OR Spektrum (1994) 15:205–215

OR Spektrum

© Springer-Verlag 1994

Tabu search for the job-shop scheduling problem with multi-purpose machines*

Johann Hurink[1], **Bernd Jurisch**[2], **and Monika Thole**[1]

[1] Fachbereich Mathematik/Informatik, Universität Osnabrück, Postfach 4469, D-49069 Osnabrück, Germany
[2] Faculty of Business Administration, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1B 3X5

Received 19 May 1993 / Accepted 16 November 1993

Abstract. In this paper we study the following generalization of the job-shop scheduling problem. Each operation can be performed by one machine out of a set of machines given for this operation. The processing time does not depend on the machine which has been chosen for processing the operation. This problem arises in the area of flexible manufacturing. As a generalization of the job-shop problem it belongs to the hardest problems in combinatorial optimization. We show that an application of tabu search techniques to this problem yields excellent results for benchmark problems.

Zusammenfassung. In dieser Arbeit behandeln wir die folgende Verallgemeinerung des Job-Shop Scheduling Problems. Jede Operation kann auf einer beliebigen Maschine aus einer Menge von Maschinen, die für diese Operation gegeben ist, bearbeitet werden. Die Bearbeitungszeit hängt dabei nicht von der gewählten Maschine ab. Das in dieser Arbeit behandelte Problem tritt im Bereich der flexiblen Fertigung auf. Als Verallgemeinerung des klassischen Job-Shop Problems gehört es zu den schwierigsten Problemen aus dem Bereich der kombinatorischen Optimierung. Wir zeigen, daß eine Anwendung der Tabu-Search Metaheuristik hervorragende Ergebnisse für die von uns untersuchten Testprobleme liefert.

Key words: Job-shop scheduling, tabu search, flexible manufacturing

Schlüsselwörter: Job-Shop Scheduling, Tabu Suche, Flexible Fertigung

* Supported by Deutsche Forschungsgemeinschaft, Project JoPTAG

Correspondence to: J. Hurink

1. Introduction

In this paper we study a problem which arises in the area of flexible manufacturing systems. Here we have a small number of so-called multi-purpose machines which can be equipped with different tools. Moreover, there is a set of jobs which have to be processed on the machines. A machine can process a job only if it is equipped with the tool the job needs for processing.

We consider the situation that the multi-purpose machines in the system are already equipped with tools. This yields the following problem which is called job-shop scheduling problem with multi-purpose machines (MPM job-shop problem).

We have a set of jobs, each one consisting of a number of operations which have to be processed in a given order. Moreover there is a set of multi-purpose machines which are equipped with different tools. Associated with each operation there is a set of machines which due to their tool equipment can process this operation. The processing of an operation takes a given amount of time. We have to find an assignment of the operations to the machines and a schedule for the operations on the machines such that a given objective function is minimized.

The MPM job-shop problem is a generalization of the classical job-shop problem which belongs to the hardest problems in combinatorial optimization. A problem with 10 jobs and 10 machines which has already been formulated in 1963 (Fisher and Thompson (1963)) has been solved only 26 year later (Carlier and Pinson (1989)).

In this paper we will present heuristic solution methods for the job-shop problem with multi-purpose machines. It is organized as follows.

In Sect. 2 we will give a description of the MPM job-shop problem and an overview on previous research. It turns out that some MPM job-shop problems are NP-hard even if their classical counterparts are solvable in polynomial time.

In Sect. 3 we will present methods for the calculation of heuristic solutions based on neighborhoods. Initial solu-

tions are calculated using a fast heuristic based on insertion techniques.

We have implemented the developed heuristics on a Sun Workstation. Computational results are presented in Sect. 4. We conclude this paper by providing final remarks in Sect. 5.

2. Preliminaries

In this section we will give a formal definition of the job-shop scheduling problem with multi-purpose machines. Moreover, we will review the previous research on related problems and give some complexity results for job-shop scheduling problems with multi-purpose machines.

2.1. Formulation of the problem

The job-shop scheduling problems with multi-purpose machines (MPM job-shop problem) may be formulated as follwos. There are *n jobs* $J_1, \ldots, J_n$, each job J_i consisting of n_i *operations* $O_{i1}, \ldots, O_{in_i}$ which have to be processed in this order. Moreover, there are m different, so-called *multi-purpose machines* $M_1, \ldots, M_m$ which are equipped with different tools. The operation $O_{ij} (i = 1, \ldots, n; j = 1, \ldots, n_i)$ has to be processed by one specific tool for p_{ij} time units, i.e. it can be processed by each machine which is equipped with this tool. Thus, associated with each operation O_{ij} there is a non-empty set $\mathcal{M}_{ij} \subseteq \{M_1, \ldots, M_m\}$: O_{ij} has to be processed by one machine of $\mathcal{M}_{ij}$. Preemption is not allowed. Moreover, the following restrictions have to be fulfilled:

- no machine can process more than one operation at the same time and
- no job can be processed by more than one machine at the same time.

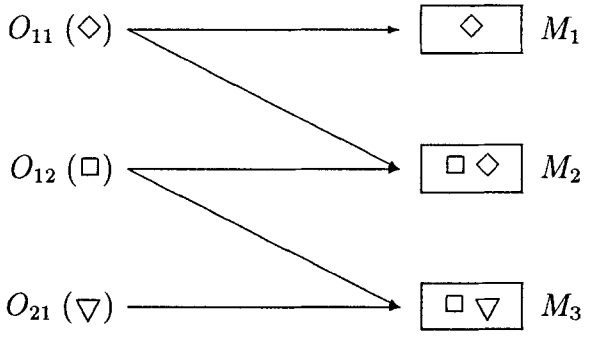


Fig. 1

Figure 1 shows an example of a job-shop problem with multi-purpose machines with two jobs, three operations, and three machines. The operations O_{11}, O_{12}, and O_{21} have to be processed by the ◇-tool, by the □-tool, and by the ▽-tool, respectively. Thus, O_{11} may be processed by M_1 or M_2, O_{12} may be processed by M_2 or M_3, and O_{21} has to be processed by M_3.

An *assignment* μ of operations to machines is feasible if $\mu(O_{ij}) \in \mathcal{M}_{ij}$ for $i = 1, \ldots, n; j = 1, \ldots, n_i$. For a given assignment μ a *μ-schedule* is defined by the completion times C_{ij} of all operations O_{ij}. Such a μ-schedule is feasible if the schedule of the job-shop problem corresponding to μ is feasible, i.e. if it fulfills the restrictions given above.

We are interested in finding a feasible assignment μ^* and a feasible μ^*-schedule C^* such that the total schedule lenght $C_{\max}$ is minimized.

