Branch-and-Bound Methods for the Multi-Processor Job Shop and Flow Shop Scheduling Problems

Michael Perregaard
Department of Computer Science
University of Copenhagen
Universitetsparken 1
DK-2100 Copenhagen East
e-mail: perre@diku.dk

June 29, 1998

Abstract

The Multi-Processor Job Shop and Flow Shop scheduling problems are variants on the standard Job Shop and Flow Shop problems in which each machine has the capacity to process more than one job simultaneously.

This class of problems has attracted the attention of researchers in recent years and several lower bounds has been proposed for a Branch-and-Bound solution method. But only a few attempts has been made to develop a complete Branch-and-Bound scheme in which these lower bounds can be used.

In this paper, which is based on [16], we present three different Branch-and-Bound methods, where the first two are similar to existing methods proposed in [7] and [4] and the third is our new proposal. To completely test the Branch-and-Bound methods, some heuristic methods to build initial solutions (based on the Shifting Bottleneck procedure [1] and Simulated Annealing), different branching strategies and the known lower bounds has been implemented. These lower bounds include the $O(n^2)$ SubSet Bound and the Adjusted SubSet Bound proposed by A. Vandevelde et al. [18] and an improved $O(mnlogn)$ Jackson's Pseudo Preemptive Schedule proposed by J. Carlier and E. Pinson. In addition a test suite of 252 problems has been generated for the purpose of testing the methods.

The results show that, within 60 seconds of computation, the two first Branch-and-Bound methods based on existing, published methods

are able to solve resp. 140 and 159 of the 252 test problems, whereas our proposed Branch-and-Bound method is able to solve 209 of the 252 test problems.

1 Introduction

In the Multi-Processor Job Shop Scheduling (MPJSS) problem n jobs are to be processed on c distinct centers. Each job is composed of a sequence of non-interruptable operations of different length to be processed in the c centers. Each center $j = 1, \ldots, c$ contains m_j identical machines where each machine is able to process one operation at a time and once an operation is started it must be processed to completion without interruptions. It is assumed that all jobs are available at time zero. In our project we then consider the problem of finding an optimal schedule in each center such that the overall processing time is as short as possible (minimize the makespan). Any such schedule must resolve which operations to place on each machine and determine a processing sequence of the operations placed on the same machine.

An equivalent formulation of the MPJSS problem is that n jobs are to be scheduled on m machines where each machine $j = 1...c$ is able to process m_j non-interruptable operations simultaneously. If $m_j = 1$ for all j then the MPJSS problem is identical to a standard Job Shop problem.

The Flow Shop differs from the Job Shop in that the structure of the jobs is fixed. In a Flow Shop every job must visit each center exactly once and in the same order, that is, every job must first be processed in center one, then every job proceeds to center two, etc. The only difference between the jobs is in the processing times, as two jobs can require different amounts of processing in the same center.

Since the standard Flow Shop problem is $\mathcal{NP}$-hard [11] it also follows that both the standard Job Shop as well as the Multi-Processor Job Shop and Flow Shop problems are $\mathcal{NP}$-hard as the standard Flow Shop is a special case of each of these problems. Hoogeven et al. [13] have shown that the Multi-Processor Flow Shop is $\mathcal{NP}$-hard with as few as two centers, even if preemption is allowed.

This paper is based on the work in [16] and is meant to present the most important results from this work. This paper is organized such that the three tested branch-and-bound schemes are presented in section 2 followed by a short introduction in section 3 of the lower bound functions tested

in each of the three branch-and-bound schemes. Finally, in section 4 we present computation results showing the efficiency of various combinations of the branch-and-bound schemes with lower bounds.

2 Branch-and-Bound schemes

Presently, the only successful optimal solution methods for the standard Job Shop and Flow Shop problems are based on Branch-and-Bound methods, and the optimal solution methods presented in this project are also all based on Branch-and-Bound.

In this project, Branch-and-Bound is performed by representing all feasible schedules in each subtree of the search tree through a partial schedule. In a partial schedule, it is not unambiguously determined which machine must process each operation or the processing order of operations on the same machine. Thus a partial schedule represents all the feasible schedules that can be created by completing it. Branching is then performed by imposing one or more constraints onto the partial schedule.

When building a complete Branch-and-Bound solution method there are a certain number of points that must be addressed:

Initial solution The better the initial solution is before the Branch-and-Bound search commences, the earlier in the search are the bounding procedures able to determine if a better solution does not exists below a given node.

In this project we have used both very simple construction methods and methods based on the Shifting Bottleneck procedure [1] and Simulated Annealing [15] to provide a good initial solution.

The construction of an initial schedule from scratch is based on a simple dispatch scheduling of each center. As operations become available a priority is assigned to them and the available operations having highest priority number are processed first by the available machines. This method is used both as a mean of creating a complete schedule fast by simply scheduling each center in turn and is also used in the Shifting Bottleneck procedure. For further information on the priorities used and a detailed description of the implemented Shifting Bottleneck procedure, please refer to [16].

The Simulated Annealing procedure is based on local rearrangements of critical paths in a complete schedule. The procedure is similar to

that used in [15] for the standard Job Shop problem.

Representation How a partial schedule is represented is closely tied to the constraints we wish to impose. In the following sections we present three fundamentally different representations, each with its own strength and weaknesses.

Lower bound The bounding procedures will be presented in section 3.

Branching rules The underlying rules of which constraints to impose on the partial schedules can have a great impact on the number of times a partial schedule have to be bounded. In each representation, two different branching schemes are considered: In the first, we simply impose a single new constraint allowed by the representation, and in the second we use the notion of *Critical Paths* introduced by Grabowski et al [12] and later refined by Brucker et al. [5] for the standard Job Shop problem. At the end of each section on the representations we briefly outline how branching is performed.

Search order When the initial solution is not optimal, the selection of the next node to expand influences when a better solution can be found and thus influences the overall computation. In this project several search orders are considered: Depth-first, breadth-first and priority based searches, where highest priority are given to either least lower bound, least upper bound or an average of the last two. The upper bound is created by completing the partial scheduling using a simple heuristic.

