ELSEVIER

Discrete Applied Mathematics 58 (1995) 191–211

DISCRETE APPLIED MATHEMATICS

Insertion techniques for the heuristic solution of the job shop problem

Frank Werner, Andreas Winkler

Fakultät für Mathematik, Technische Universität "Otto von Guericke", PSF 4120, 39016 Magdeburg, Germany

Received 7 January 1992; revised 20 November 1992

Abstract

In this paper we deal with the heuristic solution of the classical job shop problem. Both the constructive and the iterative phase of our algorithm apply insertion techniques combined with beam search. In the first phase we successively insert the operations into feasible partial schedules. In the iterative phase we generate paths in a particular neighbourhood graph instead of investigating the neighbourhood completely. To select "interesting" neighbours, we use the combinatorial path structure of feasible solutions of the job shop problem. The results of our algorithm are compared with those from other well-known methods on benchmark problems.

Keywords: Scheduling; Job shop problem; Neighbourhood heuristics; Insertion algorithm; Path search

1. Introduction

In this paper the classical job shop problem is considered. n jobs $J_1, J_2, \ldots, J_n$ have to be processed on m machines $M_1, M_2, \ldots, M_m$. Each job consists of several operations where the operation (i, j) represents the processing of J_i on M_j. The corresponding processing time is denoted as t_{ij}. For each job, the machine orders of all jobs and the processing times of all operations are given. Furthermore the usual assumptions hold i.e. each job can be processed only on one machine at the same time, each machine can handle only one operation simultaneously, and the processing of an operation may not be interrupted. The objective is to minimize the maximum completion time $C_{\max}$ of a job.

The job shop problem is not only NP-hard, even among the members of this class it belongs to the worst in practice. A problem with 10 jobs and 10 machines given by Muth and Thompson [13] was solved optimally only a couple of years ago. Branch and Bound methods for the job shop problem are given for instance in [5–7].

Job shop scheduling is an important practical problem, hence it is natural to look for heuristics. Most of these job shop heuristics are based on "priority dispatching"

SSDI 0166-218X(93)E0127-K

rules. Such rules select an operation from a subset to be scheduled next. A survey of these rules is contained for instance in [10]. By means of such rules an active schedule is generated, i.e. no operation can be started earlier without delaying some other operation. Such one-pass algorithms are fast and the hope is that the generated solutions are not too bad.

We only mention that these priority dispatching rules can also be randomized. A randomized rule is to select one of the available operations at random from a probability distribution which makes the odd of being selected proportional to the priority assigned to each operation by the applied rule. In general several runs of such randomized algorithms are made.

Iterative methods investigate a particular neighbourhood within the set of feasible solutions. We mention the algorithms from [19] where a special shift neighbourhood has been applied. This algorithm uses the critical path for generating neighbours which enables to generate only a small number of schedules. We only mention that also such general techniques as simulated annealing or tabu search can be applied to solve the job shop problem approximately. Contrary to usual local search, both types of iterative algorithms can accept also solutions which do not yield an improvement of the objective value. For details the reader is referred to [11, 8].

Finally we mention a method from the recent literature [1] which integrates iterative improvement into a constructive algorithm. This algorithm sequences the machines one at a time, consecutively. In order to do this, for each machine not yet sequenced, the sequence of each previously sequenced machine is reoptimized by solving the one-machine problem again. An extension of this shifting bottleneck procedure mentioned above consists in applying this algorithm to the nodes of a partial enumeration tree. Further details are contained in [1].

In this paper a basic concept for schedule construction and iterative improvement is derived. Both phases use insertion techniques which have been successfully applied to many combinatorial problems. For permutation problems (i.e. the set of feasible solutions is given by the set P_n of permutations of n integers or by a subset of P_n), insertion algorithms often generate good solutions. Such algorithms successively complete a partial solution. If a partial sequence $(p_1, \ldots, p_k)$ with $k < n$ already exists, the element h which has to be inserted next is determined. Then we try to insert h on all possible positions (at most $k + 1$) such that a feasible subsequence results and the permutation with the best objective value is taken as initial sequence for the next insertion step. For the travelling salesman problem several insertion algorithms have been developed (cf. [16]) which differ by the rule for selecting the element for the next insertion. Up to now the insertion algorithm by Nawaz et al. [14] is the probably best constructive algorithm for the permutation flow shop problem.

In [20] it has been shown that the job shop problem can also be handled as such a permutation problem with precedence constraints which are imposed by the machine orders of the jobs. Each feasible solution can be described by a permutation of the operations. This permutation must represent the given machine orders i.e. the operations of each job must occur according to this order in the permutation. Hence,

all feasible schedules can be determined by topological enumeration of the operations (note that several permutations can describe the same schedule). This allows the application of the mentioned insertion techniques also for the job shop problem.

