configurations in a single experiment does not reflect the real performance of the heuristics with respect to hard problems, i.e. the non-bottleneck problems. Note also that, though the number of machines is uniform through the shop in the case of CC1 and CC2, and that the processing times ranges are updated in the case of CC3, the generated instances might still contain bottleneck stages due to the random generation of the data. Reducing the number of machines by one at one stage, in the case of CC4, and increasing the lower and upper bound limits on the processing times range, in the case of CC5, creates an artificial bottleneck stage. Note that at some bottleneck stage s, in the case of CC4, the smaller is M_s the larger is the relative capacity reduction at the bottleneck stage s. Also, in the case of CC5, the larger is the increase of the processing times range at some stage s, the larger is the corresponding load vis-à-vis the other stages $t \neq s$.

Different configurations, in terms of number of stages, number of machines and number of jobs, have been designed. For the same number of stages and jobs, different numbers of machines were considered. For the non-bottleneck case (CC1, CC2 and CC3), the designed configurations have 2, 3 and 5 stages with 3, 5 and 8 parallel machines and the number of jobs are 20, 30, 50 and 100. 50 instances have been tested for each specific configuration. In CC4 and CC5 (i.e. bottleneck cases) the configurations are the same except that *one* artificial bottleneck stage has been created at the middle stage (last in case of 2-stage HFS).

9.2 AVERAGE DEVIATION BASED ANALYSIS

Tables **3.i**, i = 3,…,7 (All numerical test tables are given at the end of this chapter) present the results of the numerical tests. Column n gives the number of jobs. Columns

Ms (s=1,...,5) give the number of parallel machines at stage s. Columns depicted by the *heuristic name* (see Section 8) and "Best" are average deviations over the 50 instances (in percentage (%)). For instance, "Best" is obtained by selecting the best value (average deviation) found by all heuristics for each of the 50 problem instances. Column '*heuristic name*' gives the average deviation (in %) from the lower bound *DHB* (DHB is one of the best obtained lower bound on the HFS problem which we present in Chapter IV) on the makespan obtained by the corresponding heuristic. Column 'Best' gives the average deviation (in %) from the lower bound DHB of the best value obtained over all the heuristics *for each instance. In the remaining we call these average deviations the* ***values*** *given by the heuristics*.

The five classes of heuristics (CHi, i=1,…,5) were tested on the five classes of configurations (CCi, i=1,…,5). Table 3.3 compares the heuristics for the non-bottleneck configurations (**CC1**) with zero release dates (r_{i1}=0) and final tails (q_{iK}=0). Tables 3.4 and 3.5 compare the heuristics for non-bottleneck configurations (**CC2 and CC3**) with non-zero release dates (r_{i1}>0) and final tails (q_{iK}>0). Tables 3.6 and 3.7 compare the heuristics for the bottleneck configurations (**CC4 and CC5**) with non-zero initial release dates and final tails. Table 3.8 summarizes the first five tables by giving the average, the standard deviation, the minimum and the maximum *values given by the heuristics*. It also gives the percentage (%) of the test problems for which *each heuristic has produced the BEST solution (%BEST)* and the percentage (%) of the test problems for which each heuristic *was the only* (O) *one* producing the BEST solution *(%OBEST)*. Each row of this table corresponds to 1800 test problems.

Table **3.9** presents, *for each heuristic*, the average time needed in seconds to run 50 instances on a Pentium II 350 MHz with respect to different configuration sizes. Row 's-n' (s=2, 3, 5 and n=20, 30, 50, 100) corresponds to the average time needed for a configuration with s stages and n jobs.

The following major general conclusions based upon the values (average deviations) given by the heuristics can be drawn (Table 3.8).

COMPARING THE CLASSES OF HEURISTICS TO EACH OTHER

- When choosing the best value among all heuristics we obtain, on average, very good results compared to those found in the literature: on average 6% for non-bottleneck configurations (CC1, CC2 and CC3), and 4% for bottleneck configurations (CC4 and CC5).
- Bottleneck configurations (CC4 and CC5) seem to be much easier to solve compared to non-bottleneck configurations (CC1, CC2 and CC3), in the sense that they are giving smaller deviations from the lower bound.
- For non-bottleneck configurations (CC1, CC2 and CC3) no class of configurations seems much harder to solve than another, i.e. on average each heuristic gives nearly the same average deviation for the three classes of heuristics. The same remark can be made for bottleneck configurations.
- Assignment rule 2 (giving higher priority to jobs with largest tails, Section 4) when used in CH2, CH3 and CH4, in general, gives better results than assignment rule 1 (using the same order of the jobs to schedule each stage).

- In general, heuristics using Jackson's idea (i.e. giving higher priority to jobs with largest tails: all heuristic in CH1 and those in CH2, CH3 and CH4 using A2) prevail.
- The priority rules based heuristics (CH3) and randomly scheduling heuristics (CH4) are largely outperformed by the other heuristics (CH1, CH2 and CH5), i.e. the values obtained by CH3 and CH4 are, at least, two times those obtained by CH1, CH2 or CH5.
- SR2 results, which are the best in CH4, are at least, two to three times those obtained by the heuristics in CH2 using assignment rule A2. According to these results, one simply has to forget about using random scheduling for HFS problems. We conclude here that HFS scheduling problems do need elaborated algorithms for finding good solutions.
- Local search based heuristics (CH5) do improve the initial solution (provided by SF_J), reducing it by an average of 2 to 3 % from the lower bound. However, SF_C which also starts by the same value as (CH5) and by consuming an equivalent amount of running time, gives better results than heuristics in CH5. Both heuristics use an average running time of 190 second for processing 50 instances. (Table 3.9).
- CH2 seems to give better results than CH1. However, in the remaining analysis we will see that some heuristics in CH1 and CH2 are considered to give similar performance (see later ANOVA based analysis).

COMPARING THE HEURISTICS WITHIN EACH CLASS

CH1

- For CH1, i.e. single stage based heuristics, scheduling the stages sequentially forward (SF and SFR) and using the Shifting Bottleneck Procedure (SBP) give better solutions than scheduling the stages Sequentially Backward (SB). For all heuristics in CH1, improving local solutions by scheduling the single stages with Carlier's truncated branch and bound gives better results than using Jackson's heuristic. However, using Carlier's truncated algorithm is much more time consuming (Table 3.9). For SF_J and SF_C (for CC1 only), it is better to reschedule the critical stages than not.

CH2

- For CH2, i.e. two-stage flowshop based heuristics, CDSH heuristics outperform CDSV and CDSG. CDSV heuristics outperform CDSG except for CC1. This is certainly due to the fact that CDSH and CDSV explore a larger number of solutions by scheduling the HFS with K-1 different lists, while CDSG heuristics use only the best K-machine flowshop list to schedule the HFS.
- CDSH2 heuristic, on average, gives the smallest deviation from the lower bound, 8% for CC1, CC2 and CC3 and, 5% and 6% for CC4 and CC5, respectively. However, we can see in Table 3.8 (%BEST) that CDSH2 provides an average of 37% of the best solutions, only, and its average value is 2% (1% for CC4) higher than the best solutions found.

CH3

- For CH3, i.e. priority rules based heuristics, *SPT1*, *LPT2* and *SPT42* give the best solutions. For bottleneck configurations (CC4 and CC5) SPT1 and SPT41 give good results compared to the other priority rules based heuristics.

CH4

- The pure random scheduling (SR0) and random scheduling using assignment rule A1 (SR1) give aberrant results (average deviations from the lower bound larger than

100%). SR2 gives more reasonable results when compared to SR0 and SR1. This is certainly due to the use of the largest-tail-first rule (assignment rule A2).

CH5

- Simple Local Search Technique (SLST) gives nearly same results as simulated annealing heuristic (SA) except for CC1. It does not pay much to use a more elaborated algorithm.

As already highlighted CDSH2 gives the smallest average deviation from the lower bound but provides an average of 37% of the best solutions only. Someone interested in computing a good upper bound cannot only pick up the heuristic giving the smallest average deviation from the lower bound. However he or she needs to know which subset of heuristics to use in order to obtain robust and good solutions for all potential problem configurations or characteristics. Hence, in order to select the best subset of heuristics we decided to use three selection rules. The decision of eliminating the heuristics by rule 1 and 2 was strengthened by rule 3.

SELECTION RULES

1. The heuristics that have never given the best solution are eliminated.
2. The heuristics giving an average value greater than ($V_h + 3*s_h$) are eliminated. V_h and s_h being, respectively, the average value and the standard deviation of the best heuristic on average (CDH2).

   Table 3.10 gives for each class of configurations the maximum deviation tolerated by rule 2 and the *remaining heuristics* after applying selection rules 1 and 2. Heuristics which average deviations have been struck through are also eliminated with respect to each class of configurations, e.g. SBR_J is eliminated except for CC1.
3. The *remaining heuristics* are compared using the analysis of variance (ANOVA) to check whether their average performances are likely the same or some of them can still be eliminated because they are dominated.

9.3 ANOVA BASED ANALYSIS

In our application, ANOVA analysis tries to answer the following question: are the outputs (average deviations) of the remaining heuristics really different or the output observable differences are mainly due to other reasons than the performances of the heuristics?