2.1 Heads and Tails

As the notion of heads and tails are very important when solving scheduling problems, we will here briefly describe our use of heads and tails.

The head of an operation i, denoted h_i is a lower bound on the processing time needed before processing of operation i can commence. Likewise, the tail of an operation i, denoted t_i, is a lower bound on the processing time needed after operation i has completed to the completion of all jobs.

2.2 Intervals Representation

The intervals methods was originally proposed by j. Carlier [7] to solve the parallel machine problem. In the parallel machine problem, n operations

are to be processed non-preemptively on m identical machines, with each operation having a release time, a processing time and a due time. This problem is exactly what we get in an MPJSS if we focus on a single center and relax the capacity constraint in the remaining centers. Carlier's method can thus be used to represent an MPJSS by representing each of the centers as a parallel machine problem.

The intervals representation consists in all its simplicity of assigning a time interval to each operation i, defined by a release time r_i and a due time d_i. Each operation must then be completely processed within its time-interval.

Branching a schedule consists of splitting time intervals, such that the interval of allowed release times, given as $[r_i, d_i - p_i]$, is divided into two disjoint, equally sized halves.

A complete and unambiguous solution is reached when the number of overlapping time intervals never exceed the number of machines, in each center.

The two tested branching schemes are as follows:

single The operation i to be branch on is selected as the one have minimal sum of $h_i + p_i + t_i$.

critical path Branching on a critical path in the Intervals Representation only means that the operation selected for branching lies on a critical path in a completion of the partial schedule. The selection of the operation follows the branching strategy by J. Carlier [7].

2.3 Strings Representation

The goal when building a complete schedule in each of the multi-processor centers, is to determine which operations must be processed together on the same machine and to determine the processing order of operations on the same machine. To build a schedule of n operations in a center with m identical machines, we have to split the operations into m sets and sequence each of the sets independently. Thus, the complete schedule should consist of m sequences of operations.

In the Strings Representation we deal solely with strings of operations. The initial and totally unscheduled situation is that n strings, each composed of a single operation, should be joined in a manner, such the final result is at most m individual strings. The only allowed manipulations are

thus to join two strings to reduce the number of strings. By joining we mean appending one string to the end of another. To allow for an efficient branching strategy, we will also include the possibility that a string cannot be preceded or succeeded by another string, i.e., it must be processed respectively first or last on a machine.

If we concentrate on a single center, e.g. when calculating a bound, then whole strings share the special property with single operations that they are unbreakable, i.e., no other string or operation can interrupt the processing of a string on the same machine. Thus, with respect to bounding a single center of strings, each string can be considered as a single operation having the head of the first operation, the total processing time of all operations in the string, and the tail of the last operation. This helps to reduce the time to compute the bounds.

If strings are joined in a certain order such that the complete string of operations to be processed on the first machine is build first, the complete string of operations to be processed on the second machine is build next, and so on, the second representation becomes identical to a branch-and-bound method proposed by S.A. Brah and J.L. Hunsucker [4].

The two tested branching schemes are as follows:

single A candidate string for branching is selected such that the number of feasible branches created when joining the string to any of the remaining strings in the center is minimized.

critical path Strings are joined such that a critical path in a completion is rebuild in the partial schedule. Each time two strings s_1 and s_2 are joined as $s_1 - s_2$, new branches are created where s_1 is joined with each of the remaining strings in the respective center.

2.4 Jobshop Representation

The third and final representation is an attempt to expand the most successful Branch-and-Bound solution method from the standard Job Shop problem, to include the MPJSS. Hence the name "Jobshop Representation".

The success of recent Branch-and-Bound methods for the standard Job Shop problems lies in the ability of simple tests to identify precedence relations between operations pairwise, that must be present in any optimal schedule. Such tests were originally identified by J. Carlier and E. Pinson [8] and named *Immediate Selection Rules*. Later they presented efficient

methods to improve the computation of these immediate selection rule based on Jackson's Preemptive Schedule (in [9] an $O(n^2)$ method and in [10] an $O(n \log n)$ method), which is an optimal schedule for the preemptive one-machine sequencing problem. The presence of such immediate selection rules can speed up the Branch-and-Bound search-time about 10 times or more.

For the immediate selection rules of Carlier and Pinson to be applicable on operations in a multi-processor problem it is necessary that operations are first restricted to be processed on only one specific machine. In the Jobshop Representation we introduce in each center $j = 1, ...c$ a set of Boolean attributes $b_{i,j,k}$, $i = 1, ..., n_j$ and $k = 1, ..., m_j$, where $b_{i,j,k}$ is *true* if and only if operation i is allowed to be processed on machine k in center j. Initially $b_{i,j,k} = true$ for every combination of i, j and k as the machines in every center are identical. Then through Branch-and-Bound and a few simple tests (one might call them immediate selection rules) certain operations can be disallowed from being processed on certain machines by setting the corresponding $b_{i,j,k}$ to *false*. Note that since the machines are identical there exists a large amount of symmetric solutions with the only difference being the numbering of machines. All symmetric solutions, except one, can immediately be discarded as we are only interested in *one* optimal solution. In [16] is described how such symmetric solutions can be identified and eliminated.

When two or more operations are allowed to be processed by only one and the same machine $k \in \{1, ..., m_j\}$ in a center j the immediate selection rules of Carlier and Pinson can be applied to these operations. In our project we have chosen to use the $O(n^2)$ method to compute the rules, since the number of operations on which we apply the rules are often very small and the $O(n \log n)$ method is only competitive on larger numbers of operations.

The pairwise precedence constraints between operations are represented according to the disjunctive graph model used when representing most standard Job Shop problems and are described in both [8] and [5]. In this graph, nodes signify operations and oriented edges between nodes correspond to precedence constraints between the corresponding operations.

The two tested branching schemes are as follows:

single Branching by inserting a single new constraint in the Jobshop Representation is performed, such that each operation is first restricted to be by processed by only one machine and then unresolved presedence constraints are added. That is, in each branching step we look for an operation i in center j with $C(i, j) = |\{k \in \{1, ..., m_j\} | b_{i,j,k} = true\}| > 1$.