In [2] another model for describing feasible schedules has been introduced which is based on the representation of a schedule by a special latin rectangle. Because of its simplicity we also use this model.

In Section 2 some basic notations are introduced. Section 3 describes the use of insertion techniques for the construction of a feasible schedule combined with the principle of beam search from artificial intelligence. In Section 4 we demonstrate how the mentioned insertion ideas can be applied for the iterative approximate solution of our problem. Here we improve the iterative heuristic presented in [20]. The developed algorithm is based on the idea of path search that makes it possible to leave local optima again. Instead of investigating a particular neighbourhood structure completely, a limited number of paths in the corresponding neighbourhood graph is generated within each iteration. Finally we give computational results in Section 5.

2. The job shop problem and feasible schedules

In this paper we consider the variant that each job has to be processed on each machine at most once, i.e. for the number m_i of the operations of J_i we have $m_i \leqslant m$. First we describe the mentioned block-matrices-model without regarding the fixed machine orders in the case of the job shop problem. From the literature it is well-known that each combination of machine and job orders can be described by a directed graph $G = (V, E)$.

Let us consider the example in Fig. 1 with 3 jobs and 4 machines.

The vertex set V is given by the set of operations (i, j). The set E contains horizontal and vertical arcs which describe the machine and the job orders. For instance, from row 2 in Fig. 1 follows that job J_2 has machine order $M_3 \Rightarrow M_1 \Rightarrow M_2 \Rightarrow M_4$ and according to column 3 we have job order $J_2 \Rightarrow J_3 \Rightarrow J_1$ on machine M_3.

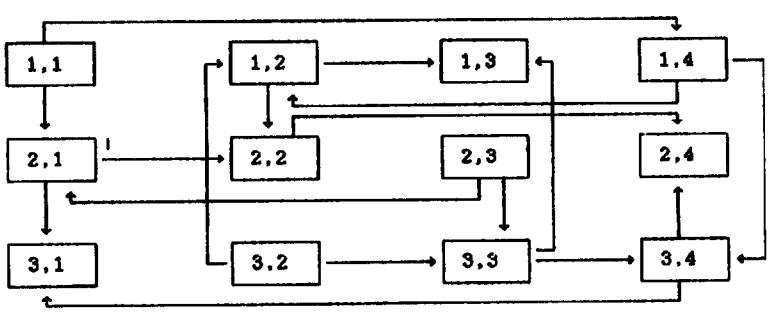


Fig. 1. $G = (V, E)$.

All machine and job orders can be described in a suitable manner by the matrices MO and JO where mo_{ij} and jo_{ij} state the position of M_j in the machine order of J_i and the position of J_i in the job order on machine M_j, respectively. For the example in Fig. 1 we have the following matrices:

$$\mathrm{MO} = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 2 & 3 & 1 & 4 \\ 4 & 1 & 2 & 3 \end{bmatrix} \quad \text{and} \quad \mathrm{JO} = \begin{bmatrix} 1 & 2 & 3 & 1 \\ 2 & 3 & 1 & 3 \\ 3 & 1 & 2 & 2 \end{bmatrix}.$$

Note that each row of MO represents a permutation of the integers $1, \ldots, m$. Analogously, a permutation of $1, \ldots, n$ is contained in each column of JO.

The mentioned model which we call block-matrices-model is based on a one-to-one correspondence between feasible schedules and special latin rectangles. A latin rectangle $\mathrm{LR}[n, m, r] = [a_{ij}]$ is an (n, m) matrix with elements from an insertion set $S = \{1, \ldots, r\}$ such that every element of S occurs at most once in each row and in each column. Now we consider only latin rectangles with the following additional property:

For each $a_{ij} > 1$, the integer $a_{ij} - 1$ exists in row i or in column j. (1)

We can formulate the following theorem.

Theorem 1. *We can one-to-one assign a given latin rectangle with the property* (1) *to a feasible schedule.*

For the proof the reader is referred to [2]. Considering the above example in Fig. 1, we obtain as corresponding latin rectangle

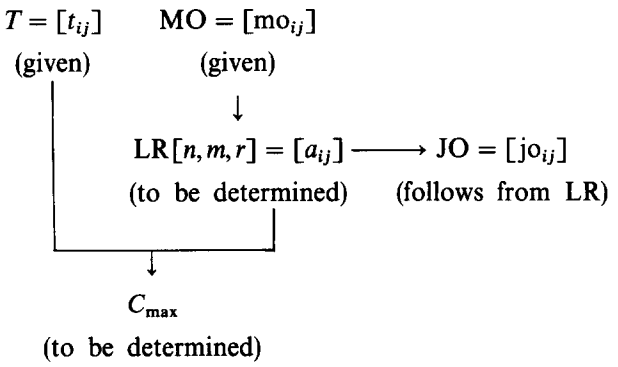
$$\mathrm{LR}[3, 4, 5] = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 2 & 4 & 1 & 5 \\ 4 & 1 & 2 & 3 \end{bmatrix}.$$