The F test. For each class of configuration CCi, let $s^2_{\overline{V}}$ be the variance of the average values $\overline{V}$ given by the remaining heuristics (CCi values in Table 3.10 given by leftover heuristics). Let s^2_p be the variance of all values given by these heuristics (CCi values in Tables 3.i, i=3,…,7 given by leftover heuristics) and n is the number of configurations in CCi (36 configurations in our case). The computed *F* by ANOVA based on the observed results is as follows:

$$F = \frac{n s^2_{\overline{V}}}{s^2_p} = \frac{\text{explained variance}}{\text{unexplained variance}}$$

If the obtained *F* is smaller than the *critical-F* given by the *Fisher distribution*, we conclude that the means of the remaining heuristics, i.e. if they were tested on the whole population of instances, are equal, i.e. the heuristics performances are the same. Otherwise, we conclude that the distributions of the heuristics results are different. The heuristics, when applied to the same instances, behave differently and consequently are different. The reader who is not familiar with ANOVA can, for example, refer to Wannacott and Wannacott (1990).

ANOVA results are given in Table 3.11 when applied to the leftover heuristics after selection rules 1 and 2, and to the final selected heuristics (marked bold) with respect to each class of configurations, e.g. SF_C is selected by ANOVA except for CC4 and CC5.

ANOVA was carried out with a 2.5% error factor. For each class CCi (i=1, …, 5), it has firstly been applied to all leftover heuristics after applying selection rules 1 and 2. The corresponding results are given in columns "*F For All H.*" and "*Crit. F For All H.*" (Table 3.11). We can see here that the *F* obtained by All leftover Heuristics is larger (for example *16.9* for CC1) than the Critical F (for example *1.89* for CC1). So in order to eliminate the heuristics that do not give the same performances as the best ones (with lower average deviations) we proceed as follows. If the obtained *F* is greater than the *critical F* we then remove the heuristic with the largest (average deviation) value and redo the ANOVA analysis. This is stopped when the obtained *F* is smaller than the *critical F* in which case we conclude that the remaining heuristics can be considered to have the same average performance. Columns "*F For Bold H.*" and "*Crit. F For Bold H.*" give ANOVA results when applied to the final selected heuristics (marked bold).

ANOVA analysis has selected 12 heuristics (from those in CH1, all those in CH2 and in CH5) for non-bottleneck configurations (CC1, CC2 and CC3) and 6 heuristics (from those in CH1 and in CH2) for bottleneck configurations (CC4 and CC5) as giving same average performances, with respect to each class of configuration, viz. CC4 and CC5, SF_C, SFR_C, CDSH1, CDSH2, CDSV1 and CDSV2 are considered giving same average performances.

This conclusion raises another question. If we are interested in finding the best solution, can we just use one of these heuristics, or do we need to use all the heuristics selected by ANOVA? And, in the latter case, are all the selected heuristics by ANOVA enough to find the best solutions? This question can be answered when analyzing the specific contribution of each heuristic in finding the best solution.

9.4 BEST SOLUTION BASED ANALYSIS

Table 3.12 provides us with some statistics when the tested heuristics give the best solution. Each column corresponds to the results given by a subset of heuristics and is in percentages over all problems tested. Each row corresponds to a set of HFS configurations having the same size in terms of number of stages (for example CC3-3s is the set of the 3-stage configurations in CC3). Row "*CCi-all*", i=1, …, 5, is the average value given by the 2, 3 and 5 stages configurations in CCi. Column "*CHi*", i=1, …, 5, shows the percentage where the best solution is given by **one** of the heuristics in class

CHi. Columns with a name composed of the term 'O' give the percentage where the best solution is **only** given by **one** of the heuristics belonging to the corresponding subset of heuristics. Terms J, C, H, V, G, A1, A2, LS and SA represent the following subsets of heuristics:

J	C	H	V	G	A1	A2	LS	SA
SF_J	SF_C	CDSH1	CDSV1	CDSG1	CDSV1	CDSV2	SLST	SA
SFR_J	SFR_C	CDSH1	CDSV2	CDSG2	CDSV1	CDSV2		
SBR_J	SBR_C				CDSG1	CDSG2		
SBP_J	SBP_C							

Column 'CH1 or CH2' gives the percentage where the best solution is given by either one of the heuristics in CH1 or one of those in CH2, but not both. 'CH1 & CH2' gives the percentage the best solution is given by at least one of the heuristics in set CH1∪CH2. Column 'Less', represents the subsets of best and least time consuming heuristics, Less=CH2∪J. Column 'ANOVA' gives the percentage where the best solution is given by at least one of the heuristics belonging to the subset of heuristics selected by ANOVA. Column 'ANOVA+Less' gives the percentage the best solution is given by at least one of the heuristics in set ANOVA∪Less. Column 'ANOVA–CH5' gives the percentage the best solution is provided by at least one of the heuristics in set ANOVA excluding those in CH5. The row 'Average' gives the contributions of each subset of heuristics in finding best solutions.

We can see that, on average, nearly 97% (Table 3.12) of the best values are produced by the heuristics selected by ANOVA and nearly only 90% when removing local search based heuristics.

We can also see that CH1 and CH2 are complementary heuristics in the sense that most of the time (for 85% of instances) the best value is given either by CH1 or CH2, but not by both. 64% of the best solutions are provided by heuristics in CH2 only, versus 21% for those in CH1. For only 11% = (96-85)% of instances both CH1 and CH2 are giving the same best value. The contribution of the heuristics in subset H is very important in general and in CH2. 27% of the best solutions are provided by heuristics in subset H only, and 14% are provided by those in subset V only. 33% of the best solutions are provided by heuristics in subset A2 only, versus 10% by A1 only. We can see that assignment rule A2 is more powerful than assignment rule A1.

Heuristics using Carlier's branch and bound (Subset 'C') and local search heuristics, (subset CH5) are the most time consuming heuristics. However, 12% of the best solutions are provided by subset 'C' only, versus 1% by CH5.

These observations lead us to the conclusion that one who is interested in always finding a good solution, i.e. to design a robust scheduling algorithm, needs to use heuristics in CH1 and CH2, so as not to miss solutions.

Actually, if we are interested in heuristics that are the least time consuming (see Table 3.9) for example for computing upper bounds at the search tree nodes in a branch and bound algorithm, we had better drop the heuristics in subset 'C' and those in CH5. These

heuristics need a large amount of time (compared to the others), especially for instances with large number of jobs.

When comparing the running times and the performances of the heuristics selected by *ANOVA analysis excluding those in subset 'C' and those in CH5*, we can see that the subset of heuristics {CDSH1, CDSH1, CDSV1, CDSV2, SF_J} gives a good balance between the running time needed for a heuristic to find a solution and the average deviation of its solution from the lower bound. The *best* and the *least time consuming* subset would be formed with the heuristics in sets 'CH2' and 'J' allowing to finding 84% of best solutions. When adding the heuristic SFR_C to this subset, the percentage goes up to 94%, but for large instances the average running time needed increases dramatically.

9.5 NUMERICAL TEST CONCLUSIONS

Based on the above analysis, the following major conclusions are made:

- A heuristic with the best average deviation does not suffice for always finding the best solutions. One should use a subset of heuristics so as to increase the probability of getting closer to the optimal solution.
- According to the analysis of variance differences in average deviations (up to 4% for non-bottleneck configurations and up 2 to 3% for bottleneck configurations in our case) do not mean that heuristics with smaller values are more robust (always delivering the best results). Some heuristics, though giving different (lower) average deviations, are considered delivering same performances.
- Random scheduling, used for tackling HFS scheduling problems, delivers the worse solutions and, therefore, elaborated heuristics are really worth developing for tackling HFS problems.
- Priority rules do not seem to be suited for HFS scheduling problems (SPT/LPT).
- Local search techniques give good solutions but are outperformed by single stage based heuristics.
- Johnson's idea for scheduling the two-machine flowshop problem when well adapted to the HFS case (see CH2) gives very good results. For the HFS case, we conclude here, like in Elmaghraby and Soewandi (1998a) and (1998b) for the two-stage HFS, that the more successful approach so far is the one using some variant of Johnson's order. However, we also state that this approach does not suffice for finding the best possible solutions.
- Jackson's idea (in CH1), giving higher priority to ready jobs with largest tails or equivalently with largest remaining *work*, also gives good results. And, it gives the best *results* when combined with Johnson's idea (see assignment rule 2 in CH2).
- We can conclude here that, due to the structure of the multistage flowshop with parallel machines (a hybrid flowshop), more elaborate heuristics are required. And, the best heuristic methods for the HFS are the ones using some of the following management guidelines:

    1. Starting and finishing the schedule with jobs having smallest processing times at first and last stages, respectively (based on Johnson's idea, used in CH2).
    2. Giving higher priority to *ready jobs with largest tails* or remaining work (based on Jackson's idea used in CH1 and in assignment rule A2 in CH2).

3. Improving (or optimizing) the makespan locally (at each stage) (based on the shifting bottleneck procedure idea used in CH1).

The first and the second points seem complementary in the sense that the first one minimizes machine idle times at stages 2, 3,…, K, with the shortest jobs at stage 1, while the second by assigning the ready jobs first tries to keep machine idle times at a low level, and by scheduling first the largest tail tries to minimize makespan in the same spirit as EDD (the Earliest Due Date first rule) minimizes the maximum lateness.

Also, the three points show that the best heuristic is the one that has a global view of the shop floor load (CDS in CH2 or the scheduling of the bottleneck stage first and the rescheduling processes in CH1), and while scheduling some stage, pays careful attention both to the local load and to the remaining one at other stages (Jackson's idea in CH1 and in A2).