If several candidates exists, we select the pair (i, c) of minimal $C(i, c)$ and on a tie the operation having maximal $h_i + p_i + t_i$. If no candidate exists we proceed instead by inserting a precedence constraint according to the scheme used by Carlier and Pinson [8] and Applegate and Cook [3].

critical path Branching on critical paths in the Jobshop Representation follows closely the proposed branching strategy by Brucker et al [5] for the standard Job Shop problem. This method has been extended to the multi-processor case such that operations in a block on a critical path is first restricted to be processed on a single machine before precedence constraints are added.

3 Lower Bounds

The lower bound functions tested are all designed to bound a single center and differ mostly in the amount of machine-information they take into consideration. Thus the problem considered in all of the lower bound functions is that of non-preemptively scheduling n operations, with release time, processing time and due time, on m identical machines.

In the following list we describe each of the lower bounds that has been implemented and tested in this project. For each lower bound we specify the worst-case time-complexity (**WCTC**) and the best-case time-complexity (**BCTC**). When specifying the time-complexities, we let n denote the number of operations and m the number of machines in the center to be bounded.

Short	**Description**	**WCTC**	**BCTC**
JB	*Job Bound*	$O(n)$	$\Omega(n)$
	The maximum of $h_i + p_i + t_i$ over all operations.		
JPS	*Jackson's Preemptive Schedule*	$O(n \log n)$	$\Omega(n \log n)$
	JPS is an optimal schedule in the case of scheduling preemptively operations with release time, processing time and due time on a single machine. JPS is build by scheduling at each time operations are available, the operation having longest tail.		
	JPS was originally proposed as a lower bound for the one-machine sequencing problem [6] and was later adopted as the de-facto lower bound for the standard Job Shop problem.		

SSB *SubSet Bound* $O(n^2)$ $\Omega(n^2)$

A lower bound on any preemptive schedule of a set of operations J on m machines is the Set Bound:

$$SB(J,m) = \sum_{i \in H(J,m)} h_i + \sum_{i \in J} p_i + \sum_{i \in T(J,m)} t_i$$

Here $H(J,m)$ ($T(J,m)$) denotes the m operations in J having smallest head h_i (tail t_i). The SubSet Bound is then the maximum of the Set Bound taken over all subsets of the n operations I:

$$SSB(I,m) = \max_{J \subseteq I, |J| \geq m} SB(J,m)$$

This is the common lower bound when dealing with multi-processor problems. The tested $O(n^2)$ implementation is similar to the one proposed by Vandevelde et al [18].

ASSB *Adjusted SubSet Bound* $O(2^{2m}m^2)$ $\Omega(1)$

The Adjusted Set Bound (ASB) is an adaption of the Set Bound (SB) to the case when preemption is not allowed. If an operation cannot be interrupted then it is both the first and last to be processed on a machine if and only if it is the only operation on that machine. Thus when preemption is not allowed we should only use both the head and the tail of an operation in the Set Bound if we assume that the operation is processed alone on a machine. The Adjusted SubSet Bound is the ASB computed on a subset of operations for which SB is maximal. This subset is provided by the SubSet Bound which must thus always be computed before ASSB.

The ASSB was first presented in [17] with an $O(2^{2m}m^2 \log m)$ implementation to compute. Later A. Vandeveldeet et al [18] claimed that the ASSB can be computed in $O(2^{2m}m^2)$ time. For a description of the $(2^m m^2)$ implementation used in this paper, please refer to [16].

JPPS *Jackson's Pseudo Preemptive Schedule* $O(mn \log n)$ $\Omega(n \log n)$

JPPS is an attempt to adapt Jackson's Preemptive Schedule to the multi-processor case. When building JPPS an operation can be processed on any (fractional) number of machines but

the total any operation has been processed at any time must never exceed the maximum amount it could have been processed by a single machine. That is, we require at any time t that $h_i + p_i^-(t) \leq t$, where $p_i^-(t)$ is the processed amount of operation i at time t. Available operations are selected for processing based on a priority rule where highest priority is given to the largest sum of $p_i^+(t) + t_i$ where $p_i^- + p_i^+ = p_i$. The length of JPPS is equal to $\max\{JB, SSB\}$.

The JPPS is due to J. Carlier et al, who presented an $O(m^2 n \log n)$ method to compute it at a conference in May 1995. To the knowledge of the present author, no material has yet been published on JPPS. For a description on the $O(mn \log n)$ method to compute JPPS used in this paper, please refer to [16].

MaxFlow *Maximum Flow - Intervals & String Rep.* $O(n^4)$ $\Omega(n \log n)$

The MaxFlow Bound is a graph formulation of the problem of preemptively scheduling n operations on m machines, first proposed by W.A. Horn [14]. The maximum flow through this graph indicates whether or not a feasible preemptive schedule exists with a length less than or equal to a trial number.

The implemented method to solve the maximum flow problem in the graph is based on the Layered Network approach [2]. This method has a worst-case time complexity of $O(n^2 e)$ (n nodes and e edges) whereas methods based on the Preflow/Push approach can attain worst-case time complexities of at most $O(n^3)$. Nevertheless, tests on multi-processor problems [16] indicates that the Layered Network approach is faster in practice.

Maximum Flow - Jobshop Rep. $O(n^6 m)$ $\Omega(n(m + \log n))$

In the third representation the graph on which the MaxFlow bound is computed is modified to take into account that certain combinations of operations and machines are not allowed. This strengthens the MaxFlow Bound against the other bounds since none of the other bounds can take this information into consideration.

For a description on how to adapt the MaxFlow bound to the third representation, please refer to [16].

4 Results

In this section we present computational results to examine the practical performance of the three presented representations when used to solve a set of test problems using various combinations of initial solutions, lower bounds, immediate selection rules, etc.

All test were carried out on a HP Apollo 9000 series 735 workstation running the UNIX operating system.

4.1 Test problems

For the purpose of testing our Branch-and-Bound implementations, a test suite of 252 problem instances has been created. These test problems consists of all combinations of the following parameters:

- *Job Shop* or *Flow Shop* structure.
- 2, 5 or 10 centers.
- 5, 10, 15 or 20 jobs.
- An average of $a = 1$, 2, 3 or 4 machines in each center. The number of machines is drawn randomly from the interval $[1, 2a - 1]$.
- 0, 5, or 10 percent of the operations having large processing times. Normal processing times are drawn randomly from the interval $[5, 50]$ whereas high processing times are drawn randomly from the interval $[50, 150]$.

