A survey of optimization algorithms for job shop scheduling

Eindhoven University
of Technology

Department of Mathematics
and Computing Sciences

Memorandum COSOR 97-19

A survey of optimization algorithms for job shop scheduling

by

J. Aerts

Eindhoven, October 1997
The Netherlands

A survey of optimization algorithms for job shop scheduling

Joep Aerts

Abstract

In this survey optimization methods for the job shop problem are discussed. Most of the algorithms developed so far are branch and bound algorithms. Different ways of branching and computing lower bounds are main issues in this overview. The computational efforts done in each node are also discussed. In branch and bound algorithms finding a good balance between the computational efforts in each node and the number of nodes investigated is crucial for the efficiency of the algorithm. Results of several algorithms are compared.

1 Introduction

In the job shop problem we have to schedule a set of jobs on a set of machines subject to the following constraints. Each job consists of a number of operations, which have to be processed in a given order, each on a specified machine. Each machine can only handle one operation at a time. For all operations a processing time is given. It is not allowed to interrupt the processing of an operation. Subject to these constraints, we want to find a schedule for which the completion time of the last operation is minimal. To state an instance of the decision variant of the problem, we also introduce an integer T. In the job shop feasibility problem, we are trying to determine if there exists a schedule which is finished at time T.

Although easily stated, the job shop problem turns out to be one of the hardest problems in the area of combinatorial optimization. This is illustrated by the fact that a classical benchmark problem of 10 jobs and 10 machines remained unsolved for more than twenty years. It was posed in 1963 by Fisher and Thompson [12] and solved by Carlier and Pinson in 1986 [8]. The job shop problem was proven $\mathcal{NP}$-hard in the strong sense by Garey, Johnson, and Sethi [14]. Some very special cases of the problem can be solved in polynomial time, but their immediate generalizations are $\mathcal{NP}$-hard. These results are summarized in Table 1 [21], where m is the number of machines, n the number of jobs, $l(j)$ the number of operations of job j and $p(i)$ the processing time of operation i.

The paper is structured as follows. In Section 2 several models for the job shop problem are introduced. Section 3 describes the lower bounds, improvements on the lower bounds, methods to reduce the number of nodes in the search tree, and several branching schemes. Based on the information of Section 3, the branch and bound algorithms proposed in literature are discussed in Section 4. Section 4 also compares the results of the algorithms and Section 5 gives our conclusions and remarks.

solvable in polynomial time	$\mathcal{NP}$-hard (*in the strong sense)
(1a) $m = 2$, all $l(j) \leq 2$	(1b) $m = 2$, all $l(j) \leq 3$
	$m = 3$, all $l(j) \leq 2$
(2a) $m = 2$, all $p(i) = 1$	(2b)*$m = 2$, all $p(i) \leq 2$
	*$m = 3$, all $p(i) = 1$
(3a) $n = 2$	(3b) $n = 3$
(4a) length≤ 3	(4b)*length≤ 4

Table 1: The complexity of job shop scheduling

2 Definitions and Notation

In this section we introduce several formal descriptions for the job shop problem. We present a disjunctive programming model and a disjunctive graph representation, which have been the basis for a large group of algorithms. Also the mixed-integer programming formulation used by Applegate and Cook [2] is introduced. Other possibilities we give for modeling the job shop problem are a packing formulation [17, 18] and the use of time windows [5, 17, 18].

An instance of the job shop problem consists of a set $\mathcal{O}$ of operations, a set $\mathcal{M}$ of m machines and a set $\mathcal{J}$ of n jobs. For the feasibility problem we also need to introduce a time T. Each operation $j \in \mathcal{O}$ has a processing time $p(j) \in \mathbb{N}^+$, a machine $M(j) \in \mathcal{M}$ on which the operation has to be processed, and a job $J(j) \in \mathcal{J}$ to which it belongs. On the set of operations a set A of precedence relations is defined in the following way: $(i,j) \in A$ if and only if $J(i) = J(j)$ and operation i is the immediate predecessor of j. In a job each operation has exactly one predecessor and one successor, except for the first and the last operation of the job.

We can now define a feasible schedule by a function $s : \mathcal{O} \to \mathbb{N}$ which assigns a starting time $s(j)$ to each operation $j \in \mathcal{O}$ satisfying the following constraints:

$$\forall_{j \in \mathcal{O}} \quad s(j) \geq 0; \tag{1}$$

$$\forall_{i,j \in \mathcal{O},\, (i,j) \in A} \quad s(i) + p(i) \leq s(j); \tag{2}$$

$$\forall_{i,j \in \mathcal{O},\, i \neq j,\, M(i) = M(j)} \quad s(i) + p(i) \leq s(j) \vee s(j) + p(j) \leq s(i). \tag{3}$$

The constraints (3) make sure that a machine cannot process two operations at the same time. Due to the form of these constraints we call this problem a *disjunctive programming problem.* Furthermore we define S as the set of feasible solutions. Among the feasible solutions we want to find the schedule with minimal makespan. Using the starting times we can define the objective function:

$$\min_{s \in S} \left(\max_{j \in \mathcal{O}} s(j) + p(j) \right). \tag{4}$$

A related way to model the job shop problem is by means of a *disjunctive graph.* This model is due to Roy and Sussmann. We define a vertex for each operation. The set A of precedence relations now corresponds to a set of arcs, which make sure that the operations of each job are processed in the correct order. Between all the operations which have to be processed on the same machine, a clique of disjunctive edges is formed. A disjunctive edge between i and j indicates that the two operations cannot be processed at the same time, but that no precedence relation between these operations has been set so far. Finally we give all vertices a weight equal to their processing times. An example is given in Figure 1 where operation 2,3 represents the operation of job 2 to be processed on machine 3. The italic numbers at the bottom of the vertices represent the weights.

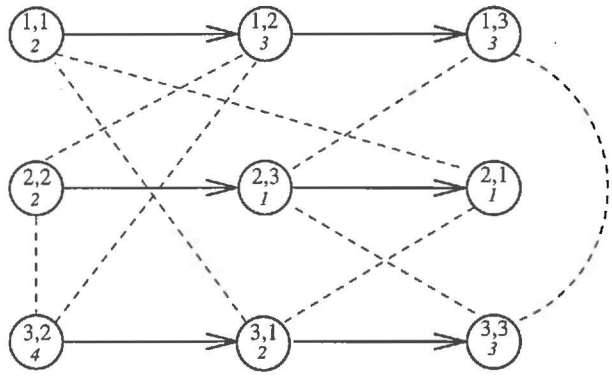

Figure 1: A disjunctive graph representation of a problem with 3 jobs and 3 machines. The dotted lines represent the disjunctive edges.

A feasible schedule corresponds to an ordering of the operations on each machine in such a way that no cycles occur. In the graph this is equivalent to an acyclic orientation of the disjunctive edges. A directed edge (arc) corresponds to a precedence relation between the two operations. So we distinguish two kinds of precedence relations in the graph, the elements of A and the directed disjunctive edges. A solution is feasible if all the disjunctive edges are directed and the resulting directed graph contains no cycle. The length of a longest path in the graph is the value of the schedule. The work of Carlier and Pinson [8, 9, 10], a part of the work of Applegate and Cook [2], and the algorithms of Brucker et al. [5, 6] are based on this disjunctive graph model. These algorithms use one or more disjunctive edges to branch on by directing them one way or the other. Each node in the branching tree of these branch and bound methods can be represented by a graph. In this graph the elements of A and the directed edges are the arcs. The edges which are not directed are still represented by disjunctive edges.

The disjunctive programming formulation given above can be turned into a *mixed integer problem.* We need one binary variable Y_{ij} for every pair of operations $\{i, j\}$ with $M(i) = M(j)$ and define Y_{ij} to be 1 if operation i is scheduled before operation j and 0 otherwise. Instead of the disjunctive constraints (3) we can now use the following constraints to make sure that there is only one job at a time on each machine:

$$\forall_{i,j\in\mathcal{O},\, i\neq j,\, M(i)=M(j)} \quad \begin{aligned} s(i) &\geq s(j) + p(j) - K \cdot Y_{ij} \\ s(j) &\geq s(i) + p(i) - K \cdot (1 - Y_{ij}), \end{aligned} \tag{5}$$

where K is a large integer. Applegate and Cook [2] use both the disjunctive and the mixed integer formulation for their cutting plane procedures.

Martin and Shmoys [17, 18] introduce a lower bound based on a *packing model.* The packing formulation is a model for the job shop feasibility problem. To state it we define a job schedule to be an assignment of starting times to the operations of one job such that no operation starts before the ending of the immediate predecessor and such that the total job is finished by time T. With the definition of job schedules we can state the feasibility problem in the following way: does there exist a set of job schedules, one for each job, such that no two schedules require the same machine at the same time?

Let $\mathcal{F}_j$ be the set of all job schedules for a job $j \in \mathcal{J}$ and let $\gamma(\sigma, i, t) \in \{0, 1\}$ indicate whether job schedule σ requires machine i at time t. Let $x_{j\sigma}$ be a 0-1 decision variable that indicates if job j is scheduled according to job schedule σ and let λ be the maximum number of jobs concurrently on any machine. Then the

objective is to minimize λ subject to:

$$\forall_{j\in\mathcal{J}} \quad \sum_{\sigma\in\mathcal{F}_j} x_{j\sigma} = 1; \tag{6}$$

$$\forall_{i\in\mathcal{M},\, t=1,\ldots,T} \quad \sum_{j\in\mathcal{J}} \sum_{\sigma\in\mathcal{F}_j} \gamma(\sigma,i,t)x_{j\sigma} \leq \lambda; \tag{7}$$

$$\forall_{j\in\mathcal{J},\, \sigma\in\mathcal{F}_j} \quad x_{j\sigma} \in \{0,1\}. \tag{8}$$

We have a positive answer to the job shop feasibility problem with value T if and only if the optimal value for λ is 1.