Following the $\alpha|\beta|\gamma$-notation suggested by Graham, Lawler, Lenstra, Rinnooy Kan (1979) we denote the job-shop problem with multi-purpose machines by $J(\text{MPM})| \, |C_{\max}$.

2.2. Previous research

Most research has been done on the classical job shop scheduling problem which is a special case of the MPM job-schop problem; we have $|\mathcal{M}_{ij}| = 1$ for all operations O_{ij}. For this problem both exact methods (branch and bound methods, see Applegate and Cook (1991); Carlier and Pinson (1990); Brucker et al. (1992), and heuristic methods (e.g. based on priority dispatching rules, insertion techniques, or neighborhoods) are known. Recently Dell'Amico and Trubian (1993) presented excellent results obtained by applying tabu search to the job-shop problem.

There are other special cases of the MPM job-shop problem studied in the literature. Graham (1966) considered flow-shop problems with parallel machines which are generalizations of the classical flow-shop problem. He assumed that there is a set $\mathcal{M}_j$ of identical machines for processing the operations $O_{ij} (i = 1, \ldots, n; j = 1, \ldots, m)$ with $|\mathcal{M}_j| = |\mathcal{M}_k|$ for $1 \leq j, k \leq m$. Salvador (1973) generalized this problem to so-called "flexible flow-shops". Again there is a set $\mathcal{M}_j$ of parallel identical machines to process the operations $O_{ij} (i = 1, \ldots, n; j = 1, \ldots, m)$, but now we may have $|\mathcal{M}_j| \neq |\mathcal{M}_k|$ for $j \neq k$. Salvador proposed a branch and bound algorithm for solving this problem exactly.

The MPM job-shop problem in the form considered in this paper has been studied first by Brucker and Schlie (1990).They gave a polynomial time algorithm for the problem of minimizing the makespan when the number of jobs is equal to 2, i.e. for $J(\text{MPM})|n = 2|C_{\max}$. The corresponding problem with 3 jobs is NP-hard even if the number of machines is restricted to 2 (Jurisch (1992)). Nevertheless, Meyer (1992) proved that the problem $J(\text{MPM})|n = r|C_{\max}$ is solvable in pseudo-polynomial time for each fixed number r of jobs. If the processing times of the operations are restricted to be equal to 1, the resulting problem $J(\text{MPM})|n = r, p_{ij} = 1|C_{\max}$ becomes polynomially solvable again (Mayer (1992)).

Brandimarte (1993) considered so-called "flexible job-shop" which slightly generalize our problem. In flexible job-shops the processing times of operations also depend on the machine which is chosen for processing. Moreover, he briefly discussed the problem of minimizing the weighted number of tardy jobs.

The main contribution of the paper of Brandimarte is a tabu search algorithm for flexible job-shops. Our approach is different in two aspects:

- Our neighborhood is more sophisticated: Brandimarte only considers exchanging successive operations on the machines.
- Brandimarte uses a more "hierarchical" approach: He solves the problem of assigning the operations to the machines and then focuses on the resulting job-shop problem for some time. A reassignment is done after a predefined number of steps. We consider a reassignment of operations in each step of the tabu search algorithm.

In the next section we will introduce the so-called disjunctive graph model which will be helpful for the presentation of the heuristic algorithms in Sect. 3.

2.3. The disjunctive graph model

We have already observed that if for the job-shop problem with multi-purpose machines an assignment μ of the operations to the machines is given, the problem of calculating an optimal μ-schedule is a classical job-shop problem. Thus, a feasible solution of the MPM job-shop problem can be described by a feasible assignment μ and a feasible schedule of the job-shop problem corresponding to μ.

We will use the disjunctive graph model (Roy and Sussmann (1964)) to describe the solution of a job-shop problem. For a given instance of the MPM job-shop problem and a corresponding feasible assignment μ we define a disjunctive graph $G=(V, C, D)$ as follows.

V is the set of nodes, representing the operations of all jobs. In addition there are two special nodes, a source 0 and a sink $*$. There is a weight associated with each node. The weights of 0 and $*$ are zero while the weights of the other nodes are the processing times of the corresponding operations.

C is the set of directed *conjunctive* arcs. These arcs reflect the job-order of the operations. Additionally there are conjunctive arcs between the source and the first operations of all jobs and between the last operation of all jobs and the sink. More precisely, we have

$$\begin{aligned} C = & \{O_{ij} \to O_{i,j+1} : i = 1,\ldots,n; j = 1,\ldots,n_i - 1\} \\ & \cup \{0 \to O_{i1} : i = 1,\ldots,n\} \\ & \cup \{O_{in_i} \to * : i = 1,\ldots,n\} \end{aligned}$$

D is the set of undirected *disjunctive* arcs. Such an arc exists for each pair of operations which are assigned to the same machine.

The basic scheduling decision is to define an ordering between all those operations which are assigned to the same machine. This can be done by turning *undirected* disjunctive arcs into *directed* ones. A set S of directed (so-called *fixed*) disjunctive arcs is called a selection. If a selection defines a feasible schedule it is called a *complete selection*. A selection is complete if

- each disjunctive arc has been fixed, i.e. there is a fixed relation between each pair of operations that are assigned to the same machine and
- the resulting directed graph $G(S)=(V, C \cup S)$ is acyclic.

It is easy to see that the finish time of a schedule corresponding to a complete selection S is equal to the length of the longest weighted (so-called *critical*) path from 0 to $*$ in $G(S)=(V, C \cup S)$.

3. Tabu search for the MPM job-shop problem

In this section we will present a tabu-search heuristic for the job-shop problem with multi-purpose machines. In Sect. 3.1 we will present both the basic ideas of local search heuristics and two neighborhoods for the problem. The initial solution for the tabu search algorithm is calculated using a fast heuristic based on insertion techniques and beam search. This algorithm is presented in Sect. 3.2.

3.1. Heuristics based on local search techniques

We will present the basic ideas of heuristics based on local search techniques in Sect. 3.1.1. In connection with these methods it is necessary to define a neighborhood on the set of all feasible solutions or – more generally – on a subset of the set of all feasible solutions which contains the optimal solution of the given problem. In Sect. 3.1.2 we will present two different neighborhoods for the MPM job-shop problem.

3.1.1. Basic ideas of local search heuristics. We will describe heuristics based on local search techniques for solving an arbitrary discrete optimization problem (Hurink (1992)). A discrete optimization problem may be described as follows.