- For a use in a branch and bound algorithm and at the root node one can use all heuristics based on the single stage subproblem and based on the two-stage flowshop problem, so one can guarantee starting the search with the best possible bound. However, at each node of the search tree, one can use the best and the least time consuming heuristics, that is those in CH2 and in J. Indeed, in our numerical test, these heuristics have allowed finding a large percentage of the best solutions (84%) with an insignificant average running time (see Table 3.9) compared to the case using also the heuristics in class C, i.e. using Carlier's branch and bound as well.

10. CONCLUSION

In this chapter, we presented and compared a large set of heuristics for scheduling a hybrid flowshop (HFS). Five approaches were assessed: single stage based heuristics, two-stage based heuristics, priority rules based heuristics, randomly based heuristics and local search based heuristics. These five approaches correspond to five tested classes of heuristics. Five shop floor configurations were tested: three different configurations without bottleneck stages and two different configurations with bottleneck stages. All configurations have initial release dates and final tails except one. For each of these approaches and configurations, a subset of heuristics was selected by ANOVA analysis.

Only small sets (12 for non-bottleneck configurations and 6 for bottleneck configurations) out of the 31 evaluated heuristics were selected as the best ones to be used without significant loss of good solutions, with respect to the tested configurations.

The best selected heuristics belong only to the first and the second classes of heuristics (the single stage based heuristics and two-stage flowshop based heuristics, respectively). These two classes of heuristics seem to be complementary heuristics in the sense that most of the time (for 85% of instances) the best value is given by either the first or the second class of heuristics. Priority rules are outperformed though the large number of variants tested, and do not seem to be effective for dealing with the HFS scheduling

problem. The same remark goes for randomly scheduling heuristics, they gave the worse solutions when the sequencing and the assignment are done randomly. Local search methods, given the same amount of running time, are outperformed by the single stage based heuristics. The heuristics of the second class (CH2) and those of the first class scheduling the single stage subproblem using Jackson's heuristic (J) are selected as the best and the least time consuming heuristics.

For a use in a branch and bound algorithm and at the root node one can use all heuristics based on the single stage subproblem and based on the two-stage flowshop problem, so one can guarantee starting the search with the best possible bound. However, at each node of the search tree, one should drop the heuristics using Carlier's branch and bound algorithm.

REFERENCES

1. Adams J., E. Balas, D. Zawack, (1988). The Shifting Bottleneck Procedure, Management Science, vol. 34, 391-401, n°3, March 1988.
2. Campbell H. G., R.A. Dudek and M.L. Smith, (1970). A heuristic algorithm for the n job, m machine sequencing problem, Management Science, Vol. 16, n°10, 630-637.
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. Elmaghraby S.E. and H. Soewandi, (1998a). Sequencing Jobs on Two-Stage Hybrid Flowshop with Uniform Machines to Minimize makespan, Department of Industrial Engineering, North Carolina State University, Raleigh, NC, USA (1998a)
5. Elmaghraby S.E. and H. Soewandi, (1998b). Sequencing Jobs on Two-Stage Hybrid Flowshop with Identical Machines to Minimize makespan, Department of Industrial Engineering, North Carolina State University, Raleigh, NC, USA (1998b)
6. 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.
7. Guinet A., M. Solomon, (1996b). Scheduling hybrid flows hops to minimize maximum tardiness or maximum completion time, Int. J. Prod. Res., 34, 1643-1654.
8. Hunsucker J.L., S.A. Brah and D.L. Santos, (1989). Simulation study of a flow shop with multiple processors scheduling, TIMS/ORSA joint National Meeting in New York City, October 16-18.
9. Hunsucker J.L. and J.R. Shah, (1994). Comparative performance analysis of priority rules in a constrained flow shop with multiple processor environment. European Journal of Operational Research 72, 102-104.
10. Jackson, J. R., (1955). Scheduling a production line to minimize maximum tardiness, Research 43, Management Science Research Project, University of California Los Angeles.
11. Johnson S. M., (1954). Optimal two and three stage production schedules with setup time included, Naval Res. Logist., Q.1, 61-68.
12. Kadipasaoglu S. N., W. Xiang and B.M. Khumawalas, (1997). A comparison of sequencing rules in static and dynamic hybrid flow systems, Int. J. Prod. Res., 1997, Vol. 35, n°5, 1359-1384.

13. Matsuo H., C.J. Suh, R.S. Sullivan, (1988). A controlled search simulated annealing method for the general job shop scheduling problem, Working paper #03-04-88, Department of Management, Graduate School of Business, The University of Texas at Austin, Austin, TX 78712.
14. Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, (1953). 'Equation of State Calculations by Fast Computing Machines', Journal of Chemical Physics, 21(1953), 1087-1092.
15. Moursli O., Y. Pochet, (1996). Heuristics for scheduling a Multiprocessor flowshop. Advances in Industrial Engineering Applications and Practice I, Houston Texas (USA), Conference proceeding 456-463.
16. Moursli O., (1997). Branch and bound lower bounds for the hybrid flowshop. Workshop on Intelligent Manufacturing Systems, (IMS'97). Seoul (Korea), Conference proceeding, 107-112.
17. Nawaz M., Jr. E.E. Enscore and I. Ham, (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, OMEGA 11/1, 91-95.
18. 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.
19. Tsubone H., M. Ohba and T. Uetake, (1996). The impact of lot sizing and sequencing on manufacturing in a two-stage hybrid flow shop, Int. J. Prod. Res., 1996, Vol. 34, n°11, 3037-3053.
20. Van Laarhoven P.J.M., E.H.L. Aarts, (1987). Simulated Annealing: Theory and Applications, D. Reidel, Dordrecht, 1987.
21. Van Laarhoven P.J.M., E.H.L., Aarts, J.K. Lenstra, (1988). Job Shop Scheduling by Simulated Annealing, Working paper. Eindhoven University of Technology, The Netherlands.
22. Van Laarhoven P.J.M., E.H.L. Aarts and J.K. Lenstra, (1992). Job shop scheduling by simulated annealing, Operations Research, 40, 1992, 113-125.
23. Wannacott H. T. and R.J. Wannacott, (1990). Introductory Statistics for Business and Economics, 4th edition, John Wiley & Sons, Chapter 10: Analysis of Variance 324-353.