The last model for the job shop problem we discuss is based on a time oriented approach, and uses time windows. Brucker, Jurisch, and Krämer [5] use the time windows to derive conditions for directing disjunctive edges and Martin and Shmoys [17, 18] develop complete algorithms based on these ideas. A time interval is constructed for each operation in which the operation has to be processed or has to start, depending on the definitions of the bounds. These bounds are determined by using the information of an upper bound, predecessors, and successors. Two possible descriptions of the time windows are given in Section 3.2.

3 Lower bounds and branching schemes

In this section we describe the lower bounds, the computational efforts in each node of the search tree, and the branching rules of the algorithms for solving the job shop problem to optimality. We give different approaches to calculate lower bounds in Section 3.1. Section 3.2 discusses iterative improvements of lower bounds and the computational efforts in the nodes. Section 3.3 compares the results of the lower bounds and Section 3.4 deals with the branching rules used by the algorithms. The branch and bound algorithms also use upper bounds. We will not discuss heuristics to get the upper bounds but refer to Vaessens, Aarts, and Lenstra [21]. In Section 4, the algorithms proposed in literature will be explained using the contents of Section 3.

3.1 Lower Bounds

3.1.1 Direct lower bounds

The most straightforward lower bounds are the machine lower bound and the job lower bound. The first one is derived by computing the sum of all the operations which have to be processed on one machine and then taking the maximum of these sums over all machines:

$$\max_{m\in\mathcal{M}} \left(\sum_{i\in\mathcal{O};M(i)=m} p(i) \right). \tag{9}$$

The second one is just the maximum total processing time of the jobs:

$$\max_{j\in\mathcal{J}} \left(\sum_{i\in\mathcal{O};J(i)=j} p(i) \right). \tag{10}$$

As expected these lower bounds perform badly in most situations.

3.1.2 One-machine relaxation

The lower bound Carlier and Pinson [8] used in their algorithm, which solved the legendary Fisher and Thompson 10×10 problem, is based on Jackson's preemptive schedule for the one-machine head-body-tail problem. Relaxing the capacity constraint of all but one machine gives a problem equivalent to the one-machine problem with release dates (heads) and delivery times (tails). In the graph representation we can see this relaxation as dropping all the disjunctive edges between the operations of the other machines. This means that we take only the arcs of A, the directed edges, and the disjunctive edges of the machine we are considering into account.

For a machine m define $O_m = \{i \in \mathcal{O} \mid M(i) = m\}$. Furthermore we define the following quantities for each $j \in O_m$:

- $r(j)$: the head of operation j, defined as the time needed for the processing of the operations preceding j, taking the arcs of A and the directed edges into account;
- $q(j)$: the tail of operation j, defined as the time needed for the processing of the operations succeeding j, taking the arcs of A and the directed edges into account.

Furthermore we define for $S \subseteq O_m$, $R(S) = \min_{j \in S} r(j)$, $Q(S) = \min_{j \in S} q(j)$, and $P(S) = \sum_{j \in S} p(j)$. We will discuss different ways to compute heads and tails in Section 3.2.

Unfortunately the one-machine head-body-tail problem is also $\mathcal{NP}$-hard. An extra relaxation is made by dropping the non-preemption constraint. Now we can use Jackson's algorithm to construct a schedule. The preemptive earliest due date (EDD) schedule is constructed in the following way:

- If there is an operation available schedule one with longest tail.
- Take release dates and completion times as decision points.

The value of the minimal preemptive schedule equals

$$\max_{S \subseteq O_m} R(S) + P(S) + Q(S). \tag{11}$$

Consider this relaxation for each machine. The maximum of the makespans of the one-machine problems is a lower bound. We will call this bound the preemptive EDD-bound (prEDD). In literature it is sometimes refered to as Jackson's preemptive schedule lower bound. We know that the right hand side of (12) is a lower bound for the job shop problem and we can prove that the prEDD equals this lower bound. So:

$$\text{prEDD} = \max_{m \in \mathcal{M}} \left(\max_{S \subseteq O_m} R(S) + P(S) + Q(S) \right). \tag{12}$$

Carlier [7] proves that this lower bound can be computed in $O(mn \log n)$ time. Instead of computing the preemptive schedule it is also possible to determine the non-preemptive schedule. For small instances we can solve this $\mathcal{NP}$-hard problem in an acceptable amount of time. Unfortunately the results of this exact one-machine relaxation are not much better than the prEDD and more computation time is needed because an instance of an $\mathcal{NP}$-hard problem has to be solved for each machine.

Although the prEDD is not very tight, it is widely used in the algorithms. The most acceptable reason for this is the lack of a stronger lower bound, computable in the same amount of time. There have been numerous efforts in trying to find

these 'better' lower bounds using other points of view. In the following sections we will discuss the use of geometric methods by Brucker and Jurisch [4], the cutting plane approach of Applegate and Cook [2], and the packing approach of Martin and Shmoys [17, 18]. In this article we only mention the surrogate duality relaxation investigated by Fisher, Lageweg, Lenstra and Rinnooy Kan [13].

3.1.3 Two-job relaxation

The two-job bound of Brucker and Jurisch [4] is a method for calculating lower bounds based on two-job relaxations. First we state the problem they solve to get a lower bound. Consider operations $i_1, \ldots, i_p$ and $j_1, \ldots, j_q$ with $J(i_1) = \ldots = J(i_p)$ and $J(j_1) = \ldots = J(j_q)$. Between operations of different jobs precedence relations may exist when two operations have to be processed on the same machine. These precedence relations correspond to directed edges. A release date and a delivery time can be computed for each operation in the same way as for the prEDD. Now we want to find a schedule for these two jobs, satisfying the precedence constraints and release dates which minimizes

$$\max_{k=i_1,\ldots,i_{n_i},j_1,\ldots,j_{n_j}} \{s(k) + p(k) + q(k)\}. \tag{13}$$

Brucker and Jurisch determine their lower bound by solving this problem for every pair of jobs. Their algorithm for every two-job relaxation is based on the graphical method for solving job shop problems with two jobs. The method consists of finding a shortest path in a two-dimensional space with obstacles. Each job corresponds to an axis and an obstacle of size $p(i_k) \times p(j_l)$ is constructed if $M(i_k) = M(j_l)$. The left-bottom corner of this obstacle is placed on $(r(i_k), r(j_l))$.

Brucker and Jurisch extend this idea to a three-dimensional space. They are able to take the precedence constraints and the heads and tails into account. For an exact description of the model and their algorithm we refer to their article. In some cases this bound can be useful. If the ratio of the number of jobs to the number of machines is small, this lower bound procedure can produce a better result than the prEDD. However, in most instances the bound produces worse values than the prEDD. Some results are given in Section 3.3 and for additional results we refer again to [4].

3.1.4 Cutting plane lower bound

In a cutting plane approach a relaxation of the original problem is solved to get the first solution. Then adding inequalities which are valid for all feasible solutions but not for the optimal solution of the relaxation makes the formulation stronger. Applegate and Cook use both the disjunctive and the mixed integer formulation as a basis for a cutting plane approach. In the first one they drop the disjunctive constraints (3) and in the mixed integer formulation they use the LP-relaxation on the Y_{ij} variables. We give here some examples of the cuts they use. For a complete enumeration we refer to their article [2].

- Basic cuts
  The basic cuts are based on a one-machine relaxation. Take a machine m and a set $S \subseteq O_m$. Then it is straightforward to check that the following equation is valid for all feasible schedules:

  $$\sum_{j \in S} p(j)s(j) \geq R(S)P(S) + \sum_{i,j \in S; i>j} p(i)p(j), \tag{14}$$

  where $i > j$ means that every pair of operations is only used once. We can add this inequality for each subset of O_m for each machine m. It is also possible to

add the reverse inequalities, by introducing completion times and considering the schedules backwards.

- Two-job cuts
  Let $i, j \in \mathcal{O}$ with $M(i) = M(j)$. In case $r(j) < r(i)+p(i)$ and $r(i) < r(j)+p(j)$ the basic cut ($S = \{i, j\}$) can be sharpened:

$$(p(i) + r(i) - r(j)) \cdot s(i) + (p(j) + r(j) - r(i)) \cdot s(j) \geq p(i)p(j) + r(i)p(j) + r(j)p(i). \tag{15}$$

  We can show that this inequality is valid by substituting the earliest possible starting times for i and j in (15) because the coefficients of the starting times are positive by assumption. The earliest possible starting times, when we assume without loss of generality that i precedes j, are $r(i)$ for i and $r(i)+p(i)$ for j. For these values the inequality is valid, so the inequality is valid for all feasible starting times. The reverse inequality can also be used. Applegate and Cook give an extension of this rule for all subsets $S \subseteq O_m$ (clique cuts) and an extension to more machines (two-job, two-machine cuts).

The above cuts do not use the mixed-integer formulation. Nevertheless they can be used to strengthen the feasible region of the relaxation of the mixed-integer formulation. The following cuts make use of the Y_{ij}-variables of the mixed integer formulation, so they can only be used in the mixed-integer approach.