It contains the "logical orders" according to the matrices MO and JO which can be obtained by ordering the numbers rowwise (and columnwise) from the smallest integer to the largest one. Because of the above relation, we denote the graph introduced in Fig. 1 in the following by $G(\mathrm{LR})$.

If we introduce vertex costs given by the processing times of the corresponding operations, then the length of a critical path (i.e. the greatest sum of vertex costs of a path in $G(\mathrm{LR})$) yields the makespan $C_{\max}$ of a schedule LR.

In this paper we are concerned with the job shop problem, i.e. the machine order is given for each job J_i. Hence, the set of feasible schedules is characterized by the set of latin rectangles with property (1) such that the rows of $\mathrm{LR}[n, m, r]$ represent the fixed machine orders of the jobs.

Consequently, the block-matrices-model can be summarized in this case in the following form:



Our algorithm operates with partial schedules represented by partial latin rectangles. Here only a subset of the operations has been inserted into the rectangle. With respect to such a partial schedule, we can introduce a head r_{ij} and a tail q_{ij} for each operation in the usual manner, i.e. r_{ij} denotes the length of a longest path from one of the sources to (i,j) and q_{ij} represents the length of a longest path from (i,j) to one of the sinks. In each case the vertex cost of (i,j) is not included.

Example 1. Let $n = 3$, $m = 4$ and

$$T = \begin{bmatrix} 12 & 14 & 9 & 8 \\ 17 & 10 & 12 & 11 \\ 6 & 15 & 3 & 13 \end{bmatrix}.$$

Moreover, let the machine orders be as in the example of Fig. 1. We consider the following partial schedule:

$$\mathrm{LR}' = \begin{bmatrix} 1 & 3 & \cdot & 2 \\ \cdot & 4 & 1 & 5 \\ 4 & \cdot & 2 & 3 \end{bmatrix}$$

which does not violate any given machine order. The corresponding graph $G(\mathrm{LR}')$ is shown in Fig. 2.

In the case of the job shop problem all horizontal arcs are inserted at the beginning. For instance, for operation (3,4), we have $r_{34} = \max\{15, 18, 20\} = 20$ and $q_{34} = \max\{6, 11\} = 11$.

Finally we only mention that our representation of schedules and partial schedules by latin rectangles is closely related to the well-known disjunctive graph model (cf. for example [7]). If using this model, a certain number of disjunctive arcs have been fixed, then our rectangle is the rank matrix of the graph containing the conjunctive and fixed

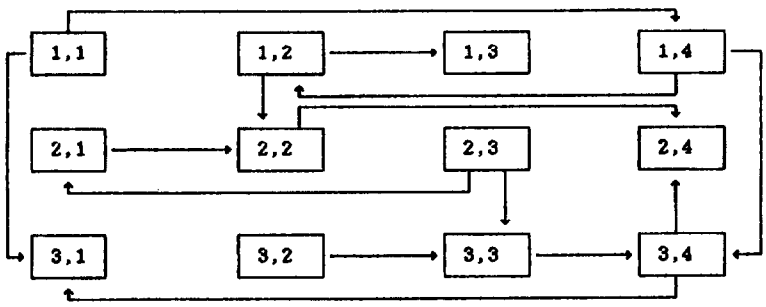


Fig. 2. $G(\mathrm{LR}')$.

disjunctive arcs and all operations as vertices. However, we use the model introduced above because of its simplicity and compact representation of schedules.

3. An insertion algorithm for constructing a feasible schedule

In [2] an enumerative algorithm has been developed for the open shop problem. This branch and bound method is based on the successive insertion of an operation into a partial schedule such that a feasible schedule results again, i.e. only a subset of the positions (i,j) in the corresponding latin rectangle is occupied and with respect to this subset property (1) holds. Bräsel and Werner [4] demonstrated how this procedure can be modified for the exact solution of the job shop problem. In this section we apply this procedure for the construction of a feasible schedule by inserting the operations successively according to nonincreasing processing times.

