ELSEVIER Discrete Applied Mathematics 49 (1994) 107–127

DISCRETE APPLIED MATHEMATICS

A branch and bound algorithm for the job-shop scheduling problem*

Peter Brucker**, Bernd Jurisch, Bernd Sievers

FB 6 Mathematik, Universität Osnabrück, D-49069 Osnabrück, Germany

Received 28 January 1991; revised 7 January 1992

Abstract

A fast branch and bound algorithm for the job-shop scheduling problem has been developed. Among other hard problems it solves the 10×10 benchmark problem which has been open for more than 20 years. We will give a complete description of this algorithm and will present computational results.

Key words: Job-shop scheduling; branch and bound method

1. Introduction

The job-shop scheduling problem may be formulated as follows. Consider n jobs $J_1, \ldots, J_n$ and m different machines $M_1, \ldots, M_m$. Each job J_i consists of a number n_i of operations $O_{i1}, \ldots, O_{in_i}$ which have to be processed in this order. Furthermore, assume that operation O_{ik} can be processed only by one machine μ_{ik} ($i = 1, \ldots, n$; $k = 1, \ldots, n_i$). Denote by p_{ik} the corresponding processing time. There is only one machine of each type which can only process one operation at a time. Such an operation must be processed without preemption. Moreover, a job cannot be processed by two machines at the same time. According to these restrictions we have to find an order of all operations O_{ik} with $\mu_{ik} = M_j$ for each machine M_j such that for the corresponding schedule the maximal completion time $C_{\max}$ of all jobs is minimal.

The job-shop problem is a NP-hard problem [13] for which it seems to be extremely hard to find optimal solutions. An indication of this is given by the fact that a 10-job, 10-machine problem formulated in 1963 [14] has been solved only recently [7].

* Supported by the Deutsche Forschungsgemeinschaft (Project JoPTAG).
** Corresponding author.

To find exact solutions of job-shop scheduling problems several branch and bound algorithms have been developed. For many years an algorithm by McMahon and Florian [17] has been the most efficient one. Others were less successful or led to improvements only in some special cases. The first algorithm which solved the 10×10 benchmark problem of Muth and Thompson [14] and proved optimality of the solution was developed by Carlier and Pinson in 1987 [7].

Besides branch and bound methods for finding exact solutions of the job-shop scheduling problem heuristics have been developed. The most popular heuristics in practice are based on priority rules. Others are more sophisticated. Among those a method by Adams et al. [1] has been very successful. Also general purpose heuristics as there are simulated annealing and tabu-search have been applied to job-shop scheduling problems or to some of its generalizations [12, 16]. Good heuristics are also of importance in connection with branch and bound methods. No heuristic with performance guarantee has been developed so far. For most heuristics there exist instances for which these heuristics perform badly.

In this paper a new branch and bound algorithm for the job-shop scheduling problem is presented. It combines two concepts which have recently been developed.

- a generalization of a branching scheme (by Grabowski [10]) which has successfully been applied to one-machine problems with release-dates and due-dates,
- a method to fix disjunctions before each branching step (due to Carlier and Pinson [8]).

Heuristics, methods for lower bound calculations, and data structures used in our implementation have been chosen according to their performance on the benchmark problems of Muth and Thompson [4]. Corresponding experiments have been performed by Jurisch and Sievers [11].

Our algorithm was coded in C. It solves the 10×10 benchmark problem of Muth and Thompson on a workstation in 16 min. It also performs well on other 10×10 problems. Furthermore, our algorithm if used as a heuristic provides better results than the heuristic of Adams et al. when applied to problems of size up to 10×10. We also applied our algorithm to the problems of higher dimension as documented in Adams et al. In most cases we were able to improve the best solution given in this paper.

In this report we give a complete description of the technical details of our algorithm. Section 2 describes the basic ideas of the branch and bound method. Further details are discussed in the subsequent sections. Section 3 describes the branching scheme. Heads and tails which are crucial for the method of fixing disjunctions before branching are introduced in Section 4. Section 5 is devoted to the method of Carlier and Pinson for fixing disjunctions. Methods for calculating lower bounds and heuristics are addressed in Sections 6 and 7. The last section describes details of the implementation and provides computational results.

When writing the code one of the objectives was to be flexible with respect to changes. This was very convenient when testing different versions of the algorithm. It also will be useful in connection with further experiments with modified versions. The C-code is available under ORSEP [5].

2. Solving job-shop problems by branch and bound methods

The most effective branch and bound methods are based on the disjunctive graph model due to Roy and Sussmann [15]. For an instance of the job-shop scheduling problem the disjunctive graph $G = (V, C \cup D)$ is defined as follows. V is the set of nodes, representing the operations of the jobs. Additionally there are two special nodes, a source 0 and a sink $*$. Each node i has a weight which is equal to the processing time p_i of the corresponding operation, whereby p_0 and p_* are equal to 0. C is the set of conjunctive arcs which reflect the job-order of the operations. For every pair of operations that require the same machine there is an undirected, so-called disjunctive arc. The set of all these arcs is denoted by D. Fig. 1 shows an example of a problem with 4 jobs and 4 machines.

The basic scheduling decision is to define an ordering between all those operations which have to be processed on the same machine, i.e. to fix precedence relations between these operations.

In the disjunctive graph model this is done by turning undirected (disjunctive) arcs into directed ones. A set of these "fixed" disjunctions is called *selection*. Obviously a selection S defines a feasible schedule if and only if

- every disjunctive arc has been fixed and
- the resulting graph $G(S) = (V, C \cup S)$ is acyclic.

In this case we call the set S a *complete selection*. Fig. 2 shows a selection that defines a feasible schedule.

For a given schedule (i.e. a complete selection S) the maximal completion time of all jobs $C_{\max}$ is equal to the length of the longest weighted path from the source 0 to the sink $*$ in the acyclic graph $G(S) = (V, C \cup S)$. This path is usually called *critical path*.

Now we will give a short description of the branch and bound algorithm for the job-shop scheduling problem. The algorithm will be represented by a search tree. Initially, the tree contains only one node, the root. No disjunctions are fixed in this node, i.e. it represents all feasible solutions of the problem. The successors of the root are calculated by fixing disjunctions. The corresponding disjunctive graph represents all solutions of the problem respecting these disjunctions. After this each successor is recursively handled in the same way. The examination of a search tree node stops if it

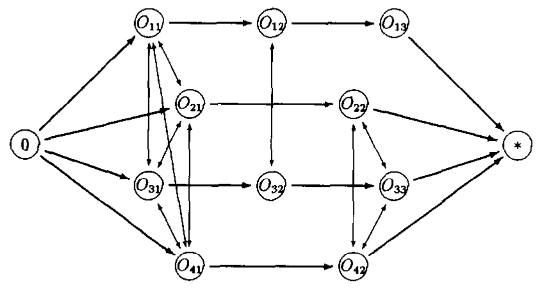

Fig. 1.