- Triangle cuts
  For any three operations i, j, k that have to be processed on the same machine, we can add the triangle inequality:

$$Y_{ij} + Y_{jk} + Y_{ki} \leq 2. \tag{16}$$

  Since only one of the two possible variables Y_{ij} and Y_{ji} is defined, it may be necessary to substitute $1 - Y_{ij}$ for Y_{ji}. This inequality is based on the transitivity of the precedence relations: if i is scheduled before j and j is scheduled before k, i has to be scheduled before k to get a feasible schedule.

- Half cuts
  Let $S \subseteq O_m$. For each operation $j \in S$ the inequality

$$s(j) \geq R(S) + \sum_{i \in S \setminus \{j\}} Y_{ij} p(i) \tag{17}$$

  is valid for all schedules, since operation j cannot start processing before all the preceding jobs on machine $M(j)$ have been completed.

The lower bounds of the cutting plane approaches are better than the values of the prEDD [2]. However, the computation times are disappointing, only the results of the relaxation of the disjunctive model with the basic cuts added seems promising. The results are for some instances substantially better and the time gap is not that big. Some computational results are given in Section 3.3. Applegate and Cook end their discussion of the cutting plane approach with the remark that it remains a research challenge to find classes of valid inequalities that will close the large gap between the lower bound values and the optimal values of the scheduling problems within a reasonable amount of time.

3.1.5 Fractional packing lower bound

Recently Martin and Shmoys [17, 18] proposed a lower bound based on a packing formulation for the job shop feasibility problem: minimize λ subject to (6), (7), and (8). The packing lower bound is based on the following statement: 'If we can prove that the optimal value for λ is greater than 1 for a fixed T we can conclude that $T+1$ is a lower bound for the problem'. If we consider the LP-relaxation by substituting $x_{j\sigma} \geq 0$ for (8) we can still use this result and determine the fractional packing lower bound. We can show that the optimal value of the LP-relaxation is greater than 1 by finding a feasible dual solution of value greater than 1. Martin and Shmoys use the packing algorithm of Plotkin, Shmoys and Tardos [20] to find such a solution. Fortunately this algorithm is insensitive to the number of variables so the exponential number of decision variables is no problem.

To find the best lower bound, a bisection over T is needed, because the packing model is a model for the job shop feasibility problem. The algorithm of Martin finds the highest value of T for which a dual solution higher than 1 is found. The results of the bound are very good, but one subroutine of the algorithm takes exponential time. This subroutine is called several times for each T so the bound is not yet of practical use because it is too computationally demanding. In Section 3.3 some computational results are presented.

3.2 Improvement procedures for lower bounds

We can conclude that, in spite of all the efforts, the available lower bounds are not very tight or not of practical use due to their computation times. This section deals with procedures to improve lower bounds. These procedures can cause a reduction of the size of the branching tree as well. We will discuss the adjustment of heads and tails and methods to direct disjunctive edges in a one-machine relaxation (local). Some related methods for the entire problem are explained (global methods) and also shaving procedures for the time-window approach are mentioned.

3.2.1 Updating heads and tails

Examining the definition of heads and tails given when we introduced the prEDD, one way to see the head of an operation is as being the longest path length to the corresponding vertex in the disjunctive graph with the elements of A and the directed edges as arcs. The tail corresponds to the longest path length from the vertex. So the heads and tails can be computed by applying a longest path algorithm to a directed graph.

Another approach to compute the heads and tails is described by Brucker, Jurisch, and Sievers [6]. We first note that an operation cannot start before the ending of the preceding operation in the same job. So if $(i,j) \in A$, $r(j) \geq r(i) + p(i)$. For the second observation we consider an operation j and a set C of predecessors of j with a disjunction between each two operations of C. So operation j cannot start before all operations of C are finished. Combining these observations we get:

$$r(j) \geq \max\left\{r(i)+p(i),\ \max_{C' \subseteq C}\left(\min_{k \in C'} r(k) + \sum_{k \in C'} p(k)\right)\right\}, \tag{18}$$

where operation i is the predecessor of j in job $J(j)$. This relation can be used to get a recursive formulation of all heads and a similar equation can be made for the tails. Later we will describe an algorithm of Carlier and Pinson [9] which finds the maximum value of (18).

When no disjunctive edges are directed, both procedures the head is defined as the sum of the processing times of the preceding operations of the same job and the

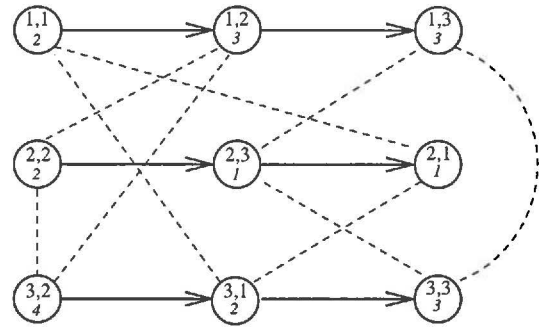

Figure 2a: Graph with no disjunctive edges directed

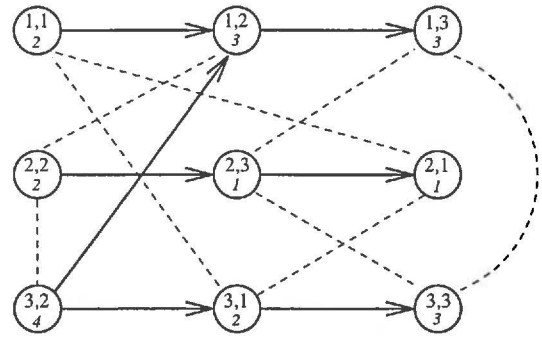

Figure 2b: Resulting graph after directing an edge

Figure 2: A graph representation of the adjustment of heads and tails. The italic numbers represent the processing times. The head of operation 2 of job 1 (operation (1,2)) is 2 before the directing of any disjunctive edge. After directing the edge between operations (3,2) and (1,2) we can update the head of operation (1,2) to 4 because the processing of operation (1,2) cannot start before the ending of operation (3,2). We can update the head of operation (1,3) and the tail of operation (3,2) in the same way.

tail as the sum of the processing times of the succeeding operations. If during the process precedence relations between the operations of the same machine are fixed, the values of the heads and tails can increase. This is shown in Figure 2 for the disjunctive graph.

We know that an update of the head of operation i can be followed by an update of the heads of the successors of i, and similarly an update of a tail by an update of the tails of the predecessors. Applying these statements makes the first part of the maximum in (18) negligible and apparently all branch and bound algorithms use these results. When the heads and tails are updated, we can determine a new lower bound, which will be at least as strong as the previous. The updates are especially useful for lower bounds based on the heads and tails like the prEDD.

One way to direct disjunctive edges is by branching. The algorithms based on the disjunctive graph branch by directing one or more disjunctive edges. Carlier and Pinson [8] introduced conditions to direct disjunctive edges without branching: select (j,i), where (j,i) represents the arc from operation j to i, by proving that scheduling i before j leads to a schedule with a makespan larger than or equal to

the upper bound. We saw that a new directed edge can improve the lower bound. It also reduces the size of the branching tree and can improve the upper bound because more precedence relations have to be respected. These reasons show the importance of the methods to increase the number of directed disjunctive edges.

In the conditions to direct edges without branching the heads and tails are used, where higher values of heads and tails lead to more directed edges. So directing disjunctive edges and updating heads and tails are strongly connected and often considered iteratively until no changes occur. We will give an outline of the conditions and algorithms. The basic ideas are by Carlier and Pinson [8]. They improve their work in [9] and [10]. Brucker, Jurisch, and Krämer [5] and Brucker, Jurisch and Sievers [6] extend the work of Carlier and Pinson, and several other algorithms also use the ideas and procedures.

Define a clique C to be a subset of O_m with at least two elements in which every two elements of C are in disjunction, so between the elements of C no precedence relations are set. We call $e \in C$ an input of C if e is scheduled before all operations of $C \setminus \{e\}$, and equivalently $s \in C$ an output if s is scheduled after all operations of $C \setminus \{s\}$. Further we define $\hat{E} \subseteq C$ as the set of all possible inputs of C, where $i \in C$ is a possible input if there exists a schedule, with makespan smaller than the current upper bound, in which i is scheduled before all elements of C. In the same way we define $\hat{S}$ as the set of all possible outputs. Let E and S be subsets of O_m that contain $\hat{E}$ and $\hat{S}$. These subsets E and S are introduced because it is often very difficult to find the sets $\hat{E}$ and $\hat{S}$. Initially, E and S are equal to C and the following implications can be applied to remove elements from E and S. Let $k \in C$ and UB be the value of the current upper bound (Carlier and Pinson use a strict greater-than sign. When only interested in improving schedules an equality is also sufficient):

$$r(k) + \sum_{j \in C} p(j) + \min_{j \in S \setminus \{k\}} q(j) \geq UB \Rightarrow k \notin \hat{E}; \tag{19}$$

$$\min_{j \in E \setminus \{k\}} r(j) + \sum_{j \in C} p(j) + q(k) \geq UB \Rightarrow k \notin \hat{S}. \tag{20}$$

To see that (19) is valid assume that operation k is scheduled before the elements of C. Under this assumption $r(k) + \sum_{j \in C} p(j) + \min_{j \in S \setminus \{k\}} q(j)$ is a lower bound. So if this value already exceeds the the current upper bound, we know that there does not exist an improving schedule with k scheduled before all elements of C and k can be removed from $\hat{E}$. In the same way we can show that (20) holds. We can also state a stronger result to find inputs or outputs of C. If we find a clique C and a $k \in C$ for which

$$\min_{j \in C \setminus \{k\}} r(j) + \sum_{j \in C} p_j + \min_{j \in C \setminus \{k\}} q(j) \geq UB \tag{21}$$

is valid, we can conclude that k is an input or an output of C. To see this, assume that k is not scheduled as the first or the last job of C. Now we know that $\min_{j \in C \setminus \{k\}} r(j) + \sum_{j \in C} p_j + \min_{j \in C \setminus \{k\}} q(j)$ is a lower bound for the makespan. So the only possibility to improve the current upper bound is to schedule k as the first or the last operation of C.