Excluded from the test suite are all problems with only 5 jobs and an average of 3 or 4 machines.

When testing the various solution methods we will often use the average of the best solutions found to all of the 252 test problems as a measure of performance. During all of the testing only 220 of the 252 problems were solved to optimality, such that the best solution found was also proven to be optimal. It is thus not possibly to give the average of the optimal solutions to all test problems, but as a reference to compare the test results against we can inform that the average of the best known solutions is: **463.24**.

4.2 Test output and notation

For each of the test runs we note the following measures of performance:

- $|S_i|$ The length of the initial constructed schedule.
- $|S_l|$ The length of the best schedule found by applying local search (search for local minimum or Simulated Annealing) to the initial schedule .
- $|S_o|$ The length of the best schedule found during the Branch-and-Bound search. Note that the search can be time-limited such that the found schedule is not necessarily optimal.
- n_o Number of nodes bounded during the Branch-and-Bound search.
- t_i The elapsed time (in seconds) for bulding the initial schedule.
- t_l The elapsed time (in seconds) spent during local search.
- t_o The elapsed time (in seconds) spent during Branch-and-Bound search.
- o *True* if the schedule returned was proven to be optimal and *false* otherwise.

To describe the results we will make use of the following additional notation:

- $\#P$ The number of problem instances for which P is *true*.
- $\overline{v}$ The average of v taken over all 252 problem instances.

4.3 Initial solutions

In table 1 we report on the quality and computation times to create the initial solution needed for a subsequent Branch-and-Bound search. Finding an initial solution consists of two parts: 1) constructing a schedule from scratch and 2) improving the constructed schedule. In the table we show horizontally two different methods to construct an initial solution from scratch: the simple construction method, where each center is completely schedule once in turn, and the Shifting Bottleneck procedure, where already scheduled centers are rescheduled, as new centers become scheduled. When scheduling a single center we either make it (near) optimally using a time-limited (5 seconds) Branch-and-Bound search or use a dispatch rule where highest

	Simple		Shifting Bottleneck	
	$\overline{\lvert S \rvert}$	$\overline{t}$	$\overline{\lvert S \rvert}$	$\overline{t}$
Initial solution - opt.	504.5	4.21	484.5	20.00
Initial solution	508.3	0.01	492.3	0.05
Local minimum	496.4	0.01	487.7	0.01
Simulated Annealing	476.8	2.54	475.2	2.45

Table 1: Quality of initial schedules and the improvement gained through the application of local search methods.

priority is given to the operation having largest tail. The results are shown in the first two lines of table 1.

The constructed schedule is subsequently improved by local search methods. We have tested two possibilities. One where we only search for a schedule whose length is local minimum with respect to the allowed manipulations, and one based on Simulated Annealing which allows us to move out of a local minimum. We initiate the local search from the schedule constructed using dispatch rules. The table shows that a local minimum is found very fast, but does not improve on the initial schedule very much. If we on the other hand allow the search to move out of a local minimum, the quality of the found solution is greatly improved. Since the Simulated Annealing procedure is not very dependent on the initial schedule, as reflected in the table, the simplest method for constructing a schedule from scratch can be used.

Compared against the Shifting Bottleneck procedure the table shows the Simulated Annealing procedure to be better than the Shifting Bottleneck procedure even when centers are scheduling optimally. For this reason all the following Branch-and-Bound tests reported on uses as initial solution the schedule created by simple dispatching and later improved by Simulated Annealing.

4.4 Lower bounds

For each of the lower bound functions we specified the worst-case and the best-case time complexities in section 3, but these expressions does not fully reflect the practical behavior of the computations.

For the purpose of testing the behavior of the bounding functions a special set of parallel-machine problems were created. Such a problem consists

of n jobs and m machines, where each job has a release time, a processing time and a due time. The test problems created contains 5, 10, 20, 40, 80, 160 or 320 jobs and 1, 2, 4, 8, 16 or 32 machines. For each combination of jobs and machines ten random problems were created. As these problem does not contain any machine-specific information, the MaxFlow bound extended for the Jobshop Representation were not tested, as the result would otherwise be identical to the standard MaxFlow bound.

Figure 1 shows the average computation time to bound a single problem for a fixed number of jobs $n = 20$ and $n = 320$. These graphs show that all of the bounding functions are virtually independent of the number of machines, and the fluctuations diminishes with increasing number of jobs. Even the bound based on Jackson's Pseudo Preemptive Schedule (JPPS), for which the worst-case time complexity is linear in the number of machines, show a very machine-independent behavior and thus shows a practical behaviour equal to its best-case time complexity of $\Omega(n \log n)$.

Figure 2 shows the average computation time to bound a single problem for a fixed number of machines $m = 1$ and $m = 8$. These graphs very will show the expected n^2 dependency of SSB and the $n \log n$ dependency of both JPPS and JPS. The graphs also show that the time to compute ASSB is insignificant compared to the time needed to find the critical set, using SSB, on which ASSB is to be applied. Please note that the time for ASSB includes both the time for SSB and the time spent to compute ASSB itself. Another interesting observation is that computing the MaxFlow bound depends on the number of jobs as slightly less than n^2. This makes the computation of the MaxFlow bound comparable and even faster than SSB with problems involving more than 50 jobs. Comparing SSB and JPPS, which computes the same bound, the graphs show that JPPS is faster for almost all number of jobs, except very small problems, where the complexity of implementing JPPS shows.

4.5 Search order and branching strategies

To test the efficiency of the Branch-and-Bound search orders we have arbitrarily chosen to use a 10 second time limited search using the SSB lower bound. Note that we always create an upper bound, i.e. a completion of the partial schedule, on each node even though we do not use it in the search or the simple branching strategy does not require it. This is partly done because it is required by the critical path branching strategy and also results in better schedules being found faster.