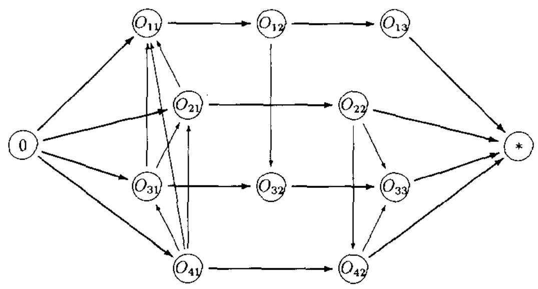

Fig. 2.

represents only one solution (i.e. the set S of fixed disjunctive arcs is a complete selection), or it can be shown that the node does not contain an optimal solution.

More precisely: every search tree node r corresponds with a graph $G(FD_r) = G(V, C \cup FD_r)$. FD_r denotes the set of fixed disjunctive arcs in node r. The node r represents all solutions $Y(r)$ representing the partial order given by FD_r. Branching is done by dividing $Y(r)$ into disjoint subsets $Y(s_1), \ldots, Y(s_q)$. Each $Y(s_i)$ is the solution set of a problem with a graph $G(FD_{s_i}) = G(V, C \cup FD_{s_i})$ where $FD_r \subset FD_{s_i}$ which means that $G(FD_{s_i})$ is derived from $G(FD_r)$ by fixing additional disjunctions. This way of branching creates immediate successors $s_1, \ldots, s_q$ of node r in the branching tree which are recursively treated. For each node r a value $LB(r)$ bounding the objective values of all solutions in $Y(r)$ from below is calculated. We set $LB(r) = \infty$ if the corresponding graph $G(FD_r)$ has a cycle. Furthermore, we have an upper bound UB for the solution value of the original problem. UB is updated each time when a new feasible solution is found which improves UB.

To specify the branch and bound procedure in more detail we have

(a) to introduce a branching scheme,

(b) to discuss methods for calculating bounds.

The following sections are devoted to these issues.

3. A branching scheme

The branching scheme we used in our implementation is based on an approach used by Grabowski et al. [10] in connection with single-machine scheduling with release-dates and due-dates. It is based on a feasible schedule which corresponds to a disjunctive graph $G(S) = (V, C \cup S)$ where S is a complete selection. Let P be a critical path in $G(S)$ and let $L(S)$ be the length of P. A sequence $u_1, \ldots, u_k$ of successive nodes in P is called a *block* in $G(S)$ if the following two properties are satisfied:

(a) The sequence contains at least two nodes.

(b) The sequence represents a maximal number of operations which have to be processed on the same machine.

We denote the jth block on the critical path by B_j. See Fig. 3 for blocks and conjunctive arcs on a critical path.

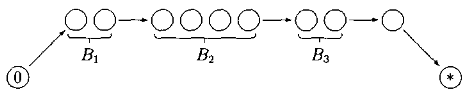

Fig. 3.

The following theorem is the basis of the considerations in this section.

Theorem 3.1. *Let S be a complete selection corresponding to some solution of the job-shop scheduling problem. If there exists another complete selection S' such that $L(S') < L(S)$, at least one operation of one block in $G(S)$ has to be processed before the first or after the last operation of the corresponding block.*

Proof. Let $P = (0, u_1^1, u_2^1, \ldots, u_{m_1}^1, \ldots, u_1^k, u_2^k, \ldots, u_{m_k}^k, *)$ be a critical path in $G(S) = (V, C \cup S)$. $u_1^j, \ldots, u_{m_j}^j$ $(j = 1, \ldots, k)$ denotes a maximal number of operations which have to be processed on the same machine (i.e. a block if $m_j > 1$). Assume that there is a complete selection S' with $L(S') < L(S)$, and no operation of any block of S is processed before the first or after the last operation of the corresponding block. Therefore $G(S') = (V, C \cup S')$ contains the arcs

$$u_1^j \to u_i^j \ (j = 1, \ldots, k;\ i = 2, \ldots, m_j),$$

$$u_i^j \to u_{m_j}^j \ (j = 1, \ldots, k;\ i = 1, \ldots, m_j - 1).$$

Thus, $G(S')$ contains a path

$$(0, u_1^1, v_2^1, \ldots, v_{m_1-1}^1, u_{m_1}^1, \ldots, u_1^k, v_2^k, \ldots, v_{m_k-1}^k, u_{m_k}^k, *),$$

where the sequence $v_2^j, \ldots, v_{m_j-1}^j$ is a permutation of $u_2^j, \ldots, u_{m_j-1}^j$ $(j = 1, \ldots, k)$. The length of the critical path in $G(S')$ is not less than the length of this path. Let $v_1^j = u_1^j$ and $v_{m_j}^j = u_{m_j}^j$ for $j = 1, \ldots, k$. Then we have

$$\begin{aligned} L(S') &\geqslant \sum_{j=1}^{k} \left(\sum_{i=1}^{m_j} p_{v_i^j} \right) \\ &= \sum_{j=1}^{k} \left(\sum_{i=1}^{m_j} p_{u_i^j} \right) \\ &= L(S), \end{aligned}$$

which is a contradiction. □

The following fact is an immediate consequence of the previous theorem:

If there are two complete selections S, S' with $L(S') < L(S)$ then at least one of the two conditions (i) or (ii) holds:

(i) At least one operation of one block B in $G(S)$, different from the first operation in B has to be processed *before* all other operations of B in the schedule defined by $G(S')$.

(ii) At least one operation of one block B in $G(S)$, different from the last operation in B has to be processed *after* all other operations of B in the schedule defined by $G(S')$.

Now consider a node r of the search tree and a solution $y \in Y(r)$. Usually, y is calculated using some heuristic. Let S be the complete selection corresponding with y. A critical path in $G(S)$ defines blocks $B_1, \dots, B_k$. For block $B_j: u_1^j, \dots, u_{m_j}^j$ the operations in

$$E_j^B := B_j \setminus \{u_1^j\} \quad \text{and} \quad E_j^A := B_j \setminus \{u_{m_j}^j\}$$

are called *before-candidates* and *after-candidates*, respectively.

For each before-candidate (after-candidate) an immediate successor s of the search tree node r is generated by moving the candidate before (after) the corresponding block. An operation $l \in E_j^B$ is moved before block B_j by fixing the arcs $\{l \to i: i \in B_j \setminus \{l\}\}$. Similarly, $l \in E_j^A$ is moved by fixing $\{i \to l: i \in B_j \setminus \{l\}\}$.