n	M1	M2	M3	M4	M5	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	CDSV2	CDSG1	CDSG2	SPT1	LPT1	SPT2	LPT2	SPT31	SPT32	LPT31	LPT32	SPT41	SPT42	LPT41	LPT42	SR0	SR1	SR2	SLST	SA	***BEST***
20	3	3				9	6	9	5	8	6	8	4	4	***5***	4	12	4	6	17	40	24	18	16	17	39	40	16	17	19	19	120	65	22	8	8	***2***
20	5	5				16	12	15	9	15	13	14	11	10	***13***	10	12	10	12	32	49	41	25	32	32	50	48	32	32	30	27	160	62	34	15	15	***7***
20	8	8				11	9	12	9	16	12	14	11	11	***13***	11	3	10	11	37	41	44	19	37	37	44	41	37	37	26	19	150	48	35	10	10	***7***
30	3	3				6	4	5	2	4	3	4	2	3	***5***	3	8	3	4	10	37	18	12	10	10	36	35	10	10	14	13	110	68	15	6	5	***2***
30	5	5				10	9	10	6	9	8	10	6	7	***10***	7	14	8	8	24	46	32	20	24	24	45	44	24	24	23	20	140	66	27	11	10	***5***
30	8	8				18	15	17	13	17	14	16	13	13	***15***	13	2	12	13	37	57	46	29	37	37	57	54	37	37	33	29	180	68	39	17	16	***10***
50	3	3				4	3	4	2	3	2	3	2	2	***3***	2	5	2	2	7	31	12	9	8	7	31	31	8	7	10	10	110	71	11	4	4	***1***
50	5	5				8	6	7	4	6	5	6	4	4	***6***	4	8	4	5	15	39	21	13	14	14	38	38	14	14	16	14	130	71	18	8	7	***3***
50	8	8				12	10	11	8	9	7	9	7	7	***11***	7	1	8	9	25	47	32	19	25	25	46	45	25	25	23	20	160	68	26	12	12	***5***
100	3	3				1	2	2	1	1	1	1	1	1	***1***	1	2	1	1	4	28	8	5	4	4	27	27	4	4	6	5	100	78	7	2	2	***0***
100	5	5				4	3	4	3	2	2	2	2	2	***3***	2	4	2	2	7	31	11	8	7	7	31	31	7	7	9	8	130	76	10	5	4	***1***
100	8	8				6	6	6	4	4	4	5	4	4	***6***	4	11	4	5	13	36	18	10	13	14	36	36	13	14	13	11	140	74	15	7	7	***3***
20	3	3	3			17	14	14	12	16	15	13	10	9	***8***	13	20	10	9	22	48	32	26	24	21	47	43	24	21	30	25	170	100	28	15	14	***6***
20	5	5	5			23	19	17	16	24	20	19	17	17	***15***	20	9	21	19	38	55	49	32	38	38	57	48	38	38	39	31	190	95	39	18	17	***13***
20	8	8	8			7	6	6	5	8	6	7	5	7	***7***	9	6	9	8	25	28	34	13	25	25	32	26	25	25	19	14	160	44	24	5	5	***3***
30	3	3	3			20	11	11	9	12	12	10	8	5	***5***	8	14	6	5	16	43	24	21	18	15	40	36	18	15	25	19	170	120	22	12	12	***3***
30	5	5	5			20	17	15	13	18	17	15	12	12	***11***	16	21	14	11	28	53	39	28	29	28	52	46	29	28	35	27	200	110	31	16	15	***9***
30	8	8	8			22	18	16	15	23	20	19	17	18	***17***	22	3	19	17	39	55	51	29	40	39	59	49	40	39	39	30	210	93	40	17	17	***13***
50	3	3	3			9	8	8	7	8	9	6	6	3	***3***	5	8	3	4	11	40	18	15	12	10	37	33	12	10	18	14	190	130	15	8	8	***2***
50	5	5	5			14	12	10	9	12	12	10	9	7	***6***	11	14	8	7	20	49	30	22	21	19	46	41	21	19	26	21	200	130	23	12	11	***5***
50	8	8	8			20	18	14	11	16	16	13	12	12	***10***	17	1	13	11	30	58	41	28	31	30	56	48	31	30	35	29	230	110	32	16	16	***8***
100	3	3	3			4	4	4	4	5	6	4	3	2	***1***	3	3	2	1	6	38	11	9	7	5	32	29	7	5	10	8	190	150	10	4	4	***1***
100	5	5	5			8	8	6	6	7	7	6	5	3	***3***	6	5	4	4	11	43	17	13	11	10	36	34	11	10	16	12	210	150	14	7	7	***2***
100	8	8	8			12	12	9	9	9	10	8	7	5	***6***	9	17	7	6	18	50	26	17	18	18	45	40	18	18	22	17	220	140	21	11	11	***4***
20	3	3	3	3	3	26	22	21	19	32	29	22	18	20	***15***	22	15	22	18	27	55	42	34	32	27	57	49	32	27	40	32	250	170	34	20	20	***14***
20	5	5	5	5	5	19	16	16	14	23	20	17	14	17	***14***	19	4	20	18	29	46	43	25	31	28	51	40	31	28	34	24	220	110	31	15	15	***10***
20	8	8	8	8	8	4	3	3	2	5	4	4	3	5	***4***	5	13	7	6	16	22	23	7	17	16	25	20	17	16	15	7	150	45	15	2	2	***1***
30	3	3	3	3	3	22	20	18	16	28	29	17	15	15	***11***	17	19	16	13	22	50	34	30	26	22	50	45	26	22	36	28	290	210	29	18	18	***10***
30	5	5	5	5	5	27	24	18	18	29	28	20	19	21	***16***	24	12	23	19	31	54	44	34	34	31	56	48	34	31	43	36	280	170	36	19	19	***14***
30	8	8	8	8	8	14	13	10	10	16	13	11	10	13	***10***	16	9	16	12	26	42	37	22	28	26	45	35	28	26	32	24	220	100	28	10	10	***8***
50	3	3	3	3	3	17	16	14	13	30	30	13	15	11	***7***	12	14	12	9	16	49	27	23	23	16	45	39	23	16	27	22	320	250	23	15	15	***6***
50	5	5	5	5	5	21	20	16	15	27	26	17	15	17	***11***	19	20	19	14	24	56	36	28	30	24	54	46	30	24	36	29	330	220	30	17	17	***10***
50	8	8	8	8	8	26	25	17	16	28	28	19	18	22	***17***	24	4	25	20	34	59	46	33	36	33	60	48	36	33	44	36	330	170	37	19	19	***15***
100	3	3	3	3	3	10	10	9	8	27	28	10	10	6	***3***	7	7	7	5	8	45	18	16	14	9	38	34	14	9	19	13	350	290	14	9	9	***3***
100	5	5	5	5	5	14	14	11	11	23	24	11	10	9	***5***	10	11	9	6	14	49	23	20	19	14	44	38	19	14	25	19	350	270	19	13	13	***5***
100	8	8	8	8	8	21	20	15	14	23	23	13	13	13	***8***	15	11	16	11	23	55	33	26	27	24	52	43	27	24	33	27	360	240	25	16	16	***8***
						14	***12***	***11***	***9***	***15***	***14***	***11***	***9***	***9***	***8***	***11***	***9***	***10***	***9***	***21***	***45***	***30***	***21***	***23***	***21***	***44***	***40***	***23***	***21***	***26***	***20***	***206***	***124***	***25***	***11***	***11***	***6***

Table 3.3: (**CC1)** non-bottleneck configurations with zero initial release dates and zero final tails.

n	M1	M2	M3	M4	M5	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	CDSV2	CDSG1	CDSG2	SPT11	LPT11	SPT12	LPT12	SPT131	SPT132	LPT131	LPT132	SPT141	SPT142	LPT141	LPT142	SR0	SR1	SR2	SLST	SA	***BEST***
20	3	3				18	11	18	10	28	14	23	12	9	***11***	9	11	26	22	24	35	29	27	34	33	49	46	27	26	38	33	114	75	36	15	14	***6***
20	5	5				18	14	16	13	23	19	19	17	10	***2***	10	2	10	13	23	31	27	25	32	32	50	43	32	32	30	32	150	62	34	15	15	***7***
20	8	8				3	3	11	11	20	18	16	15	11	***6***	11	6	10	12	7	6	8	19	37	37	44	17	37	37	26	27	150	48	35	10	10	***7***
30	3	3				13	7	6	4	9	9	6	6	3	***13***	3	13	3	5	17	35	21	12	10	10	36	42	10	10	14	14	110	68	15	6	5	***2***
30	5	5				22	14	12	10	20	17	15	13	7	***7***	7	7	8	10	29	42	35	20	24	24	45	53	24	24	23	25	150	66	27	11	10	***5***
30	8	8				12	11	17	16	26	26	22	22	13	***3***	13	3	12	15	22	26	24	29	37	37	57	40	37	37	33	36	160	68	39	17	16	***10***
50	3	3				9	5	5	3	8	7	5	4	2	***7***	2	7	2	3	13	32	16	9	8	7	31	36	8	7	10	10	110	71	11	4	4	***1***
50	5	5				14	10	8	7	13	12	9	9	4	***12***	4	12	4	7	20	39	25	13	14	14	38	45	14	14	16	17	130	71	18	8	7	***3***
50	8	8				21	17	12	10	18	17	13	13	7	***2***	7	2	8	11	32	46	37	19	25	25	46	55	25	25	23	27	160	68	26	12	12	***5***
100	3	3				5	3	2	2	5	5	3	3	1	***3***	1	3	1	1	7	29	9	5	4	4	27	30	4	4	6	6	100	78	7	2	2	***0***
100	5	5				8	5	5	4	7	7	5	5	2	***7***	2	7	2	3	12	31	15	8	7	7	31	35	7	7	9	10	130	76	10	5	4	***1***
100	8	8				11	10	8	6	11	11	8	8	4	***11***	4	15	4	6	19	39	23	10	13	14	36	42	13	14	13	15	140	74	15	7	7	***3***
20	3	3	3			21	14	15	12	22	21	14	12	9	***9***	13	11	10	9	27	41	31	26	24	21	47	51	24	21	30	26	170	100	28	15	14	***6***
20	5	5	5			15	12	18	17	31	28	21	20	17	***2***	20	2	21	19	21	26	25	32	38	38	57	36	38	38	39	35	180	95	39	18	17	***13***
20	8	8	8			2	2	6	6	11	10	8	7	7	***8***	9	9	9	9	4	5	6	13	25	25	32	13	25	25	19	15	140	44	24	5	5	***3***
30	3	3	3			16	10	12	9	19	19	13	10	5	***13***	8	17	6	6	20	40	26	21	18	15	40	42	18	15	25	21	170	120	22	12	12	***3***
30	5	5	5			21	16	16	14	26	25	18	16	12	***6***	16	9	14	13	29	42	36	28	29	28	52	49	29	28	35	31	200	110	31	16	15	***9***
30	8	8	8			9	9	17	16	30	28	21	20	18	***4***	22	4	19	18	17	22	21	29	40	39	59	32	40	39	39	38	200	93	40	17	17	***13***
50	3	3	3			13	8	8	7	15	16	9	7	3	***8***	5	9	3	4	15	40	21	15	12	10	37	38	12	10	18	15	180	130	15	8	8	***2***
50	5	5	5			15	12	12	10	21	21	14	10	7	***14***	11	18	8	7	21	43	27	22	21	19	46	43	21	19	26	24	220	130	23	12	11	***5***
50	8	8	8			21	18	16	14	28	26	18	16	12	***2***	17	2	13	12	31	50	38	28	31	30	56	53	31	30	35	33	220	110	32	16	16	***8***
100	3	3	3			6	5	4	3	16	18	6	5	2	***4***	3	4	2	2	8	37	12	9	7	5	32	32	7	5	10	8	190	150	10	4	4	***1***
100	5	5	5			10	7	8	6	15	15	8	7	3	***7***	6	7	4	4	13	42	18	13	11	10	36	36	11	10	16	14	200	150	14	7	7	***2***
100	8	8	8			15	13	11	9	19	19	11	10	5	***15***	9	17	7	7	21	46	28	17	18	18	45	43	18	18	22	21	230	140	21	11	11	***4***
20	3	3	3	3	3	24	18	21	19	35	35	22	19	20	***7***	22	9	22	18	27	42	33	34	32	27	57	47	32	27	40	33	260	170	34	20	20	***14***
20	5	5	5	5	5	12	11	16	14	26	25	17	15	17	***1***	19	1	20	18	17	23	21	25	31	28	51	33	31	28	34	25	220	110	31	15	15	***10***
20	8	8	8	8	8	2	1	3	3	8	6	4	3	5	***13***	5	15	7	6	4	4	5	7	17	16	25	10	17	16	15	7	150	45	15	2	2	***1***
30	3	3	3	3	3	21	17	18	16	38	35	18	16	15	***14***	17	17	16	13	24	43	32	30	26	22	50	44	26	22	36	28	290	210	29	18	18	***10***
30	5	5	5	5	5	21	19	19	19	35	33	20	19	21	***5***	24	5	23	19	28	40	36	34	34	31	56	47	34	31	43	36	280	170	36	19	19	***14***
30	8	8	8	8	8	7	7	11	11	20	18	12	11	13	***8***	16	9	16	12	13	18	18	22	28	26	45	26	28	26	32	24	220	100	28	10	10	***8***
50	3	3	3	3	3	16	13	15	14	43	41	15	13	11	***12***	12	15	12	9	17	44	25	23	23	16	45	41	23	16	27	22	320	250	23	15	15	***6***
50	5	5	5	5	5	21	18	18	16	39	35	18	17	17	***14***	19	17	19	14	25	45	33	28	30	24	54	47	30	24	36	29	330	220	30	17	17	***10***
50	8	8	8	8	8	22	20	19	17	36	36	20	20	22	***4***	24	5	25	20	30	45	38	33	36	33	60	47	36	33	44	36	330	170	37	19	19	***15***
100	3	3	3	3	3	10	9	9	8	47	48	12	10	6	***6***	7	7	7	5	11	43	19	16	14	9	38	35	14	9	19	13	350	290	14	9	9	***3***
100	5	5	5	5	5	13	11	13	11	42	40	12	11	9	***10***	10	12	9	6	16	46	23	20	19	14	44	37	19	14	25	19	350	270	19	13	13	***5***
100	8	8	8	8	8	18	17	16	15	40	40	16	15	13	***8***	15	11	16	11	26	49	32	26	27	24	52	44	27	24	33	27	360	240	25	16	16	***8***
						14	***11***	***12***	***11***	***24***	***22***	***14***	***12***	***9***	***8***	***11***	***9***	***11***	***10***	***19***	***35***	***24***	***21***	***23***	***21***	***45***	***39***	***23***	***21***	***26***	***23***	***205***	***123***	***25***	***12***	***12***	***6***