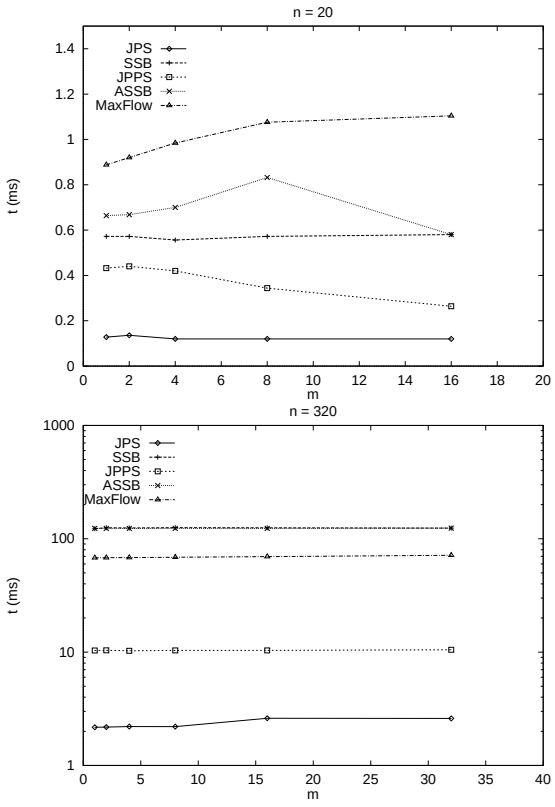

Figure 1: Average time to compute lower bounds as a function of the number of machines m at fixed number of jobs $n = 20, 320$.

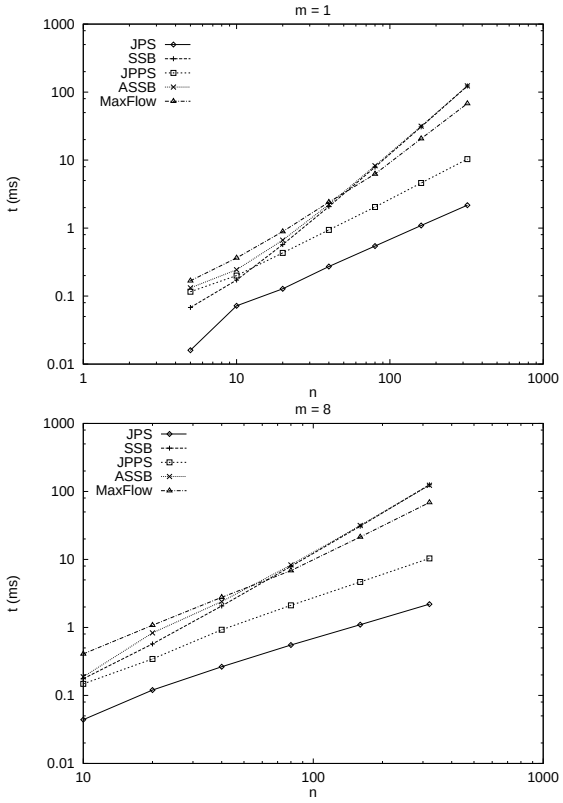

Figure 2: Average time to compute lower bounds as a function of the number of jobs n at fixed number of machines $m = 1, 8$.

	Intervals Rep.			Strings Rep.			Jobshop Rep.		
	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\#o$
Stack	475.1	2109	111	480.5	3858	114	476.1	662	152
Queue	475.2	1421	117	474.3	1683	140	478.2	795	140
PriLwr	472.9	1445	122	475.0	2321	125	476.3	658	154
PriUpr	471.0	1504	127	473.3	2227	136	475.0	586	157
PriAvg	470.7	1497	129	471.5	1925	141	475.7	605	156

Table 2: Performance of search strategies for single constraint branching in each of the three representations.

Table 2 and 3 shows the performance of the different pool types to store nodes during the search and thereby the different resulting search orders. Table 2 reports on the results obtained when branching by adding a single new constraint and table 3 reports on the results obtained when branching using critical paths.

The tables show that the classical depth-first search resulting when using a stack, as it is usually done when performing sequential Branch-and-Bound search, returns worse results than any of the prioritized search strategies. Although newly created nodes are placed on the stack such that the most promising is added last (and thus removed first), this is apparently not enough to guarantee good results.

The best results are obtained when the node selection is based on some priority that estimates the quality of the set of solutions represented by a single node. When choosing between using a lower bound and an upper bound, on the best possible solution represented by a node, to guide the search, the tables clearly indicate that using upper bounds — or at least a combination hereof — on the average always finds the best solutions independent of the representation and the branching strategy used.

A disadvantage of the prioritized search is the large amount of unbounded nodes with low priority that pile up. At the end of the ten-second Branch-and-Bound search, the number of nodes in the pool can easily be as large as 30,000-50,000 nodes which demands much of the computing memory. The advantage of the depth-first search is here that it only requires the storage of nodes along a path from the root in the search tree to the current node being bounded.

With respect to branching strategies we see a large and opposite difference for the Strings Representation and for the Jobshop Representation. The

	Intervals Rep.			Strings Rep.			Jobshop Rep.		
	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\#o$
Stack	476.7	2241	103	481.0	4354	111	470.2	540	177
Queue	475.0	1526	107	478.4	1533	122	470.1	387	177
PriLwr	473.7	1553	115	479.3	2703	125	468.9	365	182
PriUpr	473.4	1664	117	478.3	2510	126	467.3	423	184
PriAvg	473.7	1540	117	477.8	2610	128	467.2	392	186

Table 3: Performance of search strategies for longest path branching in each of the three representations.

reason why the Strings Representation fails with the critical path branching is that each time a branching is performed a lot of new nodes are generated. During the ten seconds of Branch-and-Bound search, more than 50,000 nodes can easily be generated. On the other hand, due to the large flexibility in the Jobshop Representation, where operations can be excluded from single machines, the number of generated nodes can be kept low.

To conclude on the results shown in table 2 and table 3 then the single constraint branching strategy should be used in connection with the Intervals and the Strings representations and the longest path branching strategy should be used with the Jobshop Representation. Furthermore, if the Branch-and-Bound search is time-limited such that memory will not be of any concern then the selection of nodes should be based on a priority where highest priority is given to the node having the highest average of both lower and upper bound.