For a given finite set S of feasible solutions and a cost funktion $f: S \to \mathbb{R}$ we have to find a solution $s^* \in S$ with

$$f(s^*) \leq f(s) \quad \text{for all } s \in S.$$

Heuristics based on local search techniques start from some solution $s \in S$ and search iteratively through the set S until some stop condition is fulfilled. When searching through the set S in some systematic way it makes no sense to allow moves from one feasible solution s to any other feasible solution s'. This would result in a random search procedure or a complete enumeration. The set of solutions which are reachable from s – these solutions are called *neighbors* of s – has to be restricted in some way. This is done as follows.

For each solution $s \in S$ we define a set $N(s) \subseteq S$ of neigbors of s. It is possible to move from s to another solution s' if and only if $s' \in N(s)$. The complete *neighborhood* N is defined by the set of neighbors $N(s)$ for each $s \in S$.

In general the set S contains an exponential number of solutions. For this reason it is not possible to store the whole neighborhood. The best way to overcome this

difficulty is to give a rule which for any feasible solution s describes the set $N(s)$ of neighbors. This rule is given by a set of *allowed modifications*. An allowed modification is a mapping $F: S \to S$.

In general an heuristic algorithm based on local search techniques may be formulated as follows.

Algorithm local search

0. Calculate an initial solution $s \in S$;
   REPEAT
1. Calculate some solution $s' \in N(s)$;
2. If s' is accepted THEN
   $s := s'$;
3. UNTIL some stop condition is fulfilled;

Different types of local search heuristics differ by

- the method which is used for calculating a solution $s' \in N(s)$ (Step 1.);
- the criteria for accepting a solutions $s' \in N(s)$ (Step 2.);
- the stop condition which is used (Step 3.).

The simplest heuristic based on local search techniques is the *iterative improvement* appraoch (e.g., see Papadimitriou and Steiglitz (1982)). From all solutions $\bar{s} \in N(s)$ we choose the best one (in terms of the objective function) as starting solution for the next iteration. The procedure continues until no solution $\bar{s} \in N(s)$ with $f(\bar{s}) < f(s)$ is found.

One of the main problems with the iterative improvement algorithm is the following. Because only solutions s' which improve the current solutions s are accepted it is not possible to leave a local optimum. The objective value of such a local optimum may be much greater than the objective value of the optimal solution.

To overcome this problem also solutions which do not improve the current solution have to be accepted. However, this implies that solutions can be inspected more than once and therefore the method might get stuck in a cycle.

One method to avoid these problems would be to store all solutions $s \in S$ which have already been visited in a so-called *tabu list* T. A neighbor s' of the current solution s is only accepted as starting solution for the next iteration if it is not contained in the tabu list T. Strategies of this type are usually called *tabu search* methods (see, e.g., Glover (1989), (1990)).

Due to capacity restrictions it is not possible to store all the solutions which have already been visited. Therefore the tabu list will contain only the t solutions which have been inspected last.If t is large enough the possibility of cyclin becomes small, but it may still occur. Thus, some stop-criteria have to be used to guarantee the termination of the algorithm. Furthermore we will not store whole solutions in the tabu list but only typical properties of a solution which guarantee that a visited solution becomes tabu, i.e. it will not be reached again.

We illustrate this proceeding in an example. Assume that we try to solve a one-machine problem with n jobs J_1, $J_2, \ldots, J_n$ using a tabu search approach. One solution y may be given by the ordering $J_1 \to J_2 \to \ldots \to J_n$. Now assume that we generate a neighbor of y by moving the operation J_i to the first position. Instead of storing the whole solution y in the tabu list, we only store the part $J_{i-1} \to J_i \to J_{i+1}$. All modifications which yield a solution containing this partial order are forbidden. Thus, it is not possible to return to solution y as long as $J_{i-1} \to J_i \to J_{i+1}$ is contained in the tabu list.

This example also shows a disadvantage of such a proceeding. Not only the solution y is forbidden, but all solutions containing the partial order $J_{i-1} \to J_i \to J_{i+1}$. Thus, solutions may be forbidden even if they have not been inspected yet. To overcome this problem heuristics based on tabu-search techniques use so-called *aspiration-criteria* which allow to accept neighbors even if they are forbidden due to the tabu list; i.e. the aspiration-criteria cancel the tabu status of a solution. For example, solutions which improve the best solution found so far should always be accepted. For details we refer to Dell'Amico and Trubian (1993).

Algorithm tabu search

$T := \emptyset$;
0. Calculate an initial solution $s \in S$;
   REPEAT
   IF all modifications lead to solutions which are tabu THEN
   STOP;
1. Choose an allowed modification F which does not lead to a tabu solution;
   Calculate the resulting solution $s' := F(s)$;
2. $s := s'$;
   update the tabu list;
3. UNTIL some stop-condition is fulfilled;

The stop-conditions in Step 3. of the algorithm may depend on the number of iterations, the time which has passed without improving the best solution found so far, etc. Furthermore a simple and efficient strategy for choosing the allowed modification in Step 1 is to choose the modification F which gives the best solution s' in the set of all solutions which can be generated by allowed modifications.

The quality of an heuristic based on local search techniques strongly depends on the neighborhood N which is used. In the following section we will give two neighborhoods for the MPM job-shop problem which yield quite efficient local search heuristics.

3.1.2. Neighborhoods for the MPM job-shop problem. In this section we will give two neighborhoods for the MPM job-shop problem. Both neighborhoods are based on a theorem which describes how a given solution of a MPM job-shop problem may be improved. To describe this theorem we need the notation of a *block* which has been introduced in connection with one-machine problems, permutation flow-shop problems, etc. (e.g., see Grabowski et al. (1986)).

Let μ be a feasible assignment and S a complete selection of the job-shop problem corresponding to μ. Furthermore let P be a critical path in $G(S)$. A sequence of successive nodes in P is called a *block* on P in $G(S)$ if the following properties are satisfied:

- The sequence contains at least two nodes.
- All operations represented by the nodes in the sequence are assigned to the same machine.
- Enlarging the sequence by one operation yields a sequence which does not fulfill the second property.

Note that the selection S defines an optimal solution if one critical path P in $G(S)$ does not contain any block at all. In this case, any pair of successive operations on P is processed on different machines, i.e. they are connected by a conjunctive arc. Thus, all operations on P belong to the same job J_i, and the length of the critical path P is equal to the sum of processing times of all operations of J_i. Obviously, this value defines a lower bound for the makespan of an optimal schedule.

Based on the given notation we can prove the following theorem.