If we have a $k \in C$ for which (19) and (21) are satisfied, we have found the output of clique C. In the same way we have found the input if conditions (20) and (21) are satisfied. If we have found k to be the output of a clique C we know that all other elements of C have to be scheduled before k, so we can select (j, k) for all $j \in C \setminus \{k\}$. Similarly we can select (k, j) for all $j \in C \setminus \{k\}$ when k has

been determined to be an input of C. Carlier and Pinson [8] derive a special result applicable to each clique C of two operations i and j:

$$r(j) + p(j) + p(i) + q(i) \geq UB \Rightarrow \text{ select } (i,j). \tag{22}$$

Brucker, Jurisch, and Krämer [5] use another approach to direct the disjunctive edges based on the ideas of Dewess [11]. Their first result can be derived directly from (22). We introduce some extra notation. Let $d(j)$ be the due date of operation j, defined by $d(j) := UB - q(j) - 1$. Then we can construct for operation j a time window $[r(j), d(j)]$ in which operation j has to be processed in order to construct a solution better than the current one. If we combine the definition of $d(j)$ and (22) we get:

$$p(j) + p(i) > d(i) - r(j) \Rightarrow \text{ select } (i,j). \tag{23}$$

The interpretation of (23) is that we cannot process i and j in the time interval $[r(j), d(i)]$. We say that the arc (i,j) is selected by a 2-set condition where $\{i,j\}$ is the 2-element set. If we now assume that all the arcs which may be derived from 2-set conditions are selected, we can derive (i,j) from the 3-element set $\{i,j,k\}$ by showing that the sequences j,i,k and j,k,i and k,j,i cannot improve the upper bound. In terms of time windows this means we must show that the three jobs cannot be processed entirely in the time windows $[r(j), d(k)], [r(j), d(i)]$, and $[r(k), d(i)]$. This leads to the following 3-set condition to select (i,j):

$$d(k) - r(j) < s \ \wedge \ d(i) - r(j) < s \ \wedge \ d(i) - r(k) < s, \tag{24}$$

where $s := p(i) + p(j) + p(k)$. Brucker, Jurisch, and Krämer give some complementary results for situations with more information about i,j, and k, and they give an example of an arc (i,j) that can be derived by 3-set conditions and not by the algorithms of Carlier and Pinson. Furthermore they construct an $O(n^2)$ algorithm for deriving all relations using the 3-set conditions. To do this, they modify the time windows using the information of the directed edges. If (i,j) has already been selected, the values of $r(j)$ and $d(i)$ can be updated to $\max\{r(j), r(i) + p(i)\}$ and $\min\{d(i), d(j) - p(j)\}$ respectively. This modification can also be done in $O(n^2)$ time and can be seen as being equivalent to the adjustment of heads and tails in the work of Carlier and Pinson.

As mentioned before, it is recommended to update the heads and tails of the operations after directing disjunctive edges. This can be done by using a longest path algorithm in the corresponding graph with all the directed disjunctive edges being arcs. Carlier and Pinson [8] use complementary results for the inputs and the outputs found by conditions (19), (20), and (21). If i is the output of C we can apply (18) to get:

$$r(i) := \max_{C' \subseteq C \setminus \{i\}} \left(\min_{j \in C'} r(j) + \sum_{j \in C'} p(j) \right), \tag{25}$$

and a similar result holds for inputs.

Carlier and Pinson [9] give an $O(n^2)$ algorithm to find the updates for all the operations of one machine based on the results as stated in (18), (19), (20), and (21). For the updating of heads they define a set $C \subseteq O_m$ to be an ascendant set of operation $k \notin C$, $k \in O_m$ if

$$r(k) + p(k) + \sum_{j \in C} p(j) + \min_{j \in C} q(j) \geq UB \tag{26}$$

and

$$\min_{j \in C} r(j) + \sum_{j \in C} p_j + \min_{j \in C} q(j) + p(k) \geq UB. \tag{27}$$

By (26) we know that operation k cannot be processed before all elements of C and by (27) that operation k should be before or after all elements of C. So operation k is an output of $C \cup \{k\}$. We can apply (25) to update the head of operation k. For the best possible update by using ascendant sets we have to find a set C^* which satisfies:

- For at least one ascendant set C of k, $C^* \subseteq C$,
- For all $C' \subseteq C$, where C is an ascendant set of k:

$$\min_{j \in C'} r(j) + \sum_{j \in C'} p(j) \leq \min_{j \in C^*} r(j) + \sum_{j \in C^*} p(j) \,, \tag{28}$$

- $$r(k) < \min_{j \in C^*} r(j) + \sum_{j \in C^*} p(j) \,, \tag{29}$$

where the first condition implies that we can use the ascendant set update, the second that it is the best update, and the third that we have a real update. When set C^* does not exist, we conclude that we cannot update the head of operation k using the ascendant sets. Carlier and Pinson describe an $O(n^2)$ algorithm based on the preemptive one-machine relaxation that finds $r^*(k)$, the best update by ascendant sets for the head of operation k, and prove that this value equals $\min_{j \in C^*} r(j) + \sum_{j \in C^*} p(j)$. Later they improve the algorithm to an $O(n \log n)$ algorithm [10].

The difference between the use of inputs and outputs and the ascendant set approach is the point of view. In the first one a clique of operations is selected and the goal is to find inputs and outputs; in the second approach an operation is selected for which the best update is determined by using cliques of operations.

3.2.2 Global improvements

The methods that are used to direct disjunctive edges and to update heads and tails discussed so far are based on one-machine relaxations, so the problem is examined from a local point of view. These methods check if it is possible to update heads or tails in the one-machine subproblems and apply these updates to the complete job shop problem. In contrast to these local methods, Carlier and Pinson [10] propose to use the assumption that the tail of operation j can be updated in the entire problem. In terms of starting times this assumption means that operation j is required to start between $r(j)$ and $UB - q^*(j) - 1 - p(j)$, where $q^*(j)$ is the new value of the tail of operation j. If it is proved that this assumption results in a schedule with a value higher than or equal to UB, we know that the tail cannot take this value in any improving schedule, and so the head of operation j can be updated. To prove that there is no improving schedule, Carlier and Pinson use the preemptive one-machine relaxations.

Explicitly this means that we assume that $q(j)$ can be updated to $q^*(j)$ and apply this update and the consequences to all one-machine relaxations. We try to show that at least one preemptive schedule exceeds or equals the current UB. If this is shown we know that there is no improving schedule in which operation j finishes after $UB - q^*(j) - 1$ and so the head of operation j must be at least $UB - q^*(j) - p(j)$.

In the same way an algorithm can be described for the assumption that operation i precedes operation j. If we now show that at least one preemptive one-machine problem has a makespan at least as large as the UB under this assumption, we know that the disjunctive edge between i and j can be replaced by the arc (j, i).

Using these global methods we can construct a new lower bound. Initialize $L = UB$ and apply iteratively both global procedures. Continue until no edges can

be directed anymore for this value of L. Use the updated values of the heads and tails to determine the prEDD. Decrease L and apply the procedure again. Repeat this until the new prEDD exceeds or equals the UB. Then we know that $L+1$ is a lower bound for the problem. The idea of doing a binary search between the upper bound and the lower bound to find a better lower bound was already proposed by Carlier and Pinson in 1990 [9], but the algorithm with global operations was described in 1994 [10].

3.2.3 Shaving

The approach of Brucker, Jurisch, and Sievers [6] showed that the ideas of Carlier and Pinson can be incorporated into a time window approach. The work of Martin and Shmoys [17, 18] confirms this and contains full branch and bound algorithms based on time windows. Here we discuss the bounding procedures, and the iterative improvements of these bounds. In Section 3.4 we will discuss the branching schemes. The algorithms of Martin and Shmoys have been developed for the job shop scheduling feasibility problem, so a value T is given. We define a feasible schedule to be a schedule which is finished at time T.

Martin and Shmoys do not introduce a time window for the processing of an operation but define an interval in which the operation is required to start. We can transform the processing time window to a starting time interval by substracting the processing time of the corresponding operation from the right endpoint of the window. So if we state the interval of Martin and Shmoys for operation j as $[u(j), v(j)]$ we initialize $u(j)$ by the sum of the processing times of the preceding operations of the same job and $v(j)$ by T minus the sum of the processing times of the succeeding operations minus the processing time of operation j.

Of course these intervals are closely related to the values of the heads and tails, and the ideas of updating heads and tails can also be used in a time window approach. Martin and Shmoys call it shaving and define it in the following way: if we can find a value $w(j)$ for which we can prove that there is no feasible schedule when operation j is restricted to start in the interval $[u(j), w(j)]$, we can update the time window to $[w(j)+1, v(j)]$. Due to the symmetry of the problem, it is sufficient to discuss only the shaving of the start of the time window.