Table 3.4: (**CC2)** non-bottleneck configurations with non-zero initial release dates and non-zero final tails.

n	M1	M2	M3	M4	M5	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	***CDSV2***	CDSG1	CDSG2	SPT11	LPT11	SPT12	LPT12	SPT131	SPT132	LPT131	LPT132	SPT141	SPT142	LPT141	LPT142	SR0	SR1	SR2	SLST	SA	***BEST***
20	3	2				17	7	17	6	28	11	22	9	9	***9***	9	***9***	27	24	20	33	22	26	30	30	49	45	22	29	40	34	114	80	33	15	14	4
20	5	2				17	11	16	13	23	19	19	17	16	***15***	16	***15***	10	13	24	32	25	25	32	32	50	49	32	14	30	32	150	79	34	15	15	7
20	8	3				5	5	11	11	20	18	16	15	2	***3***	2	***3***	10	12	10	10	10	19	37	37	44	20	37	16	26	27	150	42	35	10	10	7
30	3	2				12	5	6	4	9	9	6	6	6	***6***	6	***6***	3	5	14	29	17	12	10	10	36	37	10	29	14	14	110	78	15	6	5	2
30	5	2				18	10	12	10	20	17	15	13	15	***13***	15	***13***	8	10	24	38	26	20	24	24	45	47	24	31	23	25	150	91	27	11	10	5
30	8	3				16	12	17	16	26	26	22	22	15	***14***	15	***14***	12	15	25	34	25	29	37	37	57	45	37	11	33	36	160	79	39	17	16	10
50	3	2				8	4	5	3	8	7	5	4	4	***3***	4	***3***	2	3	10	30	12	9	8	7	31	35	8	18	10	10	110	83	11	4	4	1
50	5	2				12	8	8	7	13	12	9	9	8	***8***	8	***8***	4	7	16	35	18	13	14	14	38	42	14	31	16	17	130	90	18	8	7	3
50	8	3				16	11	12	10	18	17	13	13	16	***14***	16	***14***	8	11	27	42	27	19	25	25	46	50	25	6	23	27	160	94	26	12	12	5
100	3	2				4	2	2	2	5	5	3	3	2	***2***	2	***2***	1	1	6	27	7	5	4	4	27	29	4	10	6	6	100	84	7	2	2	0
100	5	2				6	3	5	4	7	7	5	5	3	***3***	3	***3***	2	3	9	30	9	8	7	7	31	34	7	17	9	10	130	90	10	5	4	1
100	8	3				9	7	8	6	11	11	8	8	7	***7***	7	***7***	4	6	15	34	17	10	13	14	36	37	13	25	13	15	140	94	15	7	7	3
20	3	2	3			20	13	15	12	22	21	14	12	12	***11***	13	***12***	10	9	24	39	30	26	24	21	47	49	24	29	30	26	170	120	28	15	14	6
20	5	2	5			19	13	18	17	31	28	21	20	11	***11***	10	***11***	21	19	24	36	29	32	38	38	57	46	38	12	39	35	180	98	39	18	17	13
20	8	3	8			5	4	6	6	11	10	8	7	2	***2***	1	***2***	9	9	8	9	10	13	25	25	32	20	25	18	19	15	140	46	24	5	5	3
30	3	2	3			17	11	12	9	19	19	13	10	8	***7***	9	***7***	6	6	17	41	24	21	18	15	40	42	18	31	25	21	170	130	22	12	12	3
30	5	2	5			20	14	16	14	26	25	18	16	11	***11***	12	***12***	14	13	29	45	35	28	29	28	52	50	29	24	35	31	200	130	31	16	15	9
30	8	3	8			15	12	17	16	30	28	21	20	8	***8***	7	***7***	19	18	21	32	27	29	40	39	59	41	40	13	39	38	200	89	40	17	17	13
50	3	2	3			10	6	8	7	15	16	9	7	4	***4***	4	***3***	3	4	13	38	18	15	12	10	37	36	12	22	18	15	180	140	15	8	8	2
50	5	2	5			15	9	12	10	21	21	14	10	7	***6***	7	***6***	8	7	22	46	29	22	21	19	46	45	21	34	26	24	220	140	23	12	11	5
50	8	3	8			19	16	16	14	28	26	18	16	12	***12***	12	***12***	13	12	30	49	37	28	31	30	56	52	31	6	35	33	220	130	32	16	16	8
100	3	2	3			5	4	4	3	16	18	6	5	2	***2***	2	***2***	2	2	7	37	13	9	7	5	32	31	7	11	10	8	190	160	10	4	4	1
100	5	2	5			7	5	8	6	15	15	8	7	3	***3***	4	***3***	4	4	12	39	20	13	11	10	36	36	11	19	16	14	200	160	14	7	7	2
100	8	3	8			12	10	11	9	19	19	11	10	6	***6***	6	***6***	7	7	18	47	27	17	18	18	45	41	18	26	22	21	230	150	21	11	11	4
20	3	3	2	3	3	25	19	21	19	35	35	22	19	17	***14***	19	***16***	22	18	27	41	35	34	32	27	57	48	32	20	40	33	260	170	34	20	20	14
20	5	5	2	5	5	15	13	16	14	26	25	17	15	11	***8***	12	***9***	20	18	19	26	24	25	31	28	51	33	31	8	34	25	220	110	31	15	15	10
20	8	8	3	8	8	2	2	3	3	8	6	4	3	1	***1***	1	***1***	7	6	5	4	6	7	17	16	25	13	17	23	15	7	150	47	15	2	2	1
30	3	3	2	3	3	21	16	18	16	38	35	18	16	15	***11***	16	***13***	16	13	23	43	32	30	26	22	50	45	26	30	36	28	290	210	29	18	18	10
30	5	5	2	5	5	26	21	19	19	35	33	20	19	19	***14***	21	***16***	23	19	29	45	37	34	34	31	56	49	34	16	43	36	280	170	36	19	19	14
30	8	8	3	8	8	7	7	11	11	20	18	12	11	6	***5***	7	***5***	16	12	14	19	19	22	28	26	45	28	28	16	32	24	220	91	28	10	10	8
50	3	3	2	3	3	16	11	15	14	43	41	15	13	10	***7***	11	***8***	12	9	16	43	25	23	23	16	45	39	23	25	27	22	320	240	23	15	15	6
50	5	5	2	5	5	19	16	18	16	39	35	18	17	15	***10***	16	***12***	19	14	23	45	32	28	30	24	54	43	30	33	36	29	330	220	30	17	17	10
50	8	8	3	8	8	22	19	19	17	36	36	20	20	20	***15***	21	***16***	25	20	32	50	39	33	36	33	60	48	36	9	44	36	330	180	37	19	19	15
100	3	3	2	3	3	9	8	9	8	47	48	12	10	6	***4***	7	***5***	7	5	10	42	18	16	14	9	38	33	14	14	19	13	350	290	14	9	9	3
100	5	5	2	5	5	13	11	13	11	42	40	12	11	10	***6***	10	***7***	9	6	15	45	22	20	19	14	44	36	19	23	25	19	350	270	19	13	13	5
100	8	8	3	8	8	18	16	16	15	40	40	16	15	14	***9***	14	***10***	16	11	23	50	30	26	27	24	52	42	27	24	33	27	360	240	25	16	16	8
						14	***10***	***12***	***10***	***24***	***22***	***14***	***12***	***9***	***8***	***10***	***8***	***11***	***10***	***18***	***36***	***23***	***21***	***23***	***21***	***45***	***39***	***23***	***20***	***26***	***23***	***205***	***131***	***25***	***12***	***12***	***6***