Theorem 1. *Let y and y' be two feasible solutions of a given MPM job-shop problem corresponding to the complete selection S and S', respectively. If y' improves y, then for any critical path P in $G(S)$ one of the following properties holds:*

- *in y' at least one operation of one block of P is processed on another machine than in y, or*
- *in y' at least one operation of one block B of P, different from the first operation in B is processed before all other operations of B, or*
- *in y' at least one operation of one block B of P, different from the last operation in B is processed after all other operations of B.*

Proof. Any critical path P in $G(S)$ has the following form:

$$P\colon\ 0, u_1^1, \ldots, u_{b_1}^1, u_1^2, \ldots, u_{b_2}^2, \ldots, u_1^k, \ldots, u_{b_k}^k, *.$$

Here $u_1^j, \ldots, u_{b_j}^j$ define a maximal number of operations which are processed on the same machine, i.e. $u_1^j, \ldots, u_{b_j}^j$ defines a block if $b_j \geq 2$. Now assume that in y' all operations of all blocks are processed on the same machine as in y, and that in y' no operation of any block B of P is processed before the first or after the last operation of B. Thus, we have:

(i) $u_{b_j}^j$ is processed before u_1^{j+1} in y' $(j=1,\ldots,k-1)$: because $u_{b_j}^j$ and u_1^{j+1} are processed on different machines in y, u_1^{j+1} is a conjunctive successor of $u_{b_j}^j$.
(ii) If $b_j \geq 2$, all operations $u_1^j, \ldots, u_{b_j}^j$ are processed on the same machine both in y and in y'.
(iii) If $b_j \geq 2$, u_1^j is processed before $u_2^j, \ldots, u_{b_j}^j$, and $u_{b_j}^j$ is processed after $u_1^j, \ldots, u_{b_j-1}^j$ both in y and in y'.

In y' the operations $u_1^j, \ldots, u_{b_j}^j$ are porcesed in an order $v_1^j, \ldots, v_{b_j}^j$, where $v_1^j, \ldots, v_{b_j}^j$ is a permutation of $u_1^j, \ldots, u_{b_j}^j (j=1,\ldots,k)$. Due to (iii) we have $v_1^j = u_1^j$ and $v_{b_j}^j = u_{b_j}^j$ for all $j=1\ldots,k$. Thus, $G(S')$ contains the path

$$P'\colon\ 0, v_1^1, \ldots, v_{b_1}^1, v_1^2, \ldots, v_{b_2}^2, \ldots, v_1^k, \ldots, v_{b_k}^k, *.$$

The length $L(S')$ of the longest (critical) path in $G(S')$ cannot be less than the lenght of P'. Thus, we have

$$\begin{aligned} L(S') &\geq \sum_{j=1}^{k} \sum_{l=1}^{b_j} p_{v_l^j} \\ &= \sum_{j=1}^{k} \sum_{l=1}^{b_j} p_{u_l^j} \\ &= L(S) \end{aligned}$$

where $L(S)$ denotes the length of the critical path in $G(S)$. This contradicts the assumption. □

Based on this theorem the first neighborhood is defined as follows.

Neighborhood N1

Consider a feasible solution y for a MPM job-shop problem. A feasible solution y' is a neighbor of y if it is constructed in the following way.

Let S denote the complete selection corresponding to y and let P be a critical path in $G(S)$. Then y' is derived from y by

- processing one operation of one block of P on another machine than in y or by
- moving one operation of one block B of P different from the first (the last) operation in B before (after) all other operations of B.

Because of theoretical and practical reasons a fundamental question is whether a given neighborhood has the *connectivity property* or not. This property is defined as follows.

A neighborhood N is called *connected* if it is possible to reach from any solution y to an optimal solution in a finite number of steps, i.e. if there exists a final sequence $\langle y=y_1, y_2, \ldots, y_k\rangle$ of solutions such that $y_{i+1} \in N(y_i)$ for all $i=1,\ldots,k-1$ and y_k is an optimal solution.

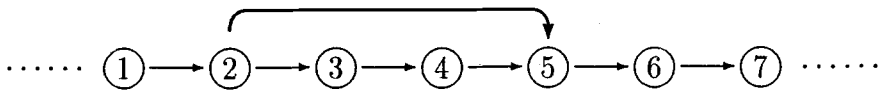
We strongly conjecture that neighborhood $N1$ has the connectivity property, but unfortunately a proof (or a counterexample) has not been found yet. Later we will prove that the second neighborhood called $N2$ which is based on an idea of Dell'Amico and Trubian (1993) is connected.

Neighborhood N2

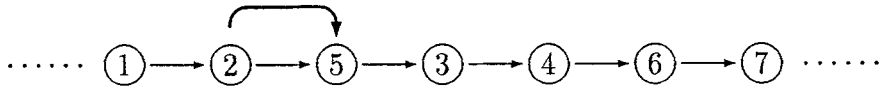
Consider a feasible solution y for a MPM job-shop problem. A feasible solution y' is a neighbor of y if it is constructed in the following way.

Let S denote the complete selection corresponding to y and let P be a critical path in $G(S)$. Then y' is derived from y by

- processing one operation of one block of P on another machine than in y or by
- moving one operation of one block B of P different from the first (the last) operation in B before (after) all other operations of B. If the move of one operation j of one block B of P before (after) all other operations of B yield an unfeasible solution, then y' may also be generated by



a



b

Fig. 2

moving j into the position inside the block B closes to the first (last) position of B such that the resulting schedule is feasible.

Note that we have $N1 \subseteq N2$, i.e. $N1(y) \subseteq N2(y)$ for all feasible schedules y. $N2$ is obtained from $N1$ by considering additional moves whenever a move of an operation before or after the corresponding block generates an unfeasible schedule due to other fixed disjunctions. In Fig. 2 such a situation is shown. The operations $1, 2, \ldots, 7$ may define a block on a critical path. Assume that moving operation 5 before 1 or 2 generates unfeasible schedules because there is a path from 2 to 5 different from the path $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ in the disjunctive graph (symbolized by the bent arc in Fig. 2a). Moving 5 after 2 and before 3 may generate a feasible schedule. Thus, a neighbor is generated by moving 5 between 2 and 3 (see Fig. 2b).

For proving the connectivity of the neighborhood $N2$ we need the following

Lemma 1. *Let y be a feasible solution of a MPM job-shop problem corresponding to a complete selection S. Let $i \rightarrow j$ be a disjunctive arc on a critical path in $G(S)$ (this implies that i and j are two operations which are processed successively on one machine in y). Then the inversion of the disjunctive arc $i \rightarrow j$ also yields a feasible schedule.*

Proof. Since the assignment of the operations to the machines does not change, the result follows immediately from a similar result for the job-shop problem given by van Laarhoven, et al. (1992).