Additional arcs can be fixed due to the following ideas. Let s be an immediate successor of r generated by moving an operation $l \in E_j^B$. After backtracking from s all solutions in $Y(r)$ with additional precedence constraints $\{l \to i: i \in B_j \setminus \{l\}\}$ have been inspected. During the processing of s and all its successors a new upper bound UB may have been found. All solutions which are calculated later on in other successors of r and improve UB have the property that l is *not* processed before all other operations of B_j. Such a solution would already have been found in s (or in its successors).

Now consider an arbitrary permutation $E_1, \dots, E_{2k}$ of all sets E_j^B and E_j^A. We call E_t a predecessor set of $E_{t'}$ ($E_t < E_{t'}$) if $t < t'$. The permutation defines a hypothetical branching order: if we generate a successor s of r by moving a candidate $l \in E_t$, we assume that all search tree nodes corresponding to candidates $l' \in E_1 \cup \cdots \cup E_{t-1}$ have already been processed. Due to the arguments given above the following disjunctive arcs can be fixed in search tree node s:

$$F_j = \{u_1^j \to i: i = u_2^j, \dots, u_{m_j}^j\}$$

for each predecessor set E_j^B of E_t and

$$L_j = \{i \to u_{m_j}^j: i = u_1^j, \dots, u_{m_j-1}^j\}$$

for each predecessor set E_j^A of E_t.

To summarize a branching of the search tree node r which is based on the permutation $E_1, \dots, E_{2k}$ may be defined as follows.

For each operation $l \in E_j^B$ generate a search tree node by fixing $FD_s = FD_r \cup S_l^B$ with

$$S_l^B = \bigcup_{E_i^B < E_j^B} F_i \cup \bigcup_{E_i^A < E_j^B} L_i \cup \{l \to i: i \in B_j \setminus \{l\}\}.$$

For each operation $l \in E_j^A$ generate a search tree node by fixing $FD_s = FD_r \cup S_l^A$ with

$$S_l^A = \bigcup_{E_i^B < E_j^A} F_i \cup \bigcup_{E_i^A < E_j^A} L_i \cup \{i \to l: i \in B_j \setminus \{l\}\}.$$

Due to Theorem 3.1 and the arguments given above all solutions in $Y(r)$ which may improve the actual upper bound are considered in the immediate successors of r. Moreover, it is easy to see that we have $Y(s) \cap Y(s') = \emptyset$ for each pair s, s' of immediate successors of r.

Note that we can inspect the immediate successors of search tree node r in an arbitrary order. The hypothetical branching order given by the permutation $E_1, \ldots, E_{2k}$ does only influence the sets of fixed disjunctive arcs in the successor nodes.

The construction is illustrated in the following example.

Example 3.2. Consider a critical path with two blocks of the form

$$P: \quad \textcircled{0} \rightarrow \underbrace{\textcircled{1} \rightarrow \textcircled{2} \rightarrow \textcircled{3} \rightarrow \textcircled{4}}_{B_1} \rightarrow \underbrace{\textcircled{5} \rightarrow \textcircled{6} \rightarrow \textcircled{7}}_{B_2} \rightarrow \textcircled{*}$$

If we take the permutation

$$E_1 = E_2^B, \qquad E_2 = E_1^A, \qquad E_3 = E_2^A, \qquad E_4 = E_1^B,$$

we get the arc-sets shown in Fig. 4.

Note that in S_5^A and S_4^B we have the cycles $5 \rightarrow 6 \rightarrow 5$ and $4 \rightarrow 3 \rightarrow 4$. Cycles may also be created in connection with the arcs in FD_r which have been fixed before. If cycles are created the corresponding sets $Y(s)$ of feasible solutions are empty.

It is advantageous to check the cycles of length two during the calculation of the before- and after-candidates in a search tree node r. For the block B_l: $u_1^l, \ldots, u_{m_l}^l$ this means if a disjunctive arc $i \rightarrow j$ $(i, j \in B_l)$ is already fixed in the actual search tree node then operation j (operation i) is not inserted into the set E_l^B (E_l^A). The cycles in Example 3.2 will be eliminated by this method. A *complete* cycle-check is done during the computation of heads and tails (see Section 4).

So far we have not specified how to choose the permutation $E_1, \ldots, E_{2k}$ of the sets E_j^B and E_j^A $(j = 1, \ldots, k)$. Our objective is to fix a large number of disjunctive arcs as early as possible. So, we arrange the sets E_j^B and E_j^A $(j = 1, \ldots, k)$ according to non-increasing cardinality of the corresponding blocks. In addition we always take the set E_j^A as a direct successor of the set E_j^B. More precisely, we choose

$$E_{2i-1} := E_{\pi(i)}^B, \qquad E_{2i} := E_{\pi(i)}^A \quad (i = 1, \ldots, k)$$

with a permutation π of $1, \ldots, k$ such that $|B_{\pi(i)}| \geqslant |B_{\pi(j)}|$ if $i < j$.

Moreover, this order can be modified to eliminate successor nodes in the search tree introduced by before-candidates in the first block B_1 and after-candidates in the last block B_k of the critical path. The elimination of such candidates is based on ideas in [10] (for details we refer to this paper).

Now we are able to formulate a recursive branch and bound procedure based on the branching rule introduced in this section.

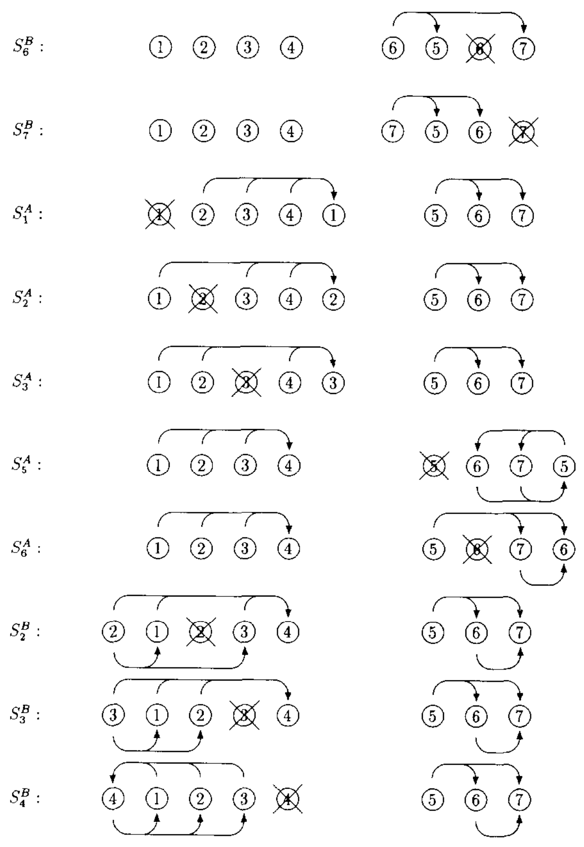

Fig. 4.

```
PROCEDURE Branch and Bound ($r$)
  BEGIN
    Calculate a solution $S \in Y(r)$ using heuristics;
    If $C_{\max}(S) < UB$ THEN $UB := C_{\max}(S)$;
    Calculate a critical path $P$;
    Calculate the blocks of $P$;
    Calculate the sets $E_j^B$ and $E_j^A$;
    WHILE there exists an operation $i \in E_j^v$ with $j = 1, \ldots, k$ and $v = A, B$ DO
      Delete $i$ from $E_j^v$;
      Fix disjunctions for the corresponding successor $s$;
      Calculate a lower bound $LB(s)$ for node $s$;
      IF $LB(s) < UB$ THEN Branch and Bound ($s$)
    END
  END
```