Table 3.5: (**CC3)** Configurations with balanced load at each stage by adapting processing time ranges ($r_{j1}>0$, $q_{jK}>0$, $\forall j \in I$)

n	M1	M2	M3	M4	M5	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	CDSV2	CDSG1	CDSG2	SPT1	LPT1	SPT2	LPT2	SPT31	SPT32	LPT31	LPT32	SPT41	SPT42	LPT41	LPT42	SR0	SR1	SR2	SLST	SA	***BEST***
20	3	2				10	6	10	6	12	8	18	8	4	***4***	4	4	16	10	14	21	15	18	18	16	35	31	13	25	29	35	85	58	21	15	14	***3***
20	5	4				17	14	18	12	23	19	22	17	10	***12***	10	12	10	13	26	31	29	28	35	32	51	45	28	9	30	14	120	65	34	1	1	***8***
20	8	7				4	3	4	3	20	18	5	15	1	***3***	1	3	10	12	7	6	8	7	15	15	26	18	10	8	26	17	85	29	14	6	6	***0***
30	3	2				7	5	8	5	9	9	18	6	2	***2***	2	2	3	5	10	18	11	13	12	10	27	24	10	20	14	31	83	59	16	13	12	***1***
30	5	4				14	10	15	10	20	17	21	13	6	***7***	6	7	8	10	22	34	25	27	28	26	47	42	24	27	23	30	130	74	33	11	11	***6***
30	8	7				15	13	16	13	26	26	20	22	7	***9***	7	9	12	15	22	30	25	27	33	32	47	39	27	4	33	12	130	60	33	4	3	***6***
50	3	2				4	3	5	3	8	7	16	4	1	***1***	1	1	2	3	6	13	7	9	8	6	19	18	5	14	10	23	81	59	11	9	9	***1***
50	5	4				10	6	11	7	13	12	18	9	3	***5***	3	5	4	7	14	29	17	21	19	16	39	36	15	26	16	34	120	77	23	15	14	***3***
50	8	7				17	14	18	14	18	17	23	13	6	***9***	6	9	8	11	25	39	27	32	32	31	53	46	28	2	23	6	150	80	34	2	2	***6***
100	3	2				2	2	3	2	5	5	16	3	0	***1***	0	1	1	1	3	9	4	5	5	4	12	11	3	5	6	13	71	58	6	4	4	***0***
100	5	4				5	4	5	4	7	7	15	5	1	***2***	1	2	2	3	8	22	9	11	10	7	28	26	7	12	9	21	110	72	12	9	8	***1***
100	8	7				9	9	10	9	11	11	16	8	2	***5***	2	5	4	6	15	31	17	19	19	17	39	34	14	14	13	24	140	78	20	13	10	***2***
20	3	2	3			14	7	15	7	22	21	20	12	5	***5***	9	7	10	9	19	27	23	21	23	17	39	33	18	23	30	27	130	89	24	13	13	***3***
20	5	4	5			15	12	16	11	31	28	21	20	10	***10***	14	12	21	19	23	31	28	25	31	29	47	40	26	10	39	10	150	76	31	2	1	***7***
20	8	7	8			5	4	5	3	11	10	6	7	1	***2***	1	1	9	9	8	8	9	6	17	16	25	20	10	9	19	17	100	32	14	7	6	***0***
30	3	2	3			9	5	9	5	19	19	16	10	3	***2***	5	3	6	6	13	25	17	15	17	13	31	27	12	25	25	31	130	99	19	15	15	***1***
30	5	4	5			17	13	17	13	26	25	25	16	12	***10***	16	12	14	13	26	39	33	29	34	30	54	47	30	20	35	21	170	100	35	9	8	***8***
30	8	7	8			11	10	12	9	30	28	15	20	6	***6***	8	8	19	18	20	25	23	19	28	27	40	34	22	8	39	13	160	66	27	6	5	***4***
50	3	2	3			7	4	7	4	15	16	15	7	1	***2***	3	2	3	4	9	23	12	12	12	8	23	21	10	16	18	22	140	100	13	10	10	***1***
50	5	4	5			11	8	12	8	21	21	18	10	7	***5***	10	5	8	7	19	39	25	21	25	20	44	37	20	28	26	32	190	120	24	15	15	***4***
50	8	7	8			17	13	18	12	28	26	24	16	13	***11***	18	14	13	12	27	45	34	30	37	35	59	51	31	3	35	6	210	110	36	3	2	***9***
100	3	2	3			3	2	4	2	16	18	14	5	1	***1***	1	1	2	2	5	19	7	6	7	4	15	13	5	8	10	13	140	110	7	6	5	***0***
100	5	4	5			6	5	7	5	15	15	12	7	3	***2***	5	2	4	4	11	32	14	13	14	10	31	26	11	15	16	21	180	130	14	8	8	***2***
100	8	7	8			9	9	10	9	19	19	18	10	5	***5***	10	4	7	7	19	42	24	21	25	21	45	39	20	20	22	26	220	130	24	17	16	***4***
20	3	3	2	3	3	17	13	20	12	35	35	28	19	11	***9***	15	10	22	18	22	34	29	26	29	23	43	37	26	19	40	19	210	140	27	12	12	***8***
20	5	5	4	5	5	15	12	15	12	26	25	16	15	8	***7***	11	10	20	18	17	24	21	20	26	22	40	31	23	6	34	5	180	93	24	1	1	***6***
20	8	8	7	8	8	2	2	2	2	8	6	3	3	1	***1***	1	2	7	6	5	4	5	3	11	10	17	11	7	13	15	20	120	36	9	11	11	***0***
30	3	3	2	3	3	13	9	14	9	38	35	23	16	6	***5***	9	5	16	13	17	29	21	20	23	17	37	28	20	24	36	29	230	170	20	17	17	***3***
30	5	5	4	5	5	17	14	20	12	35	33	26	19	16	***12***	20	16	23	19	26	41	34	29	37	31	53	42	32	14	43	12	250	150	34	6	6	***11***
30	8	8	7	8	8	8	7	8	7	20	18	9	11	5	***4***	7	4	16	12	13	18	18	12	23	21	33	24	17	8	32	14	170	77	19	7	6	***3***
50	3	3	2	3	3	9	6	9	6	43	41	22	13	2	***2***	5	2	12	9	12	27	16	15	17	11	28	23	14	18	27	24	240	190	14	12	12	***1***
50	5	5	4	5	5	15	12	15	13	39	35	23	17	11	***7***	15	8	19	14	23	38	28	25	30	23	46	35	27	31	36	30	280	190	25	17	17	***6***
50	8	8	7	8	8	17	12	20	14	36	36	26	20	19	***14***	23	17	25	20	30	43	37	33	41	37	57	44	38	5	44	8	270	160	37	5	4	***13***
100	3	3	2	3	3	5	4	6	4	47	48	20	10	1	***1***	2	1	7	5	6	22	9	8	9	6	17	14	7	10	19	15	250	210	8	8	7	***0***
100	5	5	4	5	5	10	8	10	8	42	40	19	11	4	***3***	8	3	9	6	14	35	19	17	19	13	37	28	17	19	25	24	310	240	16	13	13	***2***
100	8	8	7	8	8	16	13	16	13	40	40	22	15	12	***8***	16	9	16	11	22	45	29	26	31	26	51	38	27	24	33	27	340	220	27	16	16	***7***
						11	***8***	***11***	***8***	***23***	***22***	***18***	***12***	6	***5***	***8***	***6***	***11***	***10***	***16***	***28***	***20***	***19***	***22***	***19***	***37***	***31***	***18***	***15***	***26***	***20***	***169***	***106***	***22***	***9***	***9***	***4***

Table 3.6: (**CC4)** Bottleneck configurations (by machines) with non-zero initial release dates and non-zero final tails.