Using Lemma 1 we can prove the following

Theorem 2. *The neighborhood $N2$ is connected.*

Proof. Consider an arbitrary feasible schedule y. If y is an optimal solution we are done. Otherwise consider an optimal schedule y^* and the corresponding complete selection S^*. For an arbitrary solution y' let $n_M(y')$ be the number of operations which are assigned to different machines in y' and y^*, and let $n_D(y')$ be the number of disjunctive arcs fixed in different directions in S' and S^* (S' denotes the complete selection corresponding to y'). We do not count disjunctive arcs which are fixed in one complete selection and which are missing in the other one due to the fact that the corresponding operations are fixed on other machines.

We show that it is possible to construct a finite number of feasible schedules $y_1, y_2, \ldots, y_k$ with the following properties:

(1) $y = y_1$, and $y_k = y^*$ or y_k is another optimal schedule.
(2) $y_{i+1} \in N2(y_i)$ $(i = 1, \ldots, k-1)$.
(3) Operations which are assigned to the "optimal" machine in y_i (i.e. to the machine which processes this operation in y^*) are still assigned to this machine in y_{i+1}. Thus, we have $n_M(y_{i+1}) \leq n_M(y_i)$ $(i = 1, \ldots, k-1)$.
(4) If $n_M(y_{i+1}) = n_M(y_i)$, then $n_D(y_{i+1}) < n_D(y_i)$ $(i = 1, \ldots, k-1)$, i.e. the following property holds. Assume that the number of operations which are assigned to the "optimal" machines does not increase. Then the number of disjunctive arcs directed into the "wrong" direction (in comparison with y^*) decreases.

Assume that the solutions $y_2, \ldots, y_i$ have been constructed in this way. If y_i is optimal we are done. Otherwise, let S_i be the complete selection corresponding to y_i. Due to Theorem 1 the following property holds for any critical path P in $G(S_i)$: One operation of one block on P is processed on another machine than in y^*, or some operation of one block B on P is processed before or after all other operations of the block B in y^*.

Now consider an arbitrary critical path P in $G(S_i)$. First assume that there exists one operation j of one block of P which is assigned to machine $M_k(M_l)$ in $y_i(y^*)$ where $k \neq l$. It is possible to remove j from M_k and to insert it at some position on M_l, obtaining a feasible schedule y_{i+1}. Because this is one of the moves in $N2$, we have $y_{i+1} \in N2(y_i)$ and $n_M(y_{i+1}) < n_M(y_i)$. Thus, this move fulfills properties (2), (3), and (4).

Now assume that all operations of all blocks on the critical path P are assigned to the same machines as in y^*. At least one operation i_j of one block $B = \langle i_1, i_2, \ldots, i_j, \ldots, i_l \rangle$ is processed before or after all other operations of B in y^*.

If i_j has to be processed before all other operations of B, then we move i_j to the first possible position in B such that the resulting schedule y_{i+1} is feasible. This is one of the moves in $N2$. Note that due to Lemma 1 it is always

possible to move i_j before i_{j-1} because this defines the inversion of one disjunctive arc on the critical path. If i_j is moved directly before $i_s (1 \leq s \leq j-1)$, then we have $n_D(y_i) - n_D(y_{i+1}) = j - s$, i.e. the number of arcs directed in "wrong" direction decreases.

If i_j has to be processed after all other operations of B we can argue in a similar way. Thus, we obtain $y_{i+1} \in N2(y_i)$, $n_M(y_{i+1}) = n_M(y_i)$, and $n_D(y_i) < n_D(y_{i+1})$: the properties (2), (3), and (4) given above are fulfilled.

We still have to prove that the number of steps that we need to reach the solution y^* (or another optimal solution) is finite. Consider the pair $(n_M(y_i), n_D(y_i))$. Due to the properties (3) and (4) $(n_M(y_i), n_D(y_i))$ is strongly lexicographically decreasing in i. Because we have $n_D(y_i) \leq \left(\sum_{j=1}^{n} n_j \right)^2$ (the latter value is an upper bound for the total number of disjunctive arcs), the number of steps which are necessary to reach y^* or another optimal solution from y is bounded by $n_M(y) \left(\sum_{j=1}^{n} n_j \right)^2$. □

Based on the two neighborhoods it is possible to apply tabu search to the MPM job-shop problem. It only remains to give a method for calculating an initial solution. In the next section we will present a fast heuristic algorithm which is based on insertion techniques. It is a generalization of a heuristic proposed by Werner and Winkler (1991) for the classical job-shop problem.

3.2. A heuristic algorithm based on insertion techniques

We start with a feasible partial schedule which only contains the operations of the longest job. In this partial schedule the operations of the longest job have to be assigned to machines. The choice of a machine to process an operation is done as follows.

For each machine we calculate the sum of the processing times of the operations which have to be processed by this machine, i.e. the value

$$P(M_k) = \sum_{\substack{O_{ij} \\ \mathcal{M}_{ij} = \{M_k\}}} p_{ij}.$$

We assign the first operation of the longest job, say O_l, to the machine M_k with minimal value $P(M_k)$. Then we update the value $P(M_k)$ by defining $P(M_k) := P(M_k) + p_l$. Next we assign the second operation to a machine, etc. Thus, after each step $P(M_k)$ is the sum of processing times of all operations which can be processed only on this machine or which already have been assigned to this machine.

After scheduling the operations of the longest job (i.e. after assigning these operations to machines) we successively insert the remaining operations into the feasible partial schedule in an order of non-increasing processing times. For deciding how an operation O_{ij} should be inserted into the feasible partial schedule we check all possible positions as follows:

- We assign operation O_{ij} to all machines $M_k \in \mathcal{M}_{ij}$.
- For each machine M_k we execute the following steps. Assume that l operations have already been assigned to machine M_k. We insert O_{ij} before the first operation on M_k, then after the first and before the second one, etc. Finally we insert O_{ij} after the last operation on M_k. Thus, we check $l+1$ positions for the insertion of O_{ij} on M_k. The cost of assigning an operation in a specific position is defined as the length of the longest path through this operation in the resulting disjunctive graph. If the resulting graph contains a cycle, the cost is defined as ∞.

After assigning the operation O_{ij} to all machines in $\mathcal{M}_{ij}$ and inserting it in all feasible positions we choose the assignment and position which gave the lowest costs.

We illustrate the insertion algorithm using the following example.