Note that the handling of a search tree node stops if

- the lower bound $LB(s)$ is greater or equal than UB (this is the case if the corresponding disjunctive graph has cycles, i.e. $LB(s) = \infty$) or
- the critical path of the heuristic solution calculated for S does not contain any block or
- the sets E_j^B and E_j^A are empty for all blocks B_j.

We did not specify in which order the operations $i \in E_j^\nu$ are chosen, i.e. the order of the successors of a search tree node r. We tested different methods, especially arrangements based on lower-bound calculations. Based on our experiences the following method should be recommended: sort the candidates according to non-decreasing heads of before-candidates and tails of after-candidates (for the definition of heads and tails see Section 4) and handle the successors of a search tree node according to this order.

4. Heads and tails

With each operation i we may associate a head and a tail. Heads and tails are important data, e.g. for lower bound calculations. They are also used in heuristics.

Calculations of heads and tails are based on all conjunctive arcs and the fixed disjunctive arcs. Thus, they depend on the specific search tree node r.

A *head* r_i of operation i is an earliest possible starting time of i.

A *tail* q_i of operation i is a lower bound for the time period between the finish-time of operation i and the optimal makespan.

A simple way to get a head r_i for operation i would be to calculate the length of the longest weighted path from 0 to i in the disjunctive graph $G = (V, C \cup FD_r)$. Similarly for each operation i the tail q_i could be defined by the length of the longest weighted path from i to $*$ in $G = (V, C \cup FD_r)$.

To obtain great lower bounds it is desirable to have great heads and tails. For this purpose the following more sophisticated procedures for calculating heads have been developed.

If $P(i)$ is the set of disjunctive predecessors of operation i in a search tree node, obviously the value

$$\max_{J \subseteq P(i)} \left\{ \min_{j \in J} r_j + \sum_{j \in J} p_j \right\}$$

defines a lower bound for the earliest possible starting time of operation i. Using the head of the conjunctive predecessor $h(i)$ of i we get the lower bound $r_{h(i)} + p_{h(i)}$. Using these formulas, we may recursively define the head r_i of an operation i:

$$r_0 := 0;$$

$$r_i := \max \left\{ r_{h(i)} + p_{h(i)};\ \max_{J \subseteq P(i)} \left\{ \min_{j \in J} r_j + \sum_{j \in J} p_j \right\} \right\}.$$

The same ideas lead to a formula for the tails q_i of all operations:

$$q_*:= 0;$$

$$q_i:= \max\left\{p_{k(i)} + q_{k(i)};\ \max_{J \subseteq S(i)}\left\{\sum_{j\in J} p_j + \min_{j\in J} q_j\right\}\right\}.$$

Here $k(i)$ is the conjunctive successor of i, and $S(i)$ denotes the set of disjunctive successors of i.

The calculation of heads can be combined with a cycle-checks as follows. We call an operation a labelled operations if its head is calculated. Furthermore, we keep a set D of all operations which can be labelled next, i.e. all unlabeled operations with the property that all their predecessors are labelled. Initially $D = \{0\}$. If we label an operation $i \in D$ then i is eliminated from D and all successors of i are checked for possible insertion into D. The procedure continues until D is empty.

The disjunctive graph $G = (V, C \cup FD_r)$ contains no cycle if and only if a new head has been assigned to the fictive operation $*$ by this procedure. It is not difficult to prove this property which is due to the special structure of G.

In the following sections we will show how to use heads and tails in different parts of the branch and bound algorithm.

5. Fixing additional disjunctive arcs

One of the objectives of the branching scheme introduced in Section 3 was to add large numbers of fixed disjunctions to the set FD_r when going from search tree node r to its successors. A fast increase of the sets of fixed disjunctions is essential for the quality of a branch and bound algorithm because

- More successors s of r contain cycles in the disjunctive graph and need not be inspected furthermore (see Section 3).
- Generally, the value of the lower bound for the optimal makespan increases because more fixed disjunctive arcs have to be respected.
- If we have the additional information that j has to succeed i in any solution which improves a current upper bound then a heuristic will not look for schedules where j is processed before i. Therefore, such a heuristic generally calculates better solutions.

In this section we will present a method due to Carlier and Pinson [8] which fixes additional disjunctive arcs between jobs belonging to a set of operations which have to be processed on the same machine. The method which is independent of the branching process uses an upper bound UB for the optimal makespan and simple lower bounds. It is based on the following inequalities (5.1) and (5.2).

Let I be the set of all operations which have to be processed on a given machine. Furthermore, let $c \in I$ and $J \subseteq I\backslash\{c\}$.

$$\min_{j\in J \cup \{c\}} r_j + \sum_{j\in J\cup\{c\}} p_j + \min_{j\in J} q_j \geqslant UB, \tag{5.1}$$

$$\min_{j\in J} r_j + \sum_{j\in J\cup\{c\}} p_j + \min_{j\in J\cup\{c\}} q_j \geqslant UB. \tag{5.2}$$

The left-hand side of inequality (5.1) ((5.2)) defines a lower bound for all schedules in which c is not processed after (before) all operations of J.

From now on we will be only interested in solutions S with $C_{\max}(S) < UB$. Only these schedules will be called solutions.

The previous arguments immediately lead to the following lemma.

Lemma 5.1. *Let $c \in I$, $J \subseteq I \backslash \{c\}$.*

(a) *If inequality* (5.1) *holds then in all solutions operation c has to be processed after all operations of J.*

(b) *If inequality* (5.2) *holds then in all solutions operation c has to be processed before all operations of J.*

If condition (5.1) holds for an operation c and a set J, then the pair (J, c) is called *primal pair*. In this case all disjunctive arcs $\{j \to c\colon j \in J\}$ can be fixed. We call these arcs *primal arcs*.

Similarly, a pair (c, J) is called *dual pair* if condition (5.2) holds for the set J and the operation c. The disjunctive arcs $\{c \to j\colon j \in J\}$ which can be fixed are called *dual arcs*.