n	M1	M2	M3	M4	M5	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	***CDSV2***	CDSG1	CDSG2	SPT11	LPT11	SPT12	LPT12	SPT131	SPT132	LPT131	LPT132	SPT141	SPT142	LPT141	LPT142	SR0	SR1	SR2	SLST	SA	***BEST***
20	3	3				12	7	13	6	15	9	15	7	9	***8***	9	***8***	17	11	17	22	17	20	19	15	37	33	16	13	27	23	96	53	23	10	8	***4***
20	5	5				19	14	16	13	23	14	19	17	10	***11***	10	***11***	10	13	24	29	25	25	38	38	48	42	30	29	39	32	120	62	36	13	12	***7***
20	8	8				4	3	11	3	20	2	16	15	1	***2***	1	***2***	10	12	7	6	7	6	19	19	22	18	11	10	17	13	90	26	17	2	1	***1***
30	3	3				8	5	6	5	9	9	6	6	5	***3***	5	***3***	3	5	13	18	13	15	13	11	26	24	12	10	19	17	81	54	18	6	5	***2***
30	5	5				19	13	12	13	20	19	15	13	13	***13***	13	***13***	8	10	29	40	34	30	42	42	58	53	34	33	44	37	150	79	42	17	16	***10***
30	8	8				13	10	17	10	26	9	22	22	6	***7***	6	***7***	12	15	20	24	23	22	33	33	42	38	25	25	33	26	120	51	30	9	9	***5***
50	3	3				6	3	5	3	8	10	5	4	2	***2***	2	***2***	2	3	8	14	9	10	10	8	19	17	7	5	13	12	81	53	12	5	4	***1***
50	5	5				13	8	8	8	13	22	9	9	6	***7***	6	***7***	4	7	20	38	23	24	29	29	48	44	22	21	33	27	130	80	31	12	11	***5***
50	8	8				21	17	12	16	18	23	13	13	12	***13***	12	***13***	8	11	30	45	37	34	45	45	61	52	34	34	48	38	160	84	43	19	18	***11***
100	3	3				3	2	2	2	5	6	3	3	1	***1***	1	***1***	1	1	4	9	5	6	5	4	12	11	4	3	7	6	74	53	6	3	2	***1***
100	5	5				8	6	5	6	7	18	5	5	2	***3***	2	***3***	2	3	12	33	14	15	17	16	39	36	12	11	19	17	130	84	17	7	7	***2***
100	8	8				11	9	8	9	11	21	8	8	6	***7***	6	***7***	4	6	18	39	22	21	27	27	47	43	21	21	29	23	150	87	27	11	10	***5***
20	3	3	3			15	9	15	8	22	15	14	12	6	***7***	13	***9***	10	9	19	25	22	20	21	17	36	32	19	14	28	24	130	82	23	12	11	***4***
20	5	5	5			18	13	18	13	31	13	21	20	16	***14***	10	***12***	21	19	23	29	23	28	28	23	45	35	25	19	38	29	140	72	28	15	14	***8***
20	8	8	8			3	3	6	3	11	2	8	7	1	***1***	1	***1***	9	9	6	5	6	4	13	12	18	15	8	8	13	8	92	26	13	1	1	***0***
30	3	3	3			11	5	12	5	19	12	13	10	4	***4***	8	***5***	6	6	14	21	19	15	17	14	29	27	14	11	21	18	130	89	18	9	7	***2***
30	5	5	5			16	12	16	12	26	16	18	16	12	***9***	6	***7***	14	13	22	28	22	26	25	19	42	31	23	15	33	26	150	83	25	12	11	***5***
30	8	8	8			11	9	17	9	30	9	21	20	6	***5***	7	***7***	19	18	17	22	21	16	27	27	39	31	22	21	31	19	160	62	27	8	8	***3***
50	3	3	3			7	4	8	4	15	18	9	7	2	***2***	4	***2***	3	4	9	18	12	11	11	8	22	20	9	7	14	12	130	94	12	6	5	***1***
50	5	5	5			11	8	12	8	21	16	14	10	6	***5***	3	***5***	8	7	15	21	16	19	19	13	32	25	15	10	24	20	150	93	18	9	8	***3***
50	8	8	8			25	20	16	20	28	29	18	16	16	***13***	21	***19***	13	12	33	48	39	36	46	45	62	52	37	36	50	35	210	110	44	22	20	***12***
100	3	3	3			4	2	4	2	16	21	6	5	1	***1***	1	***1***	2	2	5	13	7	6	6	5	13	11	5	4	7	6	130	100	8	3	3	***1***
100	5	5	5			7	5	8	5	15	18	8	7	2	***2***	1	***2***	4	4	8	14	8	12	10	8	20	15	8	7	14	12	140	99	10	6	5	***1***
100	8	8	8			15	12	11	12	19	38	11	10	9	***7***	11	***7***	7	7	22	48	28	26	30	30	52	44	24	23	35	26	230	140	29	14	13	***6***
20	3	3	3	3	3	16	10	21	9	35	21	22	19	10	***8***	11	***9***	22	18	21	28	26	21	26	21	37	34	22	18	29	21	190	130	24	13	13	***6***
20	5	5	5	5	5	17	12	16	12	26	16	17	15	10	***9***	14	***12***	20	18	23	28	28	22	28	23	39	32	23	19	32	21	180	89	25	13	13	***7***
20	8	8	8	8	8	3	3	3	2	8	2	4	3	3	***3***	1	***2***	7	6	5	6	6	5	11	9	18	12	8	5	15	7	110	38	9	2	2	***0***
30	3	3	3	3	3	12	7	18	7	38	25	18	16	7	***5***	7	***6***	16	13	18	27	23	18	22	16	33	28	18	13	24	17	210	160	20	10	10	***4***
30	5	5	5	5	5	17	13	19	12	35	21	20	19	9	***8***	16	***12***	23	19	25	31	30	24	29	22	42	32	27	18	34	25	210	120	25	13	13	***7***
30	8	8	8	8	8	11	10	11	9	20	10	12	11	11	***9***	8	***7***	16	12	14	19	17	18	21	17	33	25	19	14	28	17	170	76	20	8	7	***5***
50	3	3	3	3	3	8	4	15	4	43	34	15	13	4	***3***	4	***4***	12	9	12	23	17	13	15	10	24	21	13	10	16	12	230	180	14	7	6	***2***
50	5	5	5	5	5	13	10	18	10	39	24	18	17	5	***5***	10	***6***	19	14	18	27	24	19	23	16	34	26	20	13	27	19	230	160	19	10	9	***4***
50	8	8	8	8	8	18	17	19	16	36	20	20	20	14	***11***	12	***10***	25	20	21	28	21	29	29	21	43	31	26	18	38	29	230	130	25	13	13	***9***
100	3	3	3	3	3	5	3	9	3	47	55	12	10	2	***2***	2	***2***	7	5	7	17	10	7	8	6	14	12	7	6	9	7	230	200	8	4	4	***1***
100	5	5	5	5	5	8	6	13	6	42	35	12	11	3	***2***	4	***2***	9	6	11	20	14	12	14	10	22	16	11	8	15	12	250	180	11	6	5	***2***
100	8	8	8	8	8	13	11	16	11	40	27	16	15	6	***5***	5	***5***	16	11	13	21	13	20	19	13	31	23	16	11	25	21	250	170	16	10	10	***4***
						12	***8***	***12***	***8***	***23***	***18***	***13***	***12***	***7***	***6***	***7***	***6***	***11***	***10***	***16***	***25***	***19***	***18***	***22***	***19***	***34***	***29***	***18***	***15***	***26***	***20***	***157***	***96***	***21***	***9***	***9***	***4***

Table 3.7: (**CC5)** Bottleneck configurations (by processing times) with non-zero initial release dates and non-zero final tails.

		SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	CDSV2	CDSG1	CDSG2	SPT1	LPT1	SPT2	LPT2	SPT31	SPT32	LPT31	LPT32	SPT41	SPT42	LPT41	LPT42	SR0	SR1	SR2	SLST	SA	***BEST***
AVR	CC1	14	12	11	9	15	14	11	9	9	***8***	11	10	10	9	21	45	30	21	23	21	44	40	23	21	26	21	206	123	24	12	11	***6***
	CC2	14	11	12	11	24	22	14	12	9	***8***	11	9	11	10	19	35	24	21	23	21	45	39	23	21	26	23	205	123	25	12	12	***6***
	CC3	14	10	12	10	24	22	14	12	9	***8***	10	***8***	11	10	18	36	23	21	23	21	45	39	23	20	26	23	205	131	25	12	12	***6***
	CC4	11	8	11	8	23	22	18	12	6	***5***	8	6	11	10	16	28	20	19	22	19	37	31	18	15	26	20	169	106	22	9	9	***4***
	CC5	12	8	12	8	23	18	13	12	7	***6***	7	***6***	11	10	16	25	19	18	22	19	34	29	18	15	26	20	157	96	21	9	9	***4***
STD	CC1	7	7	5	5	9	9	6	5	6	5	7	6	7	5	10	9	12	8	10	10	10	8	10	10	10	9	75	67	9	5	5	4
	CC2	6	5	5	5	11	11	6	5	6	4	7	5	7	6	8	12	9	8	10	10	10	10	10	10	11	9	77	67	10	5	5	4
	CC3	6	5	5	5	12	11	6	5	5	4	6	5	7	6	7	11	9	8	10	10	10	9	10	8	11	9	77	63	9	5	5	4
	CC4	5	5	5	5	12	11	6	5	5	4	6	5	7	5	8	11	9	8	10	9	13	11	9	8	10	9	68	55	9	5	5	3
	CC5	6	5	5	5	12	11	6	5	4	4	5	4	7	5	8	11	9	8	11	11	13	12	9	9	11	9	52	45	10	5	5	3
MIN	CC1	1	2	2	1	1	1	1	1	1	1	1	1	1	1	4	22	8	5	4	4	25	20	4	4	6	5	100	44	7	2	2	0
	CC2	2	1	2	2	5	5	3	3	1	1	1	1	1	1	4	4	5	5	4	4	25	10	4	4	6	6	100	44	7	2	2	0
	CC3	2	2	2	2	5	5	3	3	1	1	1	1	1	1	5	4	6	5	4	4	25	13	4	6	6	6	100	42	7	2	2	0
	CC4	2	2	2	2	5	5	3	3	0	1	0	1	1	1	3	4	4	3	5	4	12	11	3	2	6	5	71	29	6	1	1	0
	CC5	3	2	2	2	5	2	3	3	1	1	1	1	1	1	4	5	5	4	5	4	12	11	4	3	7	6	74	26	6	1	1	0
MAX	CC1	27	25	21	19	32	30	22	19	22	17	24	21	25	20	39	59	51	34	40	39	60	54	40	39	44	36	360	290	40	20	20	15
	CC2	24	20	21	19	47	48	23	22	22	15	24	18	26	22	32	50	38	34	40	39	60	55	40	39	44	38	360	290	40	20	20	15
	CC3	26	21	21	19	47	48	22	22	20	15	21	16	27	24	32	50	39	34	40	39	60	52	40	34	44	38	360	290	40	20	20	15
	CC4	20	19	20	19	47	48	28	22	19	14	23	17	25	20	30	45	37	33	41	37	59	51	38	31	44	35	340	240	37	17	17	13
	CC5	25	20	21	20	47	55	22	22	16	14	21	19	25	20	33	48	39	36	46	45	62	53	37	36	50	38	250	200	44	22	20	12
%	CC1	2	7	6	18	2	11	6	20	13	29	12	12	10	14	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	8	9	100
B	CC2	5	17	5	15	2	6	4	11	29	48	26	28	4	4	2	1	1	1	1	1	1	1	0	1	1	2	0	0	1	13	14	100
E	CC3	4	20	3	12	1	5	3	9	19	39	20	28	1	2	1	1	1	1	0	0	0	0	0	1	0	1	0	0	0	10	12	100
S	CC4	3	9	3	10	1	4	1	6	39	36	28	27	3	3	1	1	1	1	0	0	0	0	0	1	0	1	0	0	1	10	13	100
T	CC5	4	12	3	13	2	6	3	11	29	32	33	25	2	3	1	1	1	1	0	0	0	0	0	2	1	1	0	0	1	11	13	100
%O	CC1	0	3	1	6	0	5	2	5	1	20	1	6	3	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
B	CC2	0	7	0	2	0	2	0	1	4	28	1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	
E	CC3	0	11	0	1	0	2	0	1	4	23	5	12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	
S	CC4	0	1	0	1	0	0	0	2	11	21	2	12	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	
T	CC5	0	3	0	3	0	1	0	3	7	16	11	10	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	