The major issue of the shaving procedures is to find the maximal value of $w(j)$ for which we can prove that there exists no feasible schedule when operation j is required to start in the interval $[u(j), w(j)]$, because we want to make the windows as small as posible. In most procedures bisection search is used to find the maximum portion to shave off. The shaving principles were introduced as preprocessing procedures to strengthen the fractional packing lower bound, but a lower bound can be derived directly from the shaving procedures as well. If it is shown that no feasible schedule can be found satisfying the improved time windows for a value of T, $T+1$ is a lower bound for the problem. By doing a bisection search over T we can construct a lower bound.

It turns out that using the shaving algorithms and a bisection search over T gives much faster lower bounds comparable to the fractional packing lower bound. Martin and Shmoys describe several lower bound algorithms based on shaving principles. They divide these algorithms in two groups:

- shaving algorithms based on one-machine relaxations, comparable to the local procedures of Carlier and Pinson;
- shaving algorithms based on the total job shop problem (job shop shave), comparable to the global operations.

Given that the preemptive one-machine problem is solvable in polynomial time, it is natural to use this relaxation to show that there is no feasible schedule when

operation j is restricted to start in the first part of its interval. The assumption that operation j starts in the interval $[u(j), w(j)]$ is in the notation of heads and tails equivalent to the assumption that the value of the tail of operation j is $T - w(j) - p(j)$. So to find w_j^*, i.e. the maximal update for the left endpoint of the time window, we can focus on finding q_j^*, the minimal value of the tail for which we can prove there is no feasible preemptive schedule. Martin and Shmoys use the prEDD to describe a way to find q_j^* without doing a binary search.

We concentrate on operation j and machine $m = M(j)$ and note that

$$\max_{S \subseteq O_m} R(S) + P(S) + Q(S) \leq T, \tag{30}$$

because otherwise there would be no feasible schedule for T. We know that q_j^* exists, but it is not certain that it yields an useful update. The existence implies that when we replace $q(j)$ by q_j^*, we can find a set $S \subseteq O_m \setminus \{j\}$ such that

$$r(S \cup \{j\}) + P(S) + p(j) + \min\{Q(S), q_j^*\} > T. \tag{31}$$

However we know that q_j^* is the minimal value of $q(j)$ for which we can use this statement to prove that there is no feasible schedule. So if we substitute $q_j^* - 1$ for q_j^* in (31) we get:

$$r(S \cup \{j\}) + P(S) + p(j) + \min\{Q(S), q_j^* - 1\} \leq T. \tag{32}$$

Combining these results gives $Q(S) \geq q_j^*$ and $r(S \cup \{j\}) + P(S) + p(j) + q_j^* = T + 1$. We can now derive the value of q_j^* and the new left endpoint of the interval becomes:

$$w_j^* + 1 = r(S \cup \{j\}) + P(S). \tag{33}$$

Martin and Shmoys update the start and the end of the windows of the operations on all the machines iteratively until no changes occur. When comparing this procedure to the ascendant set algorithm of Carlier and Pinson [9] implemented in a time window approach, it can be shown that the final time windows are exactly the same in both procedures. The advantage of the algorithm based on the results of Carlier and Pinson is that it runs faster. A lower bound is determined by running the shaving algorithm based on the ascendant sets until no updates are found and is called *iterated Carlier-Pinson*. Martin and Shmoys also gives their own implementation based on the Carlier and Pinson updates. It has the advantage that it runs faster when it is rerun on a slightly changed problem instance.

Instead of using the preemptive schedule, Martin and Shmoys also try to use the exact one-machine schedule. Although known to be $\mathcal{NP}$-hard this exact one-machine relaxation is solvable in an acceptable amount of computation time for most instances. In spite of the additional computation time invested, the results are not much better than those obtained using the preemptive one-machine relaxation.

Carlier and Pinson [10] introduced global operations to improve their lower bounds. The same idea applied to a time window approach was proposed by Martin and Shmoys [17, 18]. This is what they call *job shop shave*. They assume that an operation must start in one part of the interval and use the one-machine shaving algorithms on all the machines to show that this will not lead to a schedule finished at time T. This gives the algorithms *CP-shave* and *exact one-machine shave*, where they use the preemptive one-machine relaxation and the exact one-machine relaxation in the shaving procedures respectively. Another job shop shave algorithm uses CP-shave to show that the assumption of a smaller time window leads to an infeasible schedule. Because this algorithm uses a job shop shaving algorithm on two levels it is called *double shave*. This bound is tight in most cases but takes an extremely long time to compute.

	MT10		ABZ5		ABZ6	
Optimal	930		1234		943	
prEDD	808		1029		835	
Two-job	655		859		742	
Cuts1	823	*(5.23)*	1074	*(5.61)*	835	*(4.87)*
Cuts2	824	*(305)*	1076	*(611)*	837	*(334)*
Cuts3	827	*(7552)*	1077	*(4971)*	840	*(5257)*
Packing	859	*(226)*	1133	*(234)*	882	*(249)*

Table 2: Lower bounds: results of two-job relaxation, cutting planes and fractional packing bound compared to prEDD.

3.2.4 Multiple machine relaxations

To improve on the one-machine relaxation, a natural step is to use k-machine relaxations. Martin [17] gives some computational results for these bounds and remarks that in a multiple machine relaxation, besides heads, tails, and precedence relations, also lags occur between two operations. These lags consist of the sum of the processing times of the operations to be processed between two operations. Martin gives some computational results of applying shaving algorithms to 2- and 3-machine bounds but notes that CP-shave gave better results in less time, due to the large computation times of the algorithm for solving the k-machine relaxation. Martin does not describe his algorithm in detail. For theory about scheduling with these delayed precedence constraints we refer to an article of Balas, Lenstra, and Vazacopoulos [3], which discusses the one-machine problem with delayed precedence constraints.

3.3 Results on lower bounds

In this section we give some computational results of the lower bounds we discussed. As noted, most algorithms use the prEDD as a lower bound. This is a very fast combinatorial lower bound. The gap between the prEDD and the optimal solutions turns out to be quite large. As alternatives we discussed the two-job relaxation of Brucker and Jurisch [4], a cutting plane approach of Applegate and Cook [2] and the fractional packing lower bound of Martin and Shmoys [17, 18]. The problems in the table are the well known 10×10 problem of Fisher and Thompson [12] and two 10×10 instances posed by Adams, Balas, and Zawack [1], Problem 5 and 6 from their Table 1.

In the tables the numbers in italics are computation times in seconds. For the prEDD the computation times are omitted. Applegate and Cook give a computation time of 0.1 second for the prEDD. The computation times of the packing lower bound do not include the bisection search time over T. The cutting plane lower bounds are computed using an IBM 3081D computer and Martin uses a 90Mz Pentium PC.

The three cutting plane approaches represent respectively the relaxation of the disjunctive model with only the basic cuts, the same relaxation with all the cuts described for this model, and the LP-relaxation of the mixed integer formulation with all cuts given by Applegate and Cook. The three cutting plane lower bounds as well as the packing lower bound of Martin turn out to be too slow

Brucker and Jurisch concluded that the two-job bound was only rewarding when the ratio of the number of jobs to the number of machines is small. The results do not support this conclusion. We refer to the results in the article of Brucker and Jurisch [4] for more specific results. We only illustrated here that in the instances

Problem	MT10		La36		La17	La29	
Optimal	930		1268		784	1152	
prEDD	808		1224		739	1114	
global	868		1233		777		
iterated CP	855	*(0.01)*	1233	*(0.01)*		1119	*(0.01)*
CP shave	919	*(45)*	1267	*(75)*		1119	*(89)*
double shave	930	*(530)*	1268	*(1533)*		1140	*(183465)*

Table 3: Lower bounds: results of the improvements on the lower bounds: global operations and shaving.

of Table 2 the three approaches are not of practical use.

In the discussion of the lower bounds we described in detail the improvements of the lower bounds, consisting of directing edges, adjustment of heads and tails, and shaving procedures. We also discussed the use of these procedures on a global level. Most of these improvement procedures can be seen as preprocessing or as computational efforts in the nodes of the search tree by the fact that they reduce the tree size. As in all branch and bound algorithms, the balance between the computation time in each node and the number of nodes to be investigated constitutes an important trade-off. We also saw that these procedures can be used to determine lower bounds by bisection search over T, the time input variable of the job shop feasibility problem. Here we will give some results of lower bound applications of these procedures in comparison to the prEDD. In Tabel 3 we report the results on the 10×10 problem of Fisher and Thompson [12] and on a 15×15, a 10×10 and a 10×20 instance of Lawrence [16].

In the first two instances the algorithm of Carlier and Pinson [10] can be compared to the algorithms of Martin and Shmoys [17, 18]. A comparison shows that the results of CP shave are better than the lower bound based on the global operations implemented by Carlier and Pinson. The fourth problem is included because it is the only test instance used by Martin for which double shave did not reach the actual optimum.

In this table we see a trade-off between the time invested to reach the lower bounds and the values of the lower bounds. Complexity analysis of Martin shows that CP shave and double shave are both pseudo-polynomial in the sense that they depend polynomially on the schedule length T. The global algorithm of Carlier and Pinson and the double shave algorithm of Martin and Shmoys are so expensive that they do not occur as lower bounding procedures in an algorithm. Both procedures are only used in the root node of the search tree and can be seen as a way of preprocessing. After application of these algorithms the heads and tails (or the corresponding time windows) have such values that the size of the resulting search tree is much reduced.