Example. $n = 3$, $m = 3$, prec $= \emptyset$

	p_{i1}	$\mathcal{M}_{i1}$	p_{i2}	$\mathcal{M}_{i2}$	p_{i3}	$\mathcal{M}_{i3}$
J_1	1	$\{M_1\}$	4	$\{M_2, M_3\}$	2	$\{M_3\}$
J_2	1	$\{M_2\}$	3	$\{M_1, M_3\}$	3	$\{M_1, M_3\}$
J_3	4	$\{M_1, M_3\}$	1	$\{M_2\}$	5	$\{M_2\}$

The initial feasible partial schedule only contains the operations of the longest job J_3. The sum of processing times of operations which have to be processed on M_1 is 1, and the sum of processing times of operations to be processed on M_3 is 2. Thus, we assign O_{31} to M_1.

Now we insert O_{12} into this partial schedule. Assigning O_{12} to M_3 yields the lowest cost. In the next step we insert O_{22} into the resulting schedule. Continuing in this manner we finally obtain a schedule with $C_{\max} = 11$ (see Fig. 3).

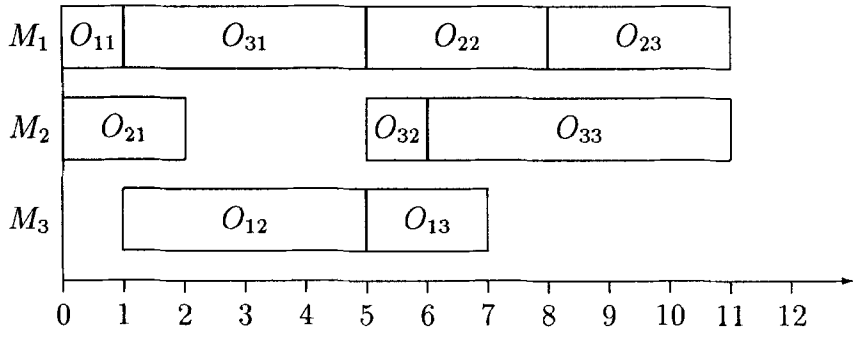


Fig. 3

For improving the quality of the heuristic solution we additionally use the so-called *beam search technique* (Ow and Morton (1989); Werner and Winkler (1991)). The main idea of this technique is to examine a fixed number k of feasible partial schedules in parallel. The insertion algorithm with beam search applied to the MPM job-shop problem works as follows.

Again we start with a partial schedule consisting only of the operations of the longest job. Then we assign the longest remaining operation to all feasible machines and schedule it in all possible positions. We select the k resulting feasible partial schedules which yield the lowest costs; if only $l<k$ feasible partial schedules exist, we consider them all. In the next step we insert the second-largest operation into all the selected partial schedules, i.e. we assign it to all feasible machines and schedule it in all feasible positions in all selected partial schedules. Again we select the k resulting feasible partial schedules with lowest costs, etc. In the last step, the best schedule which has been generated is taken as solution.

4. Computational results

We implemented the heuristics described in Sect. 3 in *C* on a Sun 4/20 Workstation. For obtaining test problems for the MPM job-shop problem we modified benchmark problems for the classical job-shop problem. In detail we considered the following problems (problems m06, m10, m20 are from Fisher and Thompson (1963), problems l01–l40 are from Adams et al. (1988)). m denotes the number of machines, n the number of jobs.

	m	n
m06	6	6
m10	10	10
m20	5	20
l01–l05	5	10
l06–l10	5	15
l11–l15	5	20
l16–l20	10	10
l21–l25	10	15
l26–l30	10	20
l31–l35	10	30
l36–l40	15	15

In each case the number of operations per jobs is equal to the number of machines, i.e. we have $n_i = m$ for $i = 1, \ldots, n$.

These benchmark problems for the classical job-shop problem are very special instances of MPM job-shop problems: we have $|\mathcal{M}_{ij}| = 1$ for all operations O_{ij}. For obtaining MPM test problems, we modified the job-shop benchmark problems in the following way.

Each operation can be processed by the machine which has to process it in the job-shop benchmark problem. For each operation O_{ij}, we consider all machines M_k one by one, enlarging the set $\mathcal{M}_{ij}$ by M_k with a given probability. By considering different probabilities, we obtain different sets *edata, rdata, vdata* of benchmark problems for the MPM job-shop problem.

The main properties of the benchmark problems are summarized in the following table. $|\mathcal{M}_{ij}|$ ave ($|\mathcal{M}_{ij}|$max) denotes the average (maximal) cardinality of the sets $\mathcal{M}_{ij}$.

	$\lvert\mathcal{M}_{ij}\rvert$ave	$\lvert\mathcal{M}_{ij}\rvert$max
edata:	1.15	2 ($m \leq 6$) 3 ($m \geq 10$)
rdata:	2	3
vdata:	$\frac{1}{2}m$	$\frac{4}{5}m$

We studied four different variants of tabu search. We considered both neighborhoods $N1$ and $N2$. For each neighborhood we tested two variants, the first one by limiting the maximal number of iterations by 1000, the other one by limiting this number by 5000. Additionally, all variants have the following properties:

- The length of the tabu list is equal to 30.
- Neighbors of the current solution are generated in an order of non-decreasing lower bounds. This proceeding allows to drop a great number of neighbors without considering them in detail.
- The tabu search algorithms terminate before the maximal number of iterations is reached if one of the following properties holds:
  - All neighbors of the current solutions are tabu.
  - The makespan of the best found solution is equal to a lower bound.
  - The methods gets stuck in a cycle. The algorithms are able to recognize cycles containing at most 100 solutions.

The choice of the parameters has been done after some preliminary computational tests. The value 1000 for the maximal number of iterations seems to be a good trade of between time and quality. If one would reduce this number from 1000 to 500 the computational times would also reduce in most cases by the factor $\frac{1}{2}$ and the quality of the solution would get worse only in some cases. However, since these deteriorations of the quality were quite large for some benchmark problems and since the computational times for 1000 iterations are acceptable this number has been chosen as basic number. In order to investigate the influence of an additional large amount of computational time on the quality of the solutions, we have chosen 5000 as second number for the maximal number of iterations.

For the length of the tabu list several values have been considered in the preliminary tests. In general there was no lenght which led in all cases to the best results. For the number 30 we got in average the best results. For details we refer to Thole (1993).

The complete results with a maximum of 1000 iterations are presented in Table 1. The table contains the following information:

- LB: The value of the best known lower bound for the problem given by Jurisch (1992). If this value is marked with an asterisk it is the makespan of the optimal solution.
- beam3: The value of the initial solution, calculated by using the insertion algorithm based on beam search with beamwidth $k = 3$.