For $|J| = 1$, we can also use the following lemma.

Lemma 5.2. *Let $c, j \in I$, $c \neq j$. If*

$$r_c + p_c + p_j + q_j \geqslant UB, \tag{5.3}$$

j has to be processed before c in every solution.

If inequality (5.3) holds, we can fix the disjunctive arc $j \to c$. This arc is called a *direct arc*.

The following procedure fixes all direct arcs for the set I. Its complexity is $O(|I|^2)$.

```
PROCEDURE Select
  BEGIN
    FOR ALL $c, j \in I, c \neq j$ DO
      IF $r_c + p_c + p_j + q_j \geqslant UB$ THEN
        fix direct arc $(j \to c)$;
  END
```

Next we will derive an efficient method for calculating all useful information associated with all primal pairs (J, c) for a given operation c. The corresponding information for all dual pairs may be calculated similarly.

If (J, c) is a primal pair then operation c cannot start before

$$r_J = \max_{J' \subseteq J} \left\{ \min_{j \in J'} r_j + \sum_{j \in J'} p_j \right\}.$$

If r_c is less than r_J, we can set r_c equal to r_J.

In this case all primal arcs $\{j \to c\colon j \in J\}$ can be calculated by the procedure Select.

Lemma 5.3. *Let (J, c) be a primal pair, $r_c \geqslant r_J$. Then all primal arcs associated with (J, c) are fixed by the procedure Select.*

Proof. For the primal pair (J, c) we have

$$\min_{j \in J} r_j + \sum_{j \in J} p_j + p_c + \min_{j \in J} q_j \geqslant UB.$$

Therefore,

$$\begin{aligned} r_c + p_c + p_i + q_i &\geqslant r_J + p_c + p_i + q_i \\ &\geqslant \min_{j \in J} r_j + \sum_{j \in J} p_j + p_c + q_i \\ &\geqslant UB \end{aligned}$$

for all operations $i \in J$.

The arcs $i \to c$ are fixed by the procedure Select. □

Now consider a primal pair (J^*, c) with $r_{J^*} \geqslant r_J$ for all primal pairs (J, c). If we set r_c equal to r_{J^*}, all arcs associated with all primal pairs (J, c) are fixed by the procedure Select.

These ideas lead to the following problem.

Primal problem 5.4. Let $c \in I$. Does there exist a primal pair (J, c) such that $r_c < r_J$? If it exists, find

$$r_{J^*} = \max\{r_J \colon (J, c) \text{ is a primal pair}\}.$$

For the solution of the Primal Problem 5.4, we look at the so-called Jackson Preemptive Schedule (JPS) for all operations of the set I. This schedule is the solution of the following problem.

Let I be a set of operations which have to be processed on one machine. Associated with each operation i there is a release-date r_i, a processing time p_i and a tail q_i. Find a preemptive schedule such that the value

$$C_{\max} = \max_{i \in I} \{C_i + q_i\}$$

is minimized. C_i denotes the completion time of operation i.

This problem can be solved by the following rule: at each time t where t is a release-date or a finish-time of an operation schedule an unfinished operation i with $r_i \leqslant t$ and $q_i = \max\{q_j \colon r_j \leqslant t\}$. Carlier [6] showed that an algorithm based on this rule has time complexity $O(n \log n)$ with $n = |I|$.

Fig. 5 shows an example for a JPS with 6 operations.

Note that the $C_{\max}$-value of a JPS for the set I gives a lower bound for all solutions in the corresponding search tree node. Furthermore, we assume that $C_{\max} < UB$. Otherwise, the corresponding search tree node can be dropped.

i	1	2	3	4	5	6
r_i	4	0	9	15	20	21
p_i	6	8	4	5	8	8
q_i	20	25	30	9	14	16

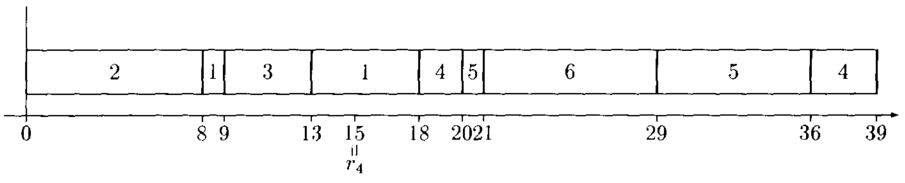

Fig. 5.

Given a JPS for a set of operations I we assume that $c \in I$ is fixed. Then denote by C_j, the completion time of operation j, by K^+, the set of operations j with higher q-priority than c, which are completed after time r_c:

$$K^+ = \{j \in I \colon q_j > q_c,\ C_j > r_c\}$$

and by p_j^+, the remaining processing time of operation j after time r_c ($p_j^+ > 0$ for all $j \in K^+$).

In Fig. 5, we have $K^+ = \{1, 5, 6\}$ for $c = 4$. For a subset $K \subseteq K^+$ we define

$$t_K = r_c + p_c + \sum_{j \in K} p_j^+ + \min_{j \in K} q_j.$$

Now a second primal problem can be formulated.

Primal problem 5.5. Let there be a given JPS for the operations of the set I. Let $c \in I$. Does there exist a non-empty subset $K \subseteq K^+$ such that $t_K \geqslant UB$? If it exists, find such a subset K^* with maximal cardinality.

If the Primal problem 5.5 is solvable, then there exists only one set K^* with maximal cardinality. This can easily be shown as follows. Let there be two subsets K_1, K_2 of K^+ with $t_{K_1} \geqslant UB$ and $t_{K_2} \geqslant UB$ and assume w.l.o.g. that $\min_{j \in K_1} q_j \leqslant \min_{j \in K_2} q_j$. Then for the set $K = K_1 \cup K_2$ we have

$$\begin{aligned} t_K &= r_c + p_c + \sum_{j \in K} p_j^+ + \min_{j \in K} q_j \\ &\geqslant r_c + p_c + \sum_{j \in K_1} p_j^+ + \min_{j \in K_1} q_j \\ &\geqslant UB. \end{aligned}$$

Due to Carlier and Pinson [8] we have the following theorem.