All the following tests will, unless otherwise specified, use the above mention combinations of branching strategy and search order.

4.6 Immediate selection rules

In table 4 we examine the effect of using immediate selection rules during Branch-and-Bound search. As we specifically designed the Jobshop Representation to allow for a large set of immediate selection rules it is naturally that it is here we find the most. In the Strings representation there are only a few rules that can be tested. As the success of the immediate selection rules depends on the upper bound, we report the results for the weaker upper bound provided by the Shifting Bottleneck procedure and for the stronger upper bound provided by Simulated Annealing. Testing is performed as

Immediate selection rules		Shifting Bottleneck				Simulated Annealing			
		$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\overline{t_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\overline{t_o}$
		Strings Representation							
With		114	480.5	4059	5.6	144	472.7	2926	4.4
Without		102	481.7	4522	6.1	132	472.9	3457	4.9
Machine	Precedence	Jobshop Representation							
All	All	182	470.1	461	3.1	187	468.1	441	2.8
All (MaxFlow)	All	171	471.8	122	3.9	177	468.9	107	3.4
All	None	157	474.5	699	4.0	167	470.3	632	3.5
None	All	173	471.3	605	3.4	179	468.4	591	3.2
None	None	153	474.7	871	4.1	162	470.3	806	3.7

Table 4: Performance of immediate selection rules in the second and third representation when starting with an initial schedule created with the Shifting Bottleneck procedure or by Simulated Annealing.

branch-and-bound search limited to 10 seconds using simple branching in the Strings representation and critical path branching in the Jobshop representation.

The results show a clear improvement in both quality and the number of problems solvable within ten seconds when applying all immediate selection rules. As expected, we get a larger increase when applying the precedence related immediate selection rules of Carlier and Pinson [9] since the Jobshop Representation was specifically designed to allow for these tests.

These tests also show that the solution quality depends on the initial upper bound provided, but does not seem to indicate that the immediate selection rules are more effective with a reduced upper bound. The reduction in time and nodes and the increase in the problems solved to optimality are the same in both cases.

In table 4 we have also included results on a special machine-selection rule based on the MaxFlow lower bound. The principle in this rule is to modify the underlying graph on which the MaxFlow bound is computed, to test if an operation can be processed exclusively on a single machine. Although this method is stronger quality-wise, the results show it to be too time-consuming to be competitive — the performance decreases just by including it.

	Intervals Rep.	Strings Rep.	Jobshop Rep.
Problems with optimum known	212	212	212
Solutions proved optimal within 60 sec.	139	161	204
Completed proofs requiring branching	3	25	42

Table 5: Number of test problems for which the solution is proved to be optimal for all lower bounds except JB for each of the three representations.

4.7 Branch-and-Bound

To test how the lower bound functions perform in a Branch-and-Bound search, we have selected the 212 of the 252 test problems for which the optimal solution is known. On each of these problems we perform a 60 second time-limited Branch-and-Bound search starting with an upper bound equal to the optimal solution. We thus ensure that the numbers are comparable with respect to lower bound functions in that the number of nodes bounded is independent of the search order and thus only depends on the quality of the lower bound — and immediate selection rules which are the same in all tests. To further ensure that variations are only due to lower bounds we only report on the problems that require branching and are solved completely within the 60 seconds time limit, for each lower bound function (except JB). In table 5 we show how many of the 252 test problems meet all of these criteria.

Table 6 shows the results obtained with the various lower bound functions in each of the three representations, for the problems that require branching and are solved completely within 60 seconds. A problem with these numbers is the weakness of the Intervals Representation since only three test problems qualify. Although SSB and JPPS returns lower bounds of equal quality the time to compute them can differ a lot (as figures 2 and 1 also indicates). Since the Strings Representation operates with complete strings instead of operations, the size of the problem to bound here is relatively smaller than in the Jobshop Representation. Since JPPS is most efficient at larger problems it is expected that SSB compares better against JPPS in the Strings Representation than in the Jobshop Representation.

Applying ASSB to improve on the SSB lower bound has a slight, but not very significant, influence on the number of nodes. But, since ASSB is very fast to compute it does not influence the running times very much.

Finally, table 6 show that the strongest bound in the Jobshop Repre-

	Intervals Rep.		Strings Rep.		Jobshop Rep.	
	$\overline{n_o}$	$\overline{t_o}$	$\overline{n_o}$	$\overline{t_o}$	$\overline{n_o}$	$\overline{t_o}$
JPS	989	0.50	2377	2.44	1437	5.65
SSB	989	0.73	2192	2.81	547	4.25
JPPS	989	0.89	2192	3.15	547	4.21
ASSB	965	0.98	1933	2.93	545	4.36
MaxFlow	989	1.08	2192	4.15	539	6.27

Table 6: Performance of the lower bounds in each of the three representations when proving optimality. Results are reported only on the problems for which optimality could be proven within 60 seconds and which required branching in the proof, for each lower bound except JB.

sentation is as expected the MaxFlow bound since this is the only bound that takes machine-information into consideration. Unfortunately, the large computation time needed to compute the bound makes it unattractive as a bounding function.

Table 7 is a comparison of the three representations when we use the best of initial schedules, search orders, branching strategies and immediate selection rules. Each of the tests is initiated with an upper bound provided by Simulated Annealing, the search order and branching strategy are as stated in section 4.5 and all immediate selection rules are applied. The Branch-and-Bound search is time-limited to 60 seconds and only the time for this search is reported in table 7. For comparison against published results, the Intervals representation is similar to a Branch-and-Bound solution method for the parallel-machine problem by J. Carlier [7] and the Strings Representation is a bit stronger than a Branch-and-Bound method for the multi-processor flow shop problem by S.A. Brah and J.L. Hunsucker [4].

This table clearly shows the Jobshop Representation to be superior against the other two representations both with respect to solution quality and proof of optimality. Which of the lower bound functions SSB, JPPS or ASSB to use does not seem to influence the results much, as both the quality of the solutions, the number of optimal solutions found and the computation times are about equal for all three bounding functions. There does seem to be a slight preference for the ASSB bound, but as table 6 indicates, this might as well be due to a different selection order of the nodes.