Table 3.8: Comparing Average, Standard deviation, Minimum and Maximum of values (percentage of average deviation over 50 instances and over all configurations in CCi, i=1, ..., 5) giving by each heuristic as well as the percentage that a heuristic has gave the best value (%BEST) and the percentage that a heuristic was the only one giving the best value (%BEST).

	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	CDSH2	CDSV1	CDSV2	CDSG1	CDSG2	SPT1	LPT1	SPT2	LPT2	SPT31	SPT32	LPT31	LPT32	SPT41	SPT42	LPT41	LPT42	SR0	SR1	SR2	SLST	SA
AVR	1	48	2	104	2	110	2	145	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	70	72
2-20	0	8	0	10	0	11	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	6
2-30	1	14	0	17	0	18	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	13
2-50	1	25	1	31	1	31	1	34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	25
2-100	2	55	1	67	1	69	1	75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	72	72
3-20	1	11	0	17	0	17	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	8
3-30	1	19	1	29	1	32	1	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	18
3-50	1	33	1	52	1	61	1	66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	36
3-100	3	108	3	183	3	211	3	239	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	108	111
5-20	1	15	1	28	1	31	1	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	13
5-30	1	28	2	53	2	60	2	71	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	29	29
5-50	2	61	3	130	3	147	4	187	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	61
5-100	5	189	8	431	9	491	11	584	1	2	1	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	177	181

Table 3.9: Average time (in seconds on a Pentium II 350 MHz) needed for running 50 instances

		Max. Deviation Tolerated By rule 2	SF_J	SF_C	SFR_J	SFR_C	SBR_J	SBR_C	SBP_J	SBP_C	CDSH1	***CDSH2***	CDSV1	CDSV2	CDSG1	CDSG2	SPT1	LPT2	SPT32	SPT42	LPT42	SLST	SA	***BEST***
	CC1	23	14	12	11	9	15	14	11	9	9	***8***	11	10	10	9	21	21	21	21	21	12	11	***6***
A	CC2	20	14	11	12	11	~~24~~	~~22~~	14	12	9	***8***	11	9	11	10	19	~~21~~	~~21~~	~~21~~	~~23~~	12	12	***6***
V	CC3	20	14	10	12	10	~~24~~	~~22~~	14	12	9	***8***	10	***8***	11	10	18	~~21~~	~~21~~	20	~~23~~	12	12	***6***
R	CC4	17	11	8	11	8	~~23~~	~~22~~	~~18~~	12	6	***5***	8	6	11	10	16	~~19~~	~~19~~	15	~~20~~	9	9	***4***
	CC5	18	12	8	12	8	~~23~~	18	13	12	7	***6***	7	***6***	11	10	16	18	~~19~~	15	~~20~~	9	9	***4***

Table 3.10: Remaining heuristics after applying discrimination rules 1 and 2

		F For All H.	Crit. F For All H.	F For Bold H.	Crit. F For Bold H.	***Max. Deviation Tolerated By ANOVA***	SF_J	SF_C	SFR_J	SFR_C	SBP_J	SBP_C	CDSH1	*CDSH2*	CDSV1	CDSV2	CDSG1	CDSG2	SLST	SA
	CC1	16.9	1.68	1.41	1.97	*12*	14	**12**	**11**	**9**	**11**	**9**	**9**	***8***	**11**	**10**	**10**	**9**	**12**	**11**
A	CC2	7.5	1.89	1.93	2.08	*12*	14	**11**	**12**	**11**	14	**12**	**9**	***8***	**11**	**9**	**11**	**10**	**12**	**12**
V	CC3	7.35	1.89	1.95	2.08	*12*	14	**10**	**12**	**10**	14	**12**	**9**	***8***	**10**	***8***	**11**	**10**	**12**	**12**
R	CC4	11.1	1.89	2.35	2.63	*8*	11	**8**	11	**8**	~~18~~	12	**6**	***5***	**8**	**6**	11	10	9	9
	CC5	14.1	1.86	2.13	2.46	*8*	12	**8**	12	**8**	13	12	**7**	***6***	**7**	***6***	11	10	9	9

Table 3.11 : Selected heuristics by ANOVA (The corresponding average deviation is marked bold)

	CH1	CH2	CH3	CH4	CH5	CH1_O	J_O	C_O	CH2_O	H_O	V_O	G_O	A1_O	A2_O	CH3_O	CH4O	CH5_O	LS_O	SA_O	CH1or CH2	CH1& CH2	Less	ANOVA	ANOVA + Less	ANOVA - CH5
CC1-2s	58	44	2	0	8	48	1	21	39	0	0	8	7	1	1	0	2	0	2	87	97	51	97	98	95
CC1-3s	40	66	1	0	10	27	1	18	60	26	11	11	8	40	0	0	1	0	1	86	99	74	90	92	89
CC1-5s	36	68	1	1	10	25	6	18	62	38	10	12	1	58	0	0	2	0	1	87	98	79	89	93	87
CC1-all	44	59	1	0	9	33	3	19	53	21	7	10	5	33	0	0	1	0	1	87	98	68	92	94	90
CC2-2s	37	64	4	1	20	25	0	17	55	0	0	0	0	0	0	0	5	1	3	80	95	71	100	100	76
CC2-3s	28	76	4	0	13	17	0	9	68	43	19	1	12	50	0	0	2	0	1	86	98	82	96	98	82
CC2-5s	24	80	5	1	13	14	1	11	71	52	14	1	2	64	1	0	3	1	1	85	97	84	97	98	84
CC2-all	30	73	4	1	15	19	1	12	65	31	11	0	5	38	0	0	3	1	1	84	97	79	98	98	80
CC3-2s	51	41	3	0	20	44	0	29	34	0	0	0	0	0	1	0	10	1	3	79	89	51	98	99	88
CC3-3s	21	80	2	1	8	16	0	8	74	27	31	0	23	35	0	0	2	0	1	90	98	82	98	98	96
CC3-5s	17	85	4	0	11	10	0	7	77	53	18	2	3	70	0	0	3	0	1	88	97	87	98	98	94
CC3-all	30	69	3	0	13	23	0	15	62	27	16	1	9	35	1	0	5	1	2	85	94	73	98	98	93
CC4-2s	15	82	5	0	12	10	0	3	74	0	0	0	0	0	2	0	5	0	2	84	93	85	98	98	93
CC4-3s	17	81	3	0	14	11	0	5	73	45	22	0	23	44	1	0	6	0	2	84	94	83	99	99	93
CC4-5s	15	85	4	1	12	9	0	5	77	50	20	1	14	58	0	0	5	0	1	85	95	87	99	99	94
CC4-all	16	83	4	1	13	10	0	4	75	31	14	1	12	34	1	0	5	0	2	85	94	85	99	99	93
CC5-2s	32	65	5	1	17	23	0	12	57	0	0	1	0	1	1	0	6	1	3	80	93	72	96	97	91
CC5-3s	20	83	4	1	12	12	0	9	74	32	36	1	36	33	0	0	3	1	1	86	97	85	99	99	96
CC5-5s	25	74	3	1	12	18	2	12	67	38	25	2	17	48	1	0	5	1	2	85	95	79	99	99	94
CC5-all	26	74	4	1	14	18	1	11	66	23	20	1	17	27	1	0	4	1	2	84	95	79	98	98	94
Average	***29***	***72***	***3***	***0***	***13***	***21***	***1***	***12***	***64***	***27***	***14***	***3***	***10***	***33***	***1***	***0***	***4***	***0***	***1***	***85***	***96***	***77***	***97***	***98***	***90***

Table 3.12: Percentages that the best value is given by a subset of heuristics