Theorem 5.6. *Let $c \in I$. The Primal problem 5.4 is solvable if and only if the Primal problem 5.5 is solvable. If it is, we have for the corresponding sets J^* and K^*:*

$$r_{J^*} = \max_{j \in K^*} C_j.$$

For the solution of the Primal problem 5.5 we have to calculate the set $K^* \subseteq K^+$ for a given operation c. This problem becomes easy because if K^* is not empty it has the following form:

$K^* = \{j \in K^+ : q_j \geqslant q_{j_0}\}$ for some operation $j_0 \in K^*$.
Otherwise, there would exist an operation $i \in K^+ \setminus K^*$ with $q_i \geqslant \min_{j \in K^*} q_j$ which implies

$$\begin{aligned} t_{K^* \cup \{i\}} &= r_c + p_c + \sum_{j \in K^* \cup \{i\}} p_j^+ + \min_{j \in K^* \cup \{i\}} q_j \\ &\geqslant r_c + p_c + \sum_{j \in K^*} p_j^+ + \min_{j \in K^*} q_j \\ &= t_{K^*} \geqslant UB. \end{aligned}$$

But this is a contradiction to the assumption that K^* is the maximal cardinality subset of K^+ with $t_{K^*} \geqslant UB$.

Due to Carlier and Pinson [8] the value $\max_{j \in K^*} C_j$ is given by

$$\max_{j \in K^*} C_j = \max_{J \subseteq K^*} \left\{ \max\left\{ r_c, \min_{j \in J} r_j \right\} + \sum_{j \in J} p_j^+ \right\}. \tag{5.4}$$

This value is equal to the completion time of the schedule we get if we schedule the operations $j \in K^*$ with processing times p_j^+ after time r_c in the order of non-decreasing heads.

Now we are able to solve the Primal problem 5.4 for a given operation $c \in I$ as follows:

(1) Calculate the JPS up to time r_c.
(2) Calculate the set K^+.
(3) Calculate the operation $j_0 \in K^+$ with smallest tail such that the inequality

$$r_c + p_c + \sum_{\{j \in K^+ : q_j \geqslant q_{j_0}\}} p_j^+ + q_{j_0} \geqslant UB$$

holds. If there does not exist such an operation, the Primal problem 5.5 is unsolvable. Otherwise, set

$$K^* = \{j \in K^+ : q_j \geqslant q_{j_0}\}.$$

(4) If $K^* \neq \emptyset$, calculate $r_{J^*} = \max_{j \in K^*} C_j$ using (5.4). Set $r_c = r_{J^*}$.

The overall complexity for the calculation of the JPS is $O(n \log n)$ where $n = |I|$. If the JPS has been computed up to time r_c, the set K^+ can be calculated in $O(n)$ time. The set K^* can be computed by scanning the operations of K^+ in the order of non-decreasing tails. If we use a sorted list, this can also be done in $O(n)$ time. Finally, we can calculate the value r_{J^*} with complexity $O(n)$. This is done by scanning the operations of K^* in the order of non-decreasing heads using (5.4). We also have to use a sorted list here. Because we have to solve n primal problems, we can fix all primal arcs in time $O(n^2)$.

The calculation of all additional disjunctive arcs is done between the calculation of a lower bound and the computation of a heuristic solution. In detail we proceed as follows:

(1) calculation of all primal arcs for all machines,
(2) calculation of new heads and tails,
(3) calculation of all dual arcs of all machines,
(4) calculation of new heads and tails.

New heads and tails are computed in steps 2 and 4 because the additional arcs influence the heads and tails of all operations. Steps 1–4 should be repeated as long as new disjunctive arcs are fixed.

6. Calculation of lower bounds

Let r be a search tree node with a set FD_r of fixed disjunctive arcs. Based on the arcs FD_r for each operation i a head r_i and a tail q_i is given. A lower bound $LB(s)$ is calculated for each successor s of r. If this value exceeds the actual upper bound UB then an inspection of s is not necessary.

We tested different methods for calculating lower bounds, especially 1-machine and 2-job-relaxations [4]. It turned out to be optimal to compute different lower bounds at different places of the algorithm:

(1) Lower bound calculation during the computation of the sets E_l^B and E_l^A: if operation i should be moved before block B, all disjunctive arcs $\{i \to j : j \in B \setminus \{i\}\}$ are fixed. Thus the value

$$r_i + p_i + \max\left\{ \max_{j \in B\setminus\{i\}} (p_j + q_j);\ \sum_{j \in B\setminus\{i\}} p_j + \min_{j \in B\setminus\{i\}} q_j \right\}$$

is a simple lower bound for the search tree node s. Similarly, the value

$$\max\left\{ \max_{j \in B\setminus\{i\}} (r_j + p_j);\ \min_{j \in B\setminus\{i\}} r_j + \sum_{j \in B\setminus\{i\}} p_j \right\} + p_i + q_i$$

is a lower bound for the node s if i should be moved after block B.

(2) Lower bound calculation during the computation of heads and tails: if the value $r_i + p_i + q_i$ of an operation i exceeds the actual upper bound, the node does not need to be inspected. Also the head r_* of the sink and the tail q_0 of the source of the disjunctive graph are used as lower bounds for all solutions in the search tree node s.

(3) Lower bound calculation after the computation of heads and tails: the Jackson Preemptive Schedule (see Section 5) is calculated for each machine. The maximal makespan of these schedules gives a lower bound for the search tree node s.

Note that the value of the lower bound $LB(s)$ may increase when fixing additional disjunctions by the procedure described in Section 5. Thus it is advantageous to check $LB(s)$ each time when additional disjunctive arcs are fixed.

The calculation of all these lower bounds is advantageous because every time a lower bound exceeds the upper bound a time consuming part of the algorithm (e.g. the computation of heads and tails or the fixation of disjunctions) becomes useless.

7. Calculation of heuristic solutions

The branching scheme we use is based on a heuristic solution of the problem. We implemented several heuristics and compared the results [11].

When experimenting with the 10×10-problem a heuristic based on a priority dispatching rule [3] gave the best results. The heuristic calculates the solutions step by step in the following way:

- Calculate the set C of all operations which can be scheduled next, i.e. C is the set of operations c with the property that all predecessors of c are already scheduled. Initially C contains the source 0 of the disjunctive graph.
- Let $u \in C$ be the operation with minimal value $r_u + p_u$, i.e. $r_u + p_u = \min_{c \in C}\{r_c + p_c\}$. Let M_k be the machine which has to process operation u. We define the set $\bar{C}$ by

$$\bar{C} := \{c \in C;\ r_c < r_u + p_u;\ c \text{ has to be processed on } M_k\}.$$

- For each operation $c \in \bar{C}$ we calculate a lower bound for the makespan of the schedule if we schedule c next. We choose the operation $\bar{c} \in \bar{C}$ with minimal lower bound.
- The set C is updated by inspecting all successors c of $\bar{c}$. If all predecessors of c have already been scheduled, we set $C = C \cup \{c\}$.

After this $\bar{c}$ is deleted from C, and we start again.

We tested different methods to calculate lower bounds for the operations $c \in \bar{C}$. The bound which gave the best results was calculated as follows.

Let T be the set of operations on machine M_k which are not scheduled yet (note that $\bar{C}$ is a subset of T). Take as lower bound the solution value of the Jacksons Preemptive Schedule (JSP) for the set T assuming that c has to be scheduled first.