3.4 Branching schemes

We distinguish three different approaches when considering the branching schemes of the algorithms for solving the job shop problem to optimality:

- branching on edges of the disjunctive graph representation;
- branching on the operations of the critical path based on a block theorem;
- time oriented branching based on time windows.

3.4.1 Disjunctive graph

When using the disjunctive graph representation of the problem it is straightforward to branch on the direction of the disjunctive edges. Every node in the branching tree corresponds to a graph with the elements of A and the directed disjunctive edges as arcs. As said before a graph represents a feasible schedule if and only if all the disjunctive edges have been directed and the graph contains no cycle. The easiest way to use the disjunctive edges for branching is by directing an edge one way or the other to create two new nodes (children). Carlier and Pinson [8] and Applegate and Cook [2] use this branching rule. The only problem left is finding a promising disjunctive edge to branch on.

Carlier and Pinson note that it is beneficial to have sets of possible inputs or outputs (E and S) of low cardinality. If the cardinality of one of these sets is 1, an input or an output is found, and it is possible to direct the disjunctive edges from or to this vertex. A set with cardinality 0 leads to the truncation of the node. So Carlier and Pinson pick a disjunctive edge out of the set E or S with the smallest cardinality. It is easy to see that in both new nodes at least one of the sets E or S contains fewer elements.

In order to determine the order on the machine with the highest initial one-machine relaxation first (this value equals prEDD), Carlier and Pinson consider two operations of this machine. The operations are both elements of E or of S depending on which set has smallest cardinality. When the order of the operations on this machine has been defined, they take among all the other machines the set E or S with smallest cardinality. When it is decided which set to use, the next step is to find the elements to branch on. If E is the set to use for branching and LB is the value of the makespan of the corresponding preemptive one-machine problem, compute for every two elements $i, j \in E$:

$$d_{ij} := \max\{0, r(i) + p(i) + p(j) + q(j) - LB\};$$
$$d_{ji} := \max\{0, r(j) + p(j) + p(i) + q(i) - LB\}; \tag{34}$$
$$v_{\{ij\}} := |d_{ij} - d_{ji}|; \tag{35}$$
$$a_{\{ij\}} := \min\{d_{ij}, d_{ji}\}. \tag{36}$$

We use the operations with maximal $v_{\{ij\}}$ and in case of ties the elements with maximal $a_{\{ij\}}$. The similarity between (22) and (34) explains why we want $v_{\{ij\}}$ to be large. A large value of $v_{\{ij\}}$ means a large difference between $r(i) + q(j)$ and $r(j) + q(i)$ and this implies that one of the directions of the disjunctive edge is unlikely to lead to an improvement. So a fast truncation of one of the two new nodes might be possible. For the same reason we take the operations with the maximum value of $a_{\{ij\}}$ in case of ties, because a large $a_{\{ij\}}$ makes sure that we take the largest d_{ij} among the ties. When the operations i and j are found, we create the children by scheduling i before j to get the first new node and j before i to get the second.

Applegate and Cook [2] use the ideas of Carlier and Pinson in their algorithm. They also tested a greedy rule: try all possible choices of branching and take the edge for which the minimum of the lower bounds (prEDD) in the children achieves the maximum value. The implementation of the greedy rule gave promising results, so Applegate and Cook concluded that branching on a disjunctive edge was a good procedure and they implemented the scheme of Carlier and Pinson in their algorithm.

Later Carlier and Pinson propose a new branching scheme [9]. In this new branching scheme more disjunctive edges are directed simultaneously to create a new node. An advantage is that this leads to better improvements of the heads and tails. A disadvantage is the faster growth of the search tree because branching may

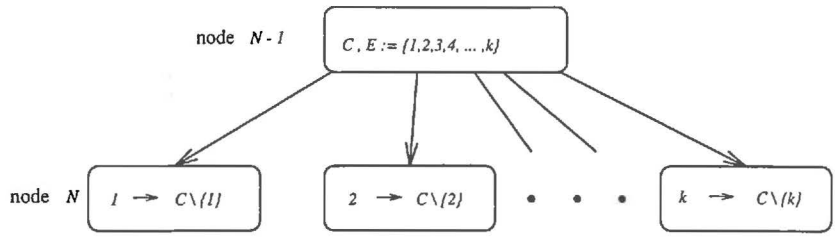

Figure 3: Branching on a set of possible inputs gives | E | new nodes by using each element of E as an input of K.

mean the creation of more than two nodes. Again they use a set E or S with smallest cardinality for branching. When branching is the next step in the algorithm, they determine the prEDD and find the operation for which $C(j)+q(j) = \text{prEDD}$, where $C(j)$ is the completion time of operation j in the corresponding preemptive one-machine schedule. They derive the set $C \subseteq O_{M(j)}$ with $j \in C$, as being the clique with maximal local lower bound and prove that this value equals $C(j) + q(j)$, so:

$$C(j) + q(j) = \min_{i \in C} r(i) + \sum_{i \in C} p_i + \min_{i \in C} q(i). \tag{37}$$

This set C can be seen as a critical set in that it contains the elements which determine the lower bound. Branching on this set probably allows us to use the conditions for directing disjunctive edges efficiently and to truncate new nodes soon. For this set C, determine E and S, and if | E | < | S | use the elements of E to branch as shown in Figure 3. If | S | < | E | use each element of S as the output of C to create the new nodes.

A third branching scheme of Carlier and Pinson [10] is based on the ideas of ascendant sets. If we can find a set and an operation which satisfy one of the two conditions of ascendant sets (i.e. (26),(27)), the extra information can be used by branching on this pair. If we know that operation k cannot be an input of a set C, we can create two new nodes, one in which k is scheduled after all elements of C and the other with k scheduled after at least one operation of C. In this way the validity of one of the conditions for ascendant sets can be used. If there are more pairs of candidates, we take the pair (C, k) for which the value of the second condition of ascendant sets (the one not satisfied) is closest to UB. If no pair satisfies one of the conditions, the scheme takes a disjunctive edge to branch on as in Carlier and Pinson [8].

3.4.2 Block approach

We already saw that a feasible solution of a job shop instance can be represented by a directed graph. The value of the corresponding schedule can be found by determining a longest path in the graph. This so-called critical path can be seen as a sequence of blocks, where a block represents a sequence of operations. A sequence of operations on the critical path is called a block if it contains at least two operations and all the operations have to be processed on the same machine. This block approach was introduced by Grabowski, Nowicki, and Zdrzalka [15] for single-machine scheduling with release dates and due dates. An example of a critical path divided into blocks is given in Figure 4.

Brucker, Jurisch, and Sievers [6] use the block approach for their branching scheme. When they decide to branch, they use a heuristic to get a solution given the set of already directed disjunctive edges, then determine a critical path and use

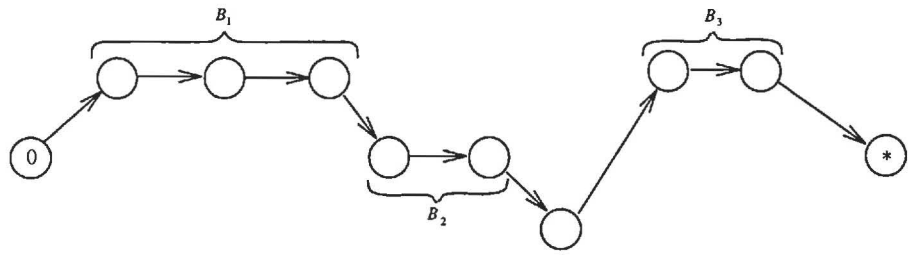

Figure 4: A critical path which contains three blocks. The horizontal arcs are directed disjunctive edges and the diagonal arcs represent elements of A.

the blocks to branch. The branching principles are based on the following theorem (for a proof see [6]):

Theorem 3.1 *Let S be a schedule for a job shop instance with value $v(S)$. Every schedule S' with $v(S') < v(S)$ has at least one operation of any block processed before the first or after the last operation of the corresponding block.*

After finding a solution with a heuristic we know that an improvement is only possible by scheduling one of the internal operations of a block before the block or after the block. Because the arcs between the blocks are elements of A and hence given by the problem instance, any schedule with only permutations of the internal operations of the blocks will contain the same blocks in the critical path. Permuting the operations in the blocks without changing the first and the last one does not change the value of the critical path, because the weights are in the vertices and not in the arcs.

The branching procedure of Brucker, Jurisch, and Sievers exploits Theorem 3.1. When the blocks of the critical path are determined (say, $B_1, \ldots, B_k$), we can define for each block the set of 'before' candidates as the set of operations of the block minus the first operation and the set of 'after' candidates as the operations of the block minus the last operation. Furthermore, we need a permutation on these sets to know which set to use first. When the objective is to direct as many edges as possible, placing the sets in order of non-increasing cardinality will lead to a good permutation. Brucker, Jurisch and Sievers use this permutation with the extra rule that a set of before candidates of a block is always the direct predecessor of the set of after candidates of that block.

The elements in a block are sorted according to non-decreasing heads for before candidates and according to non-decreasing tails for after candidates. When decided which element to use for branching, we can direct the edges between that element and the other elements of the block in the new node. The information of the preceding blocks in the permutation can lead to extra precedence relations. When using block B_i for branching we can fix the first and last operations of the blocks $B_1, \ldots, B_{i-1}$ as being the original first and last operation, because the other possibilities for the first and the last place of each preceding block has been inspected in earlier nodes (a depth-first search is used). After creating a new node with these precedence relations, a cycle check is needed to prevent creating infeasible schedules.