Finally, in figure 3 we illustrate the performance of the three representations as a graph where problems are sorted in increasing order of solution

	Intervals Representation				Strings Representation				Jobshop Representation			
	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\overline{t_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\overline{t_o}$	$\#o$	$\overline{\lvert S_o \rvert}$	$\overline{n_o}$	$\overline{t_o}$
JB	17	467.2	40424	56.1	44	473.8	74365	50.1	194	465.9	1872	15.1
JPS	126	467.6	14333	30.1	146	470.3	14411	25.6	202	464.8	1766	13.4
SSB	137	467.8	6458	27.5	157	468.5	7437	23.0	208	465.0	992	12.2
JPPS	137	467.7	6670	27.5	157	468.5	7642	23.2	208	465.0	1017	12.1
ASSB	140	467.8	5006	27.0	159	468.6	6496	22.6	209	465.0	946	12.0
MaxFlow	136	467.8	5003	27.7	153	470.8	6450	24.5	201	465.5	775	14.5

Table 7: Performance of the lower bounds in each of the three representations on a test suite of 252 problems. For comparison $\overline{|S_l|} = 476.8$.

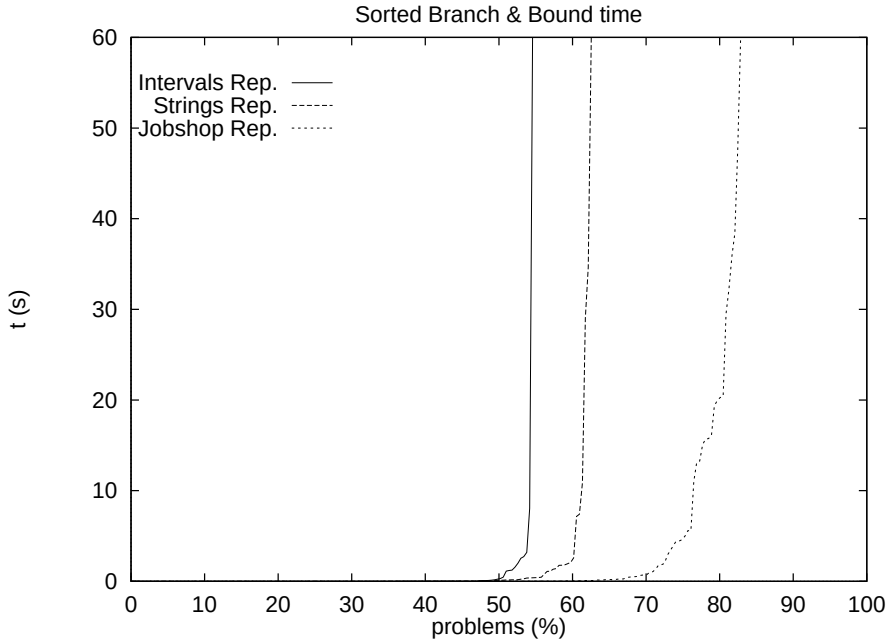

Figure 3: Test problems sorted in increasing order of Branch-and-Bound solution time for each of the three representations. Lower bound used is JPPS.

time. We have arbitrarily chosen to depict results obtained using the JPPS bounding function. First of all, this figure shows that about half of the 252 test problems are sufficiently easy for the Simulated Annealing procedure to find an optimal solution and subsequently requiring only a single application of a lower bound to prove the optimality. Next, the figure shows how well the Jobshop Representation compares against the other two representations, by being able to solve more problems in less time. The curve for the Jobshop representation also looks more appealing since it is less steep than the remaining curves. This indicates that there is a much better chance of solving a problem using the Jobshop Representation by applying Branch-and-Bound than there is for any of the other two representations.

4.8 Solving hard problems

The results presented in the previous section indicates that the best of the tested methods at solving larger and difficult multi-processor problems is the Jobshop Representation combined with the JPPS lower bound. Of the 252 test problems only 212 were solved to optimality during the previously presented tests. Table 8 shows the results when we try to solve the remaining 40 hard problems. We proceed by first applying Simulated Annealing to provide a good initial solution. This is followed by a 60 seconds time-limited Branch-and-Bound search where highest priority is given to the node having the least average of lower and upper bounds. The solution from this search is then used as an initial upper bound in the final, one hour limited Branch-and-Bound search, using simple depth-first search due to memory constraints.

The test problems are named according to how they were generated. In a name sc_m_jp, s is either 'f' for flow-shop or 'j' for job-shop, c is the number of centers ('0' for 10 centers), m is the average number of machines, j is the number of jobs and p denotes the percentage of high processing times ('a', 'b' or 'c' for resp. 0, 5 or 10 percent).

5 Conclusion

We have presented a Branch-and-Bound method for the multi-processor Job Shop and Flow Shop problems which on a set of random test problems is superior to existing published methods. Furthermore we have presented results from an extensive testing of the different components that constitutes a complete Branch-and-Bound solution methods. These results indicate that