8. Implementation and computational results

We implemented the branch and bound method in C on a Sun 4/20 Workstation. In this section we discuss the data structures used in this implementation which had some important influence on the speed. We will also present computational results.

Our branch and bound method is based on depth-first search. Although this method can recursively be implemented, we prefer to use an iterative algorithm which uses a stack of search tree nodes. As a result we got a much faster code.

We have to distinguish two different types of data:

- local data in a search tree node, e.g. the lower bound,
- global data for all search tree nodes, e.g. the conjunctive arcs.

First we will describe the local data.

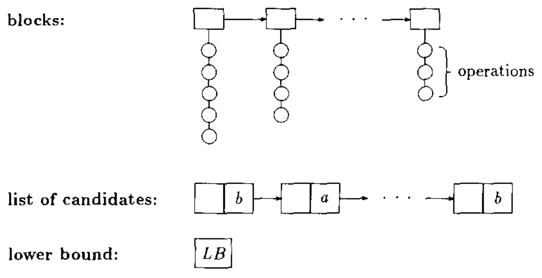

Fig. 6.

For each node r of the search tree we used the following information:

- blocks on a critical path corresponding with a heuristic solution calculated in r,
- before- and after-candidates;
- a lower bound.

In Fig. 6 we explain the corresponding data structures. The blocks are linked according to non-increasing cardinality, and for each block we store its position on the critical path. The candidates are stored according to non-increasing head-values of before-candidates (b) and tail-values of after-candidates (a).

In addition we have to store all disjunctive arcs fixed in a search tree node. Due to the fact that we used depth first search in our algorithm only the search tree nodes on the path from the root to the actual node have to be stored simultaneously. Furthermore, the set FD_r of fixed disjunctive arcs in node r is a subset of FD_s if s is a successor of r. Thus, we can store the fixed disjunctive arcs in the following way.

We use two *global* arrays of linked lists, in which the sets of disjunctive predecessors and successors are stored for each operation. New elements are always appended to the end of these lists. This implies that we only have to store the number of disjunctive predecessors and successors of all operations in each search tree node. Using these numbers it is easy to delete the disjunctive arcs during the backtracking step correctly. Fig. 7 shows an example for the storage of disjunctive predecessors for the search tree node r and its successor s.

The global data for all search tree nodes are the conjunctive arcs and the actual upper bound. The conjunctive arcs are stored in the same way as the disjunctive arcs.

All other data used in the algorithm (e.g. heads, tails) can be computed using these local and global data.

The computational results are given in Tables 1 and 2. In Table 1 problems 1, 2 and 3 are the well-known 6×6, 10×10 and 5×20 benchmark problems of Muth and Thompson [14]. Problems 4 and 5 are the 10×10 problems given by Adams et al. [1] (see problem 5 and 6 of Table 1).

The problems in Table 2 are also given by Adams et al. Their results of these problems can be found in Table 2 of the corresponding paper.

Our tables contain the following information:

UB opt: Makespan of the optimal schedule. If this value is marked with $*$, we could not prove its optimality within a time of 3–5 days and give the makespan of the best solution found so far.

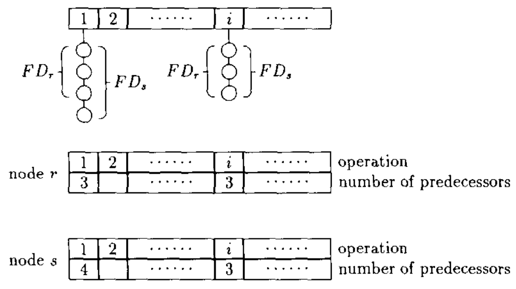

Fig. 7.

Table 1

Problem	*UB* opt	*UB* init	*LB*	nodes	CPU sec	*UB* heur	*UB* ABZ
1	55	55	52	1	0	55	55
2	930	1090	808	4242	1138	938	930
3	*1179	1454	1164	73	39	1179	1178
4	1234	1379	1028	2146	508	1234	1239
5	943	1052	835	135	31	943	943

UB init: The makespan of the heuristic solution calculated in the root of the search tree.

LB: A lower bound for the optimal makespan.

nodes: The number of search tree nodes the branch and bound method calculated to find the optimal solution and to prove its optimality. If we could not prove the optimality of *UB* opt we give the number of search tree nodes to reach the best found solution.

CPU sec: The time in CPU seconds used by our branch and bound method. If we could not prove the optimality of *UB* opt we give the time used to reach the best found solution.

UB heur: The value of the best solution found by our algorithm within the time Adams et al. used to find their best solution applying the SBII-algorithm.

UB ABZ: The makespan of the best solution by Adams et al. using the SBII-algorithm.

The branch and bound algorithm gave very good results for problems with 5 machines and 10, 15 or 20 jobs and for problems with 10 machines and 10 jobs. Not only we were able to improve the solution of Adams et al. within the time they needed, we were also able to prove optimality of the best found solution very fast. For problems with 5 machines, the algorithm terminates within a maximal time of 4 sec. The hardest problem among these ones was the 5×20 problem given by Muth and Thompson. We were not able to get a solution with a makespan less than 1179 within 3 days.

For the 10×10 benchmark problem by Muth and Thompson we were able to get the optimal solution and to prove its optimality within 19 min using 4242 search tree

Table 2

Problem	*UB* opt	*UB* init	*LB*	nodes	CPU sec	*UB* heur	*UB* ABZ
5 *machines*, 10 *jobs*							
1	666	671	666	4	0	666	666
2	655	835	655	34	3	655	669
3	597	696	588	12	1	597	605
4	590	696	567	40	4	590	593
5	593	593	593	1	0	593	593
5 *machines*, 15 *jobs*							
6	926	926	926	1	0	926	926
7	890	960	890	1	0	890	890
8	863	893	863	2	0	863	863
9	951	951	951	1	0	951	951
10	958	958	958	1	0	958	959
5 *machines*, 20 *jobs*							
11	1222	1222	1222	1	0	1222	1222
12	1039	1050	1039	2	1	1039	1039
13	1150	1189	1150	1	0	1150	1150
14	1292	1292	1292	1	0	1292	1292
15	1207	1363	1207	21	4	1207	1207
10 *machines*, 10 *jobs*							
16	945	1074	875	252	58	945	978
17	784	849	739	63	15	784	787
18	848	926	770	271	64	848	859
19	842	977	709	1456	340	842	860
20	902	987	807	1381	343	902	914
10 *machines*, 15 *jobs*							
21	*1059	1175	995	626 723	414 353	1124	1084
22	927	1060	913	10 524	6 700	949	944
23	1032	1155	1032	6 616	3 451	1071	1032
24	935	1085	881	136 512	89 062	1011	976
25	977	1086	894	428 833	273 162	996	1017
10 *machines*, 20 *jobs*							
26	1218	1342	1218	56 564	43 800	1231	1224
27	*1270	1413	1235	185 039	211 266	1350	1291
28	*1276	1468	1216	20 030	14 272	1279	1250
29	*1202	1352	1114	325 665	392 989	1242	1239
30	1355	1577	1355	368	239	1355	1355
10 *machines*, 30 *jobs*							
31	1784	1903	1784	8	7	1784	1784
32	1850	1850	1850	1	1	1850	1850
33	1719	1766	1719	77	75	1737	1719
34	1721	1825	1721	15	12	1721	1721
35	1888	2028	1888	24	23	1888	1888
15 *machines*, 15 *jobs*							
36	1268	1406	1224	129 706	113 419	1364	1305
37	*1425	1588	1355	550 980	396 484	1477	1423
38	*1232	1371	1077	99 546	89 229	1295	1255
39	1233	1442	1221	104 739	94 739	1385	1273
40	*1238	1417	1170	69 551	64 336	1283	1269