3.4.3 Time windows

The last group of branching rules does not use the disjunctive graph or the disjunctive edges in any way. These rules are time-oriented and developed by Martin and Shmoys [17, 18] to be used in their time-oriented algorithms. Both branching rules

of Martin and Shmoys are based on starting time windows, so a window $[u(j), v(j)]$ is defined for each operation as in Section 3.2.3.

The first rule takes the operation with smallest left endpoint of its window and decides to schedule or to delay it. So branching gives two new nodes, one where the next available operation is scheduled and one where it is delayed. A delayed operation is marked and can only be scheduled again after scheduling another operation the same machine, otherwise it would be possible to schedule an operation after unnecessary idle time.

Another problem in this way of branching is the possiblity that the amount of idle time scheduled after the release date of operation j exceeds the processing time of this delayed operation. This means that in any schedule constructed afterwards operation j can be moved forward without disturbing any other operation. To deal with this Martin and Shmoys introduce tentativily fixed operations. Such an operation is considered to be not fixed during the branching process. When the tentativily fixed operation can be scheduled without disturbing the rest of the schedule it is fixed. This can happen if delaying an operation leads to more idle time than the processing time of one of the formerly delayed operations.

A second time oriented way of branching Martin and Shmoys uses so called α-tight sets. A set $U \subseteq O_m$ is an α-tight set for a value T if

$$R(U) + P(U) + Q(U) \geq T - \alpha. \tag{38}$$

An α-tight set is useful because if a schedule solves the feasibility problem the maximum amount of idle time between the operations of U equals α. This means that one operation of U starts within the interval $[R(U), R(U) + \alpha]$. Branching is done by deciding which operation of U should start in this interval. For this operation k we can replace the time window by $[R(U), R(U)+\alpha]$ and we can update the time windows of the other operations because they cannot start before operation k is completed. A nice result is that the set $U\backslash\{k\}$ is an α-tight set in the node where k is scheduled first. Martin [17] proves that the branches are mutually exclusive if α is smaller than the sum of the processing times of the two shortest operations of U.

3.4.4 Comparison of the branching schemes

The results of the lower bounds can be compared by the computational results. Comparing the branching schemes is more difficult because the algorithms differ not only in branching schemes but also in e.g. lower bounds. We also note that even if the different branching schemes are tested in branch and bound algorithms which only differ in branching schemes, it is questionable to draw conclusions from the results, because each branching scheme may be superior in certain situations.

4 Branch and bound algorithms and results

In this section we describe the different branch and bound algorithms proposed in the literature. The lower bound procedures and the branching schemes have been discussed in the previous section, so this section consists mostly of references to the previous section. In Section 4.2 the computational results of the algorithms are compared. The names of the algorithms introduced in Section 4.1 are used in the presentation. These names are not introduced in the original articles.

4.1 Algorithms

Since the first appearance of the job shop problem in the literature much effort has been devoted to create branch and bound algorithms. The breakthrough in

this area of combinatorial optimization came in 1989 with the algorithm of Carlier and Pinson [8], which solved the legendary 10×10 instance posed by Fisher and Thompson [12].

4.1.1 Carlier and Pinson

Carlier and Pinson [8] introduce in 1989 the possibility to direct disjunctive edges without branching and this makes their algorithm very efficient. Most of the succeeding work is based on these results of Carlier and Pinson as discussed in Section 3.2.1. The algorithm (*CP1*) described in [8], uses the prEDD, the conditions to fix disjunctive edges, the rules to update the heads and tails of inputs and outputs, and a longest path algorithm to update the other heads and tails. The upper bound is a value taken from literature. Branching is done by directing a disjunctive edge one way or the other, selecting the edge as described in Section 3.4.1.

In 1990 Carlier and Pinson [9] describe an improved algorithm (*CP2*) which uses the ascendant sets to update heads and tails. The prEDD is used to give lower bounds and branching is done on sets of possible inputs and outputs. The upper bound is taken from the literature and is reported in the table of results in their article.

An improvement of Carlier and Pinson of their $O(n^2)$ algorithm for updating heads and tails leads to an $O(n \log n)$ algorithm described in 1994 [10]. The branch and bound algorithm (*CP3*) described in the same article uses this new algorithm and the branching scheme based on ascendant sets. Carlier and Pinson describe also an algorithm (*CP4*) that uses the global procedures. They note that the implementation is quite naive, due to the fact that the global algorithm is performed once per operation at each level of the search tree in the order of decreasing processing times.

4.1.2 Applegate and Cook

Although Applegate and Cook [2] gave a nice analysis for a cutting plane lower bound, they use the prEDD in their algorithm (*AC*). The upper bound they use is based on the work of Adams, Balas, and Zawack [1]. Branching is done on a disjunctive edge in the same way as in *CP1*. The rest of the algorithm is based on the conditions to direct disjunctive edges derived by Carlier and Pinson. Before the branch and bound algorithm starts, a heuristic is used to determine an initial upper bound. This upper bound is improved by applying the conditions for directing disjunctive edges to a subset of machines. On the other machines the order of the operations is considered to be fixed. The improvement algorithm is run two times. In the first pass the t machines with the highest values of the preemptive one-machine schedules are fixed, in the second pass the t machines with the lowest values. The number t of fixed machines is an input parameter. With this improved upper bound the actual branch and bound starts in order to reach the optimal solution or to prove that the solution found after the preprocessing steps is optimal. They use the improved upper bound in the procedures to direct disjunctive edges.

4.1.3 Brucker et al.

Brucker, Jurisch, and Sievers [6] use the branching scheme based on the block approach in their branch and bound algorithm (*BJS*). Furthermore they describe an algorithm to update heads and tails and to direct disjunctive edges based on the one described by Carlier and Pinson [9]. The main lower bound is the prEDD, but some other combinatorial lower bounds are used during the determination of the sets of before-candidates and after-candidates of a block and during the computation of heads and tails. The upper bound is based on a priority dispatching rule.

	MT10		LA16		LA17		ABZ5		ABZ6	
	CPU	nodes	CPU	nodes	CPU	nodes	CPU	nodes	CPU	nodes
CP2	253	4336	12	197	11	111	-	-	-	-
CP3	135	3122	9	45	10	53	-	-	-	-
CP4	450	37	55	2	100	12	-	-	-	-
BJS	1138	4242	58	252	15	63	508	2146	31	135
BJK1	1091	4242	56	252	14	63	475	2146	30	135
BJK2	1038	3631	64	255	14	55	508	1936	31	125
AC	372	16055	-	-	-	-	952	57848	91	1269

Table 4: Algorithms: computation times and number of nodes in the search tree for the algorithms based on the disjunctive graph.

Brucker, Jurisch, and Krämer [5] improve the algorithm of Carlier and Pinson [9] to direct disjunctive edges. They describe an $O(\max\{n \log n, f\})$ algorithm that results in the same directed edges, where f is the number of disjunctive edges directed in a loop. They also present an $O(n^2)$ algorithm to check the 3-set conditions. Both algorithms are tested in the branch and bound algorithm of Brucker, Jurisch, and Sievers [6] by substituting the new algorithms for the algorithm of Carlier and Pinson. We will use *BJK1* and *BJK2* to refer to these algorithms, where *BJK2* uses the 3-set conditions.

4.1.4 Martin and Shmoys

Martin and Shmoys [17, 18] introduce different algorithms based on the time window lower bounds and the time oriented branching rules. The first algorithm (*Ma1*) uses *double shave* to reduce the time windows and applies a branch and bound procedure with *iterated Carlier-Pinson* as a lower bound and with branching on the next available operation. The second algorithm (*Ma2*) starts immediately with branch and bound, without the preprocessing of the time window bounds, and uses *C-P shave* as a lower bound and the next available operation branching scheme. The third branch and bound algorithm (*Ma3*) also uses *C-P shave* as lower bounding procedure but branches on α-tight sets.

4.1.5 Perregaard and Clausen

Perregaard and Clausen [19] describe a parallel implementation of branch and bound algorithms for the job shop problem. These algorithms are based on the results of Carlier and Pinson and Brucker et al. Different branching rules and different parallel settings are tested. They distinguish between computing different nodes in parallel and distributing the computational efforts of one node. In the last case several one-machine relaxations are considered in parallel to check for directing disjunctive edges. We will not discuss the algorithms in detail, but give some results in the next section to show that a parallel implementation can be used to solve larger instances.

4.2 Computational results

In this section we compare the algorithms. First we discuss the results of the branch and bound algorithms based on the disjunctive graph formulation. The algorithms of Carlier and Pinson, Brucker et al., and Applegate and Cook belong to this group. For these algorithms we also compare the number of nodes in the search-tree. Later we show that a parallel implementation of the disjunctive graph algorithms gives promising results and we compare the results of the disjunctive graph algorithms to the time oriented algorithms of Martin and Shmoys.

	LA22		LA23		LA30		LA36	
	CPU	nodes	CPU	nodes	CPU	nodes	CPU	nodes
CP2	594	5882	13	70	174	125	7136	4416
CP3	475	4912	12	75	135	102	6201	3910
CP4	4732	73	1081	14	1100	22	3583	23
BJS	6700	10524	3451	6616	239	368	113419	129706
BJK1	6530	10524	3223	6616	229	368	108896	129706
BJK2	6570	8404	3776	6625	174	255	109407	112466

Table 5: Algorithms: computation times and number of nodes in the search tree for some larger problems.