Table 1

data	edata						rdata						vdata					
			$N1-1000$		$N2-1000$				$N1-1000$		$N2-1000$				$N1-1000$		$N2-1000$	
	LB	beam3	UB	CPU	UB	CPU	LB	beam3	UB	CPU	UB	CPU	LB	beam3	UB	CPU	UB	CPU
m06	*55	57	57	0.9	57	0.9	*47	50	47	5.6	47	29.7	*47	48	47	1.0	47	0.9
m10	*871	995	917	1:06.2	899	24.4	679	803	737	13.2	737	13.0	*655	655	655	8.6	655	8.6
m20	*1088	1210	1109	2:22.0	1135	3:41.3	1022	1072	1028	1:28.0	1028	9:31.2	*1022	1038	1023	11:39.6	1023	14:03.4
l01	*609	688	611	17.1	618	6.1	570	591	574	1:16.8	577	1:29.7	*570	595	573	2:19.6	575	2:22.7
l02	*655	667	655	39.6	656	39.2	529	580	535	1:19.7	535	1:36.8	*529	659	531	2:12.1	530	2:34.6
l03	*550	647	573	7.9	566	3.5	477	537	481	1:25.0	486	1:33.6	477	498	482	2:02.3	481	2:10.0
l04	*568	613	578	31.7	578	39.4	*502	550	509	1:23.4	506	1:33.7	*502	517	504	1:54.9	503	2:23.3
l05	*503	510	503	34.0	503	5.7	*457	487	460	1:41.9	458	1:56.6	457	486	464	1:49.3	461	2:00.1
l06	*833	900	833	1:18.3	833	1:47.8	799	831	801	3:16.2	803	4:28.7	*799	841	802	5:26.9	799	6:24.3
l07	*762	807	765	1:33.6	778	1:59.3	749	780	752	3:32.6	752	4:33.3	749	774	751	5:12.7	752	6:02.0
l08	*845	949	845	1:19.7	845	1:55.0	765	788	767	3:30.5	768	4:33.6	765	774	766	5:17.2	766	6:17.5
l09	*878	912	878	1:33.0	878	2:09.7	853	895	859	3:44.0	857	4:29.4	853	857	854	5:55.9	854	6:46.9
l10	*866	880	866	22.2	866	1:27.3	804	824	806	3:42.0	805	4:49.3	*804	486	805	5:57.1	805	4:03.0
l11	1087	1158	1106	2:42.4	1106	4:36.1	*1071	1093	1073	7:50.3	1073	9:50.0	*1071	1079	1073	11:03.1	1073	13:18.3
l12	*960	1039	960	2:26.6	960	4:31.1	936	961	937	8:25.3	937	9:51.0	*936	951	940	1:23.2	940	14:19.3
l13	*1053	1215	1053	1:50.5	1053	2:56.3	*1038	1046	1039	8:20.6	1039	11:24.4	*1038	1052	1040	11:00.6	1041	14:34.1
l14	*1123	1173	1151	3:13.2	1123	4:26.2	*1070	1086	1071	8:03.2	1071	10:32.5	1070	1091	1071	10:30.5	1080	4:41.0
l15	*1111	1217	1111	2:22.6	1121	3:15.9	1089	1126	1093	3:15.0	1093	8:43.3	1089	1096	1091	11:18.7	1091	14:14.3
l16	*892	961	924	36.5	961	12.5	*717	835	717	32.2	717	1:02.2	*717	717	717	8.8	717	8.8
l17	*707	774	757	12.9	757	13.0	*646	898	646	7.4	646	7.4	*646	646	646	8.8	646	8.8
l18	*842	864	864	7.4	864	7.3	*666	755	674	1:33.6	673	1:31.1	*663	663	663	8.8	663	8.8
l19	*796	854	850	8.2	813	11.4	647	777	725	1:39.9	709	1:41.5	*617	648	617	2:15.9	617	2:17.8
l20	*857	947	919	9.1	919	9.2	*756	808	756	11.4	756	8.5	*756	756	756	8.7	756	8.7
l21	895	1259	1066	2:22.8	1085	2:30.7	808	960	861	4:56.2	861	5:28.9	800	844	826	15:20.8	825	15:37.6
l22	832	1049	919	1:57.2	905	2:06.8	737	960	790	4:39.1	795	5:00.3	733	757	745	14:17.0	744	15:25.4
l23	950	1122	980	2:06.1	980	2:19.4	816	961	884	4:19.5	887	5:03.9	809	842	826	15:19.4	829	14:41.5
l24	881	1047	952	1:58.9	952	2:18.1	775	925	825	4:58.9	830	5:28.2	773	817	796	15:18.6	796	16:06.8
l25	894	1148	970	2:10.1	969	2:10.2	752	914	823	4:21.5	821	4:41.4	751	804	770	15:16.4	769	15:53.4
l26	1089	1268	1169	3:58.3	1149	4:14.1	1056	1148	1086	10:46.9	1087	12:59.2	1052	1073	1058	34:58.0	1058	36:16.7
l27	1181	1403	1230	3:39.8	1236	4:19.5	1085	1214	1109	10:03.1	1115	11:50.6	1084	1118	1088	34:50.6	1088	36:22.9
l28	1116	1335	1204	3:37.8	1197	4:14.9	1075	1165	1097	10:10.8	1090	11:20.2	1069	1109	1073	34:56.8	1073	33:06.6
l29	1058	1369	1210	3:47.6	1205	4:35.0	993	1082	1016	9:10.5	1017	10:32.5	993	1020	995	34:31.6	996	35:32.1
l30	1147	1436	1253	3:42.6	1286	4:32.1	1068	1221	1105	9:37.9	1108	10:23.3	1068	1078	1071	36:50.4	1070	38:11.5
l31	1523	1797	1596	10:34.0	1593	14:33.1	1520	1595	1532	29:27.0	1533	34:34.0	1520	1543	1521	1:41:12.7	1521	1:56:17.1
l32	1698	1835	1769	3:36.6	1757	6:00.4	1657	1768	1668	29:39.5	1668	39:19.1	1657	1662	1658	1:40:15.0	1659	1:54:42.2
l33	*1547	1749	1575	9:32.2	1575	15:43.5	1497	1575	1511	32:13.6	1507	40:15.2	1497	1509	1498	1:45:08.9	1499	1:55:03.1
l34	1592	1781	1627	8:28.1	1636	13:36.1	1535	1640	1542	30:00.3	1543	38:57.4	1535	1550	1536	1:55:42.1	1538	36:53.3
l35	*1736	1817	1736	5:34.6	1736	12:37.0	1549	1629	1559	29:18.0	1559	37:29.4	1549	1571	1553	1:54:15.4	1551	1:53:56.5
l36	1006	1355	1247	3:03.0	1235	3:08.3	1016	1214	1054	5:13.6	1071	5:22.1	*948	948	948	1:12.8	948	1:13.6
l37	1355	1621	1453	2:47.3	1456	3:05.2	989	1264	1122	5:59.4	1132	6:06.5	*986	993	986	1:21.6	986	1:21.0
l38	1019	1232	1185	2:45.8	1185	2:58.0	943	1134	1004	5:35.2	1001	5:37.9	*943	943	943	1:07.4	943	1:08.0
l39	1151	1390	1226	2:56.3	1226	3:06.9	966	1169	1041	5:01.4	1068	5:19.8	*922	952	922	5:53.6	922	2:29.8
l40	1034	1324	1214	2:51.5	1236	3:02.4	955	1105	1009	5:47.6	1009	5:41.9	*955	955	955	1:08.2	955	1:08.5