nodes. This result is improved if we start the branch and bound algorithm with better upper bounds: for $UB = 951$, it needs only 14 min of CPU time and 3598 search tree nodes for reaching the optimal solution and proving its optimality. If we start with $UB = 930$, the algorithm does need only 5 min and 1288 search tree nodes for the proof that no better solution exists.

We also got very good results for the problems with 10 machines and 30 jobs given in Table 2. We calculated the optimal solution and proved its optimality with 1 up to 77 sec with a maximal number of 75 search tree nodes.

For problems of size 10×15, 10×20 and 15×15 we improved the solution given by Adams et al. in 11 of 15 cases. We proved optimality for 8 test problems.

Nevertheless, there are 2 of these problems for which we did not reach the upper bound given by Adams et al., and for others we could not prove optimality of the best found solution. Also the improvement of the upper bound given by Adams et al. were often done at the cost of high computational time.

Currently there exist two other fast branch and bound implementations [2, 8].

In comparison with the algorithm of Carlier and Pinson our algorithm gives very good results for small instances of the problem (up to 10×10). For problems with large ratio between the number of jobs and the number of machines (5×10, 5×20, 10×30) our algorithm is very much faster than the one of Carlier and Pinson (e.g. Table 2, Problems 6, 31, 32). For large instances of the problem (10×20, 15×15) our algorithm gives poorer results.

In comparison with the results for the 10×10 problems reported by Applegate and Cook our algorithm also gives very good results. In most cases their algorithm needs much more time than ours does (e.g. Table 2, Problems 19, 20). The algorithm of Applegate and Cook, improves the best found solution of our algorithm for some hard problems (e.g. Table 2, Problems 27, 38, 40). Unfortunately they do not report the computation times which were necessary for reaching these bounds.

9. Concluding remarks

We have presented a branch and bound algorithm for solving the job-shop scheduling problem which solves the famous 10×10 benchmark problem in less than 19 min (including optimality proof) on a workstation. For other benchmark problems new optimal solutions are given or the best known solutions are improved considerably. The algorithm may also be used as a heuristic by stopping it after a fixed amount of time. For problems up to size 10×10 this heuristic outperforms the heuristic of Adams et al.

There is still room for improvement. To solve problems of size larger than 10×10 one could apply the heuristic of Adams et al. for providing a good initial solution in the root of the enumeration tree. Also in the vertices of the enumeration tree which are close to the root we may increase the effort of getting better lower bounds. For vertices deeper in the enumeration tree it seems to be too time consuming to apply the heuristic of Adams et al. Heuristics which are faster but provide better results than our priority driven heuristics should be applied in these nodes. Finally more sophisticated methods for fixing additional disjunctions should be developed.

Acknowledgement

The authors would like to thank Prof. J. Grabowski, Prof. J. Carlier, and Dr. E. Pinson for fruitful discussions concerning the branch and bound algorithm. They are also very grateful for the constructive comments of the referees.

References

[1] J. Adams, E. Balas and D. Zawack, The shifting bottleneck procedure for job-shop scheduling, Management Sci. 34 (1988) 391–401.
[2] D. Applegate and W. Cook, A computational study of job shop scheduling, CMU-CS-90-145, ORSA J. Comput. 3 (1991) 149–156.
[3] R.W. Bouma, Job-shop scheduling: A comparison of three enumeration schemes in a branch and bound approach, Master's thesis, Erasmus University Roterdam, Faculty of Econometrics and Operations Research (1982).
[4] P. Brucker and B. Jurisch, A new lower bound for the job-shop scheduling problem, European J. Oper. Res. 64 (1993) 156–167.
[5] P. Brucker, B. Jurisch and B. Sievers, Job-shop (C-codes), European J. Oper. Res. 57 (1992) 132–133.
[6] J. Carlier, One machine problem, European J. Oper. Res. 11 (1982) 42–47.
[7] J. Carlier and E. Pinson, An algorithm for solving the job-shop problem, Management Sci. 35 (1989) 164–176.
[8] J. Carlier and E. Pinson, A practical use of Jackson's preemptive schedule for solving the job shop problem, Ann. Oper. Res. 26 (1990) 269–287.
[9] J. Grabowski, E. Nowicki and C. Smutnicki, Algorytm blokowy szeregowania operacji w systemie gniazdowyn, Przeglad Statystyczny R. XXXXV, zeszyt 1 (1988) 67–80.
[10] J. Grabowski, E. Nowicki and S. Zdrzalka, A block approach for single machine scheduling with relase dates and due dates, European J. Oper. Res. 26 (1986) 278–285.
[11] B. Jurisch and B. Sievers, Ein Branch and Bound Verfahren für des Job-Shop Scheduling Problem, Osnabrück. Schrift. Math. Reihe P, Heft 122.
[12] P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job shop scheduling by simulated annealing, Oper. Res. 40 (1992) 113–125.
[13] J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, in: Annals of Discrete Mathematics 1 (North-Holland, Amsterdam, 1977) 343–362.
[14] J.F. Muth and G.L. Thompson, Industrial Scheduling (Prentice-Hall, Englewood Cliffs, NJ, 1963).
[15] B. Roy and B. Sussmann, Les problèmes d'ordonnancement avec contraintes disjonctives, Note DS no. 9 bis, SEMA, Paris.
[16] M. Widmer, Job-shop scheduling with tooling constraints: a tabu search approach, OR working paper 89/22, Dèpartement of Mathèmatiques, Ecole Polytechnique Fédérale de Lausanne.
[17] G. McMahon and M. Florian, On scheduling with ready times and due dates to minimize maximum lateness, Oper. Res. 23 (1975) 475–482.