In Table 4 the algorithms based on the disjunctive graph are compared for some 10×10 instances. The problems are respectively the 10×10 instance of Fisher and Thompson [12], Problems 16 and 17 of Lawrence [16], and Problems 5 and 6 of Table 1 of Adams, Balas, and Zawack [1]. In the table no results are reported for the first algorithm of Carlier and Pinson (*CP1*), because for only one of these test instances results are presented, the 10×10 instance of Fisher and Thompson. The results for this problem give an indication of the compuational results of the first algorithm of Carlier and Pinson. The search tree consisted of 22021 nodes and the total computation time was 17982 seconds on a PRIME 2655.

In Table 5 we give some supplementary results for larger problems, all posed by Lawrence [16]. The problems are respectively two 10×15 instances, a 10×20 instance, and a 15×15 instance, where 10×15 means 10 machines and 15 jobs.

The number of nodes in the search tree reduces when more conditions to direct disjunctive edges are used. A comparison of the number of nodes of the algorithm with the global operations (*CP4*) to the number of nodes of the same algorithm without the global operations (*CP3*) confirms this. Also a comparison of the two algorithms of Brucker, Jurisch, and Krämer supports this statement. Furthermore the results show that a reduction of the number of nodes does not automatically implies less computation time.

Drawing conclusions about the algorithms in a fair way is very hard. It is obvious that proclaiming one algorithm to be better than the others is unfair, because the computation times are on different systems and for each test instance the algorithm with smallest computation time can be different. We see for example that the global operations do not work well on the easy instances, like LA17 but very well on the harder (e.g. LA36).

Perregaard and Clausen [19] note that a way to look for a better chance to solving larger job shop scheduling problems is to use faster hardware. They modify some branch and bound algorithms of Carlier and Pinson and of Brucker et al. to run in a parallel setting. The computational results are given to show that a parallel implementation is worth the effort, especially for the more difficult problems. In Table 6 the computational results of the implementation of Perregaard and Clausen are given for a sequential (one processor) setting and two multiprocessor settings. The computation time is reported as well as the speed-up of the multiprocessor settings in comparison to the one processor situation.

Besides the algorithms based on the disjunctive graph formulation we discussed the time-oriented algorithms of Martin and Shmoys [17, 18]. We present some of their results and compare these to the other results in Table 7.

If we compare the results of the three algorithms of Martin and Shmoys, we see that the first algorithm solves more problems than the second, but for the easyinstances the compuatation times are larger. The large running times of *Ma1* are mainly caused by the preprocessing. The double shave algorithm takes most of the computation time. The third algorithms seems to be the most efficient ,since it

problem	1 processor time	8 processors time	8 processors speed-up	16 processors time	16 processors speed-up
MT10	265.6	72.3	3.7	57.9	4.6
LA16	31.2	5.2	6.0	4.7	6.6
LA17	16.9	3.3	5.1	3.2	5.3
LA22	1221.9	261.5	4.7	181.3	6.7
LA23	507.6	88.2	5.8	38.7	13.1
LA30	1917.3	237.2	8.1	133.0	14.4
LA36	1400.0	235.2	6.0	137.3	10.2

Table 6: Parallel algorithms: computation times and speed-up

problem	MA1	MA2	MA3
MT10	483	24	1100
LA19	252	20	455
LA21	18953	-	3588
LA24	2978	2325	4227
LA25	4756	-	5258
LA36	1533	10323	312
LA37	-	-	124
LA38	132564	-	-
LA39	587	210	210
LA40	45951	507	1407

Table 7: Results of the algorithms of Martin and Shmoys

solved most of the problems. LA38 was not solved; due to the large gap between the one-machine bound and the optimal schedule nearly tight sets are hard to find.

If we try to compare all the algorithms, we see that the fastest result on the 'easy' MT10 problem is given by the second algorithm of Martin and Shmoys. The larger 15×15 instance of Lawrence (LA36) was solved with the smallest computation time by the parallel algorithm of Perregaard and Clausen, with the third algorithm of Martin and Shmoys as a good second.

5 Conclusions and remarks

In this section we give some conclusions and ideas for further investigation. In the previous section we mentioned that comparing the algorithms in a fair way is not easy. An algorithm can work well for some benchmark problems and badly for others. For this reason some authors try to divide the problems in different groups.

Carlier and Pinson [8] distinguish problems with a null-gap between the prEDD and the optimal solution and problems with a strictly positive gap. Martin [17] divides the difficult job shop problems in two classes: problems with a tight one-machine lower bound where the optimality proof is difficult and problems with a poor one-machine bound. It is straightforward that some algorithms give good results for one group but perform badly on the problems of the other group. An example is the α-tight branching rule of Martin. For the first group of problems α-tight sets can be found easily for the machine with highest one-machine relaxation. This rule works very well for these problems. On the other hand, extreme problems of the other group, like problem LA38, are not solvable by this rule.

Carlier and Pinson [8] solved the famous problem of Fisher and Thompson for the first time. This was a breaktrough in job shop scheduling. In the discussion of the algorithms we saw that the work of Carlier and Pinson has been a basis for most of the later work in the area. Even the time oriented algorithms of Martin and Shmoys [17, 18] rely on their results. We can conclude that the breaktrough of

Carlier and Pinson has been (and still is) a great influence on the later work.

Finally we will give some ideas for further investigation. These ideas are mostly based on conclusions and remarks of the authors the articles discussed in this survey.

Brucker, Jurisch, and Krämer [5] discussed r-set conditions for $r = 2$ and $r = 3$. We saw a reduction of the number of nodes in the search tree. Deriving 4-set conditions and finding an efficient algorithm to use them probably leads to directing more disjunctive edges. For algorithms based on the disjunctive graph model new possibilities to direct disjunctive edges can be worth the effort. The results of the global algorithm of Carlier and Pinson [10] confirm this, however the algorithm presented is too slow. Ideas for improvement are heuristic rules to apply the global conditions only under special conditions and an improvement of the global algorithm.

In the branch and bound algorithms heuristics play an important role. An example of the use of an heuristic is the determination of an upper bound. We have not discussed these heuristic algorithms in this survey but better heuristics will lead to more efficient branch and bound algorithms.

Martin [17] concludes that his time-oriented approach was promising and suggests that further research in a combination of the purely combinatorial approaches and the time oriented approach can be worth the effort.

In the introduction we mentioned the difficulty of the job shop problem. The 10×10 instance of Fisher and Thompson stayed unsolved for over twenty years. Since this problem has been solved the algorithms have been improved and the computers are much faster but problems of 200 to 250 operations still seem to be very difficult.

Acknowledgements

I would like to acknowledge Professor Jan Karel Lenstra and Professor Jens Clausen for supervising this project and Roy Willemen for his usefull comments and support. I would like to thank Paul Martin for providing information about his work.
The work is done in the algorithms group at the Department of Computer Science of the Copenhagen University. I would like to thank my colleagues at DIKU for my pleasant stay in Denmark. Special thanks to Jens, David, Martin, Stefan, Ole, and Torben.

References

[1] J. Adams, E Balas, and D Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[2] D. Applegate and W. Cook. A computational study of the job shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

[3] E Balas, J.K. Lenstra, and A. Vazacopoulos. The one-machine problem with delayed precedence constraints and its use in job shop scheduling. *Management Science*, 41:94–109, 1995.

[4] P. Brucker and B. Jurisch. A new lower bond for the job-shop scheduling problem. *European Journal of Operational Research*, 64:156–167, 1993.

[5] P. Brucker, B. Jurisch, and A. Krämer. The job-shop problem and immediate selection. *Annals of Operations Research*, 50:73–114, 1994.

[6] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[7] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[8] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[9] J. Carlier and E. Pinson. A practical use of Jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[10] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:238–251, 1994.

[11] G. Dewess. An existence theorem for packing problems with implications for the computation of optimal machine schedules. *Optimization*, 25:261–269, 1992.

[12] H. Fisher and G.L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J.F. Muth and G.L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, 1963.

[13] M.L. Fisher, B.J. Lageweg, J.K. Lenstra, and A.H.G. Rinnooy Kan. Surrogate duality relaxations for job shop scheduling. *Discrete Applied Mathematics*, 5:65–75, 1983.

[14] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flow shop and job shop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[15] J. Grabowski, E. Nowicki, and S. Zdrzalka. A block approach for single machine scheduling with release dates and due dates. *European Journal of Operational Research*, 26:278–285, 1986.

[16] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, 1984.

[17] P. Martin. *A time-oriented approach to computing optimal schedules for the job-shop scheduling problem.* PhD thesis, Cornell University, Ithaca, N.Y., U.S.A., 1996.

[18] P. Martin and D.B. Shmoys. A new approach to computing optimal schedules for the job shop scheduling problem. In W.H. Curnigham, S.T. McCormick, and M. Queyranne, editors, *Lecture Notes in Computer Science; Integer Programming and Combinatorial optimization*, 1996. Proceedings fifth international IPCO conference, Vancouver, Canada, June 1996.

[19] M. Perregaard and J. Clausen. Parallel branch-and-bound methods for the job-shop scheduling problem. To appear in *Annals of Operations Research.*

[20] S.A. Plotkin, D.B. Shmoys, and É. Tardos. Fast approximation agorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[21] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317, 1996.