Table 2

		$N1-1000$	$N2-1000$	$N1-5000$	$N2-5000$
edata	ave (%)	5.2	5.3	4.8	4.5
	max (%)	24.0	22.8	23.4	19.8
rdata	ave (%)	2.8	2.9	2.3	2.3
	max (%)	13.4	14.5	12.0	10.7
vdata	ave (%)	0.5	0.5	0.4	0.4
	max (%)	3.2	3.1	1.9	2.1

- $N1-1000$: The tabu-search algorithm based on neighborhood $N1$ with a maximum of 1000 iterations.
- $N2-1000$: The tabu-search algorithm based on neighborhood $N2$ with a maximum of 1000 iterations.
- UB: The makespan of the obtained solution.
- CPU: The CPU-time (hours:minutes:seconds).

Table 2 summarizes the obtained results, both, with a maximum of 1000 and 5000 iterations. By ave (max) we denote the average (maximal) percentage deviation from the best known lower bound.

The results can be summarized as follows.

- Both neighborhoods give very similar results. $N1$ is a little better than $N2$ if the number of iterations is limited to 1000, $N2$ is better than $N1$ if this number is limited to 5000. In average the $N2$-heuristics need more computational time than the $N1$-heuristics, even if the number of iterations is identical. In the worst case $N2$ needs 80% more time than $N1$ (for a benchmark problem with 5000 iterations).
- It is only useful to consider 5000 iterations for the "large" problems of edata and rdata. In the remaining cases an improvement is obtained only in a few cases, and the improvements are only small. Especially for large vdata problems $N1-5000$ and $N2-5000$ need an enormous amount of computational time: in five cases the heuristics run for approximately 10 hours.
- In most cases the tabu-search algorithm improves the initial solution given by beam3 considerably.
- Only for a small number of small problems the algorithms terminate before the maximal number of iterations is reached. In most of these cases the value of the best found solution is equal to a lower bound, i.e. the best found solution is optimal.

Summarizing the tabu-search heuristic yield excellent results for almost all benchmark problems. The best results are obtained for the vdata problems. The average deviation from the best lower bound is 0.5%, even if the number of iterations is limited to 1000.

The computational times for the heuristics with a maximal number of 1000 iterations are not too large (max. 16 min for edata, max. 40 min for rdata, and max. 2 h for vdata). The computational time strictly increases with the number of jobs. If the number of jobs is less than or equal to 15 then 1000 iterations never need more than 15 min!

5. Concluding remarks

We have considered job-shop scheduling problems with multi-purpose machines. These problems arise in the area of flexible manufacturing systems where tool equipped machines can execute different types of operations.

We proposed a tabu-search based algorithm for calculating good heuristic solutions for the MPM job-shop problem. Initial solutions are calculated using a fast heuristic based on insertion techniques and beam search.

We presented computational results derived by testing the algorithms which have been developed in this paper on a number of benchmark problems. The tabu search algorithms yield excellent results for almost all problems.

Acknowledgements. We gratefully acknowledge the comments of two anonymous referees which improved the presentation of the paper.

References

Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job-shop scheduling. Manag Sci 34:391–401

Applegate D, Cook W (1991) A computational study of the job shop scheduling problem. ORSA J Comput 3:149–156

Brandimarte P (1993) Routing and scheduling in a flexible job schop by tabu search. Ann Ope Res 41:157–183

Brucker P, Jurisch B, Sievers B (1992) A branch and bound algorithm for the job-shop scheduling problem. Osnabrücker Schriften zur Mathematik, Reihe D, Heft 136 (to appear in: Discr Appl Math)

Brucker P, Schlie R (1990) Job-shop scheduling with multi-purpose machines. Computing 45:369–375

Carlier J, Pinson E (1989) An algorithm for solving the job-shop problem. Manag Sci 35:164–176

Carlier J, Pinson E (1990) A practical use of Jackson's preemptive schedule for solving the job shop problem. Ann Oper Res 26:269–287

Dell'Amico M, Trubian M (1993) Applying tabu search to the job-shop scheduling problem. Ann Oper Res 41:231–252

Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial scheduling. Prentice Hall, Englewood Cliffs, pp 225–251

Glover F (1989) Tabu search, Part I. ORSA J Comput 1:190–206

Glover F (1990) Tabu search, Part II. ORSA J Comput 2:4–32

Graboswski J, Nowicki E, Zdrazalka S (1986) A block approach for single machine scheduling with release dates and due dates. Eur J Oper Res 26:278–285

Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Tech J 45:1563–1581

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a Survey. Ann Disc Math 5:287–326

Hurink J (1992) Polygon scheduling. Dissertation, Fachbereich Mathematik/Informatik, Universität Osnabrück

Jurisch B (1992) Scheduling jobs in shops with multi-purpose machines. Dissertation, Fachbereich Mathematik/Informatik, Universität Osnabrück

Laarhoven PJM van, Aarts EHL, Lenstra JK (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125

Meyer W (1992) Geometrische Methoden zur Lösung von Job-Shop Problemen und deren Verallgemeinerungen, Dissertation, Fachbereich Mathematik/Informatik, Universität Osnabrück

Ow PS, Morton TE (1989) The single machine early/tardy problem. Manag Sci 35:177–191

Papadimitriou CH, Steiglitz K (1982) Combinatorial optimization. Prentice Hall, Englewood Cliffs

Roy B, Sussmann B (1964) Les problèmes d'ordonnancement avec contraintes disjonctives, Note DS no. 9 bis, SEMA, Paris

Salvador MS (1973) A solution of a special class of flowshop scheduling problems. Proceedings of the Symposium on the Theory of Scheduling and its Applications. Springer, Berlin Heidelberg New York, pp 83–91

Thole M (1993) Lösung von Multi-Purpose Job-Shop Problemen durch Tabu-Suche, Diplomarbeit, Fachbereich Mathematik/ Informatik, Universität Osnabrück

Werner F, Winkler A (1991) Insertion techniques for the heuristic solution of the job shop problem. TU Magdeburg, Preprint 26/91