	Sim. An.		Initial B & B			Branch & Bound				
Problem	$\|S\|$	t	$\|S\|$	n	t	$\|S\|$	n	t	o	t_{total}
f0_1_10a	644	3.3	589	5150	60.0	586	469269	3600.0		3663.3
f0_1_20c	821	3.4	722	6227	60.0	717	157850	1147.8	yes	1211.2
f0_1_15a	820	6.9	727	1847	60.0	727	281407	3600.0		3666.9
f0_1_15b	896	6.1	830	1799	60.0	830	244999	3600.0		3666.1
f0_1_15c	1280	6.6	1140	2551	60.0	1106	256382	3600.0		3666.6
f0_1_20a	1068	10.3	931	1273	60.0	931	230160	3600.0		3670.3
f0_1_20b	1139	10.8	1024	1185	60.0	1024	161108	3600.0		3670.8
f0_1_20c	1558	9.4	1330	978	60.0	1330	186968	3600.0		3669.4
f0_2_10a	508	3.8	471	9393	60.0	471	16677	96.9	yes	160.7
f0_2_10c	585	3.2	585	9051	60.0	569	610608	3600.0		3663.2
f0_2_15a	625	3.9	594	6728	60.0	594	363794	3600.0		3663.9
f0_2_15b	889	6.1	829	3017	60.0	787	328263	3600.0		3666.1
f0_2_20c	1461	8.6	1140	2923	60.0	1140	278397	3600.0		3668.6
f0_3_20b	628	7.6	597	2258	60.0	597	261431	3600.0		3667.6
f0_3_20c	680	5.8	664	5196	60.0	664	328176	3600.0		3665.8
f0_4_15a	386	4.6	381	4561	60.0	381	512306	3600.0		3664.6
f0_4_15b	567	4.6	565	5764	60.0	518	91114	689.6	yes	754.2
f0_4_20a	410	6.7	405	2602	60.0	405	327223	3600.0		3666.7
f5_1_20a	771	4.7	677	3603	60.0	677	473598	3600.0		3664.7
f5_2_15a	401	2.7	381	11863	60.0	381	105204	532.3	yes	595.0
f5_2_15b	826	3.0	752	5283	60.0	752	475674	3600.0		3663.0
f5_3_20a	418	3.2	391	10287	60.0	391	563722	3600.0		3663.2
f5_4_15a	269	2.0	258	13096	60.0	258	1097396	3600.0		3662.0
f5_4_15c	305	1.6	293	17128	60.0	292	45068	144.0	yes	205.6
f5_4_20b	274	2.6	269	6519	60.0	269	638764	3600.0		3662.6
j0_1_15a	670	4.9	636	1803	60.0	629	211035	3600.0		3664.9
j0_1_15c	719	4.4	719	3215	60.0	708	240768	3600.0		3664.4
j0_1_20a	731	6.4	731	821	60.0	728	159131	3600.0		3666.4
j0_1_20b	782	7.5	777	1281	60.0	777	148453	3600.0		3667.5
j0_1_20c	870	6.9	866	1985	60.0	866	149621	3600.0		3666.9
j0_2_20b	771	5.9	762	3193	60.0	750	235	6.4	yes	72.3
j0_2_20c	681	5.1	681	2828	60.0	679	238225	3600.0		3665.1
j2_2_15b	198	0.9	190	26188	60.0	190	1646786	3600.0		3660.9
j2_3_15b	186	0.8	184	30399	60.0	184	1921542	3600.0		3660.8
j2_3_15c	157	0.7	152	22832	60.0	151	1646269	3600.0		3660.7
j2_4_20b	265	1.6	258	22272	60.0	258	1656852	3600.0		3661.6
j2_4_20c	216	1.1	209	17641	60.0	209	1492699	3600.0		3661.1
j5_1_15a	476	2.4	460	6268	60.0	457	44607	340.8	yes	403.2
j5_4_20a	224	2.3	219	10416	60.0	219	681232	3600.0		3662.3
j5_4_20c	268	2.5	257	9404	60.0	257	682513	3600.0		3662.5

Table 8: Solving the hard problems that could not be solved within one minute of BranchandBound search. The results are for the Jobshop Representation using JPPS as a lower bound.

it is mainly the presence of *immediate selection rules* that lies behind the success of our presented Branch-and-Bound method. Furthermore, our results also indicates the importance of a well-chosen search strategy when selecting the next Branch-and-Bound node for processing. Finally, we have presented results on a new implementation of a lower bound based on Jackson's Pseudo Preemptive Schedule (JPPS) which is faster on larger problems (15 or more jobs) than the much more widely used Subset Bound (SSB).

References

[1] J. Adams, E. Balas, D. Zawack, *The shifting bottleneck procedure for job shop scheduling*, Management Science 34 (1983) 391-401.

[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network flows: Theory, algorithms, and applications*, Prentice Hall, Englewood Cliffs, New Jersey 07632 (1993) chapter 6-8.

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

[4] S.A. Brah, J.L. Hunsucker, *Branch and bound algorithm for the flow shop with multiple processors*, European Journal of Operational Research 51 (1991) 88-99.

[5] P. Brucker, B. Jurisch, B. Sievers, *A Branch-and-Bound Algorithm for the Job-Shop Scheduling Problem*, Discrete Applied Mathematics 49 (1994), 107-127.

[6] J. Carlier, *The One-Machine Sequencing Problem*, European Journal of Operational Research 11 (1982), 42-47.

[7] J. Carlier, *Scheduling jobs with release dates and tails on identical machines to minimize the makespan*, European Journal of Operational Research 29 (1987) 298-306.

[8] J. Carlier, E. Pinson, *An Algorithm for Solving the Job- Shop Problem*, Management Science 35 (1989), 164-176.

[9] J. Carlier, E. Pinson, *A Practical Use of Jackson's Preemptive Schedule for Solving the Job-Shop Problem*, Annals of Operations Research 26 (1990), 269-287.

[10] J. Carlier, E. Pinson, *Adjustment of Heads and Tails for the Job-Shop Scheduling Problem*, European Journal of Operational Research 78 (1994) 146-161.

[11] M.R. Garey, D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, Ca (1979).

[12] J. Grabowski, E. Nowicki, S. Zdrzalka, *A block approach for single-machine scheduling with release dates and due dates*, European Journal of Operational Research 26 (1986).

[13] J.A. Hoogeven, J.K. Lenstra, B. Veltman, *Minimizing the makespan in a multiprocessor flowshop is strongly NP-hard*, European Journal of Operational Research, to appear.

[14] W.A. Horn, *Some simple scheduling algorithms*, Naval Research Logistics Quarterly 21 (1974) 177-185.

[15] P.J.M. van Laarhoven, E.H.L. Aarts, J.K. Lenstra, *Job shop scheduling by simulated annealing*, Operations Research 40 (1992) 113-125.

[16] M. Perregaard, *Branch-and-Bound methods for the Multi-Processor Job Shop and Flow Shop scheduling problems*, Master's Thesis, Department of Computer Science, University of Copenhagen.

[17] A. Vandevelde, *Minimizing the makespan in a multiprocessor flow shop*, Master's Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology (1994).

[18] A. Vandevelde, H. Hoogeveen, C. Hurkens, J.K. Lenstra, *Lower bounds for the multiprocessor flow shop*, Draft, Department of Mathematics and Computing Science, Eindhoven University of Technology (1996).