Next we describe this insertion algorithm in detail. We start with a partial latin rectangle which contains elements only in one row, i.e. the job with the greatest sum of processing times is inserted first. The corresponding numbers in $\mathrm{LR}[n,m,r]$ directly follow from the given matrix MO. Now we successively insert individual operations into the partial latin rectangle according to nonincreasing processing times. This is an analogy to the insertion algorithm by Nawaz et al. for the permutation flow shop problem where the jobs are inserted according to nonincreasing sums of the processing times belonging to each job.

Now we consider the question how the selected operation can be inserted into the partial latin rectangle. The following conditions have to be satisfied with respect to an insertion of operation (i,j):

(a) If $\mathrm{mo}_{ij} - \mathrm{mo}_{ik} = \Delta > 0$, then $a_{ij} - a_{ik} \geqslant \Delta$ must hold (especially, this includes $a_{ij} \geqslant \mathrm{mo}_{ij}$);

(b) a_{ij} is chosen such that the integer $a_{ij} - 1$ occurs at most once in row i or in column j (we mention that possibly $a_{ij} - 1$ is not inserted in row i but it results from MO, cf. (a));

(c) all previously established precedence relations between operations must remain valid and the insertion of (i,j) may not create any cycle in the corresponding graph.

Condition (c) requires that the partial schedule must be modified in the case that the inserted number k already occurs in row i or column j. All the operations with number k in row i or column j and their successors must also be successors of the inserted operation (i,j) in the resulting schedule described by $G(\mathrm{LR}')$. Next we answer the question which costs are assigned to a partial schedule. If operation (i,j) has been inserted, the corresponding costs $g(\mathrm{LR}')$ of the resulting schedule LR′ are given by the longest path through operation (i,j), i.e.

$$g(\mathrm{LR}') = r_{ij} + t_{ij} + q_{ij}$$

are the costs of LR′.

Our experiments demonstrated that the above criterion turned out to be better than taking the longest path in $G(\mathrm{LR}')$ as costs. For all possible insertions where the inserted operation does not belong to a longest path, there would be no sharp criterion to evaluate these insertions. However, applying our criterion, the costs are not necessarily monotonically nondecreasing with respect to the successive insertion of operations.

Considering Example 1 again, we illustrate the above insertion algorithm.

We start with

$$\mathrm{LR}^0 = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 2 & 3 & 1 & 4 \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

because J_2 has the greatest sum of processing times. Next we consider operation (3,2). We have the following possibilities for inserting (3,2):

$$\mathrm{LR}^1 = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 2 & 3 & 1 & 4 \\ \cdot & 1 & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \mathrm{LR}^2 = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 2 & 3 & 1 & 4 \\ \cdot & 4 & \cdot & \cdot \end{bmatrix}$$

We have $g(\mathrm{LR}^1) = 37$ and $g(\mathrm{LR}^2) = 76$. Hence LR^1 is taken for the further considerations. Now we insert operation (1,2) with $t_{12} = 14$. The following partial schedules are obtained.

$$\mathrm{LR}^3 = \begin{bmatrix} \cdot & 3 & \cdot & \cdot \\ 2 & \underline{5} & 1 & \underline{6} \\ \cdot & \underline{4} & \cdot & \cdot \end{bmatrix}, \quad \mathrm{LR}^4 = \begin{bmatrix} \cdot & 3 & \cdot & \cdot \\ 2 & \underline{4} & 1 & \underline{5} \\ \cdot & 1 & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \mathrm{LR}^5 = \begin{bmatrix} \cdot & 4 & \cdot & \cdot \\ 2 & 3 & 1 & 4 \\ \cdot & 1 & \cdot & \cdot \end{bmatrix}.$$

The underlined integers represent the modifications which result from the insertion of operation (1,2) (condition (c)). Because of $g(\mathrm{LR}^3) = 71$, $g(\mathrm{LR}^4) = 55$ and

$g(LR^5) = 62$, LR^4 is chosen. In the next step we insert operation (3,4) into the partial schedule LR^4. Hence we obtain

$$LR^6 = \begin{bmatrix} \cdot & 3 & \cdot & \cdot \\ 2 & 4 & 1 & 5 \\ \cdot & 1 & \cdot & 3 \end{bmatrix} \quad \text{and} \quad LR^7 = \begin{bmatrix} \cdot & 3 & \cdot & \cdot \\ 2 & 4 & 1 & 5 \\ \cdot & 1 & \cdot & 6 \end{bmatrix},$$

with $g(LR^6) = 42$ and $g(LR^7) = 68$, i.e. LR^6 is selected.

Proceeding in this manner, we finally obtain

$$LR^* = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 2 & 4 & 1 & 5 \\ 4 & 1 & 2 & 3 \end{bmatrix},$$

with $C_{max}(LR^*) = 55$ which is an optimal solution.

As we already mentioned, the basic principle of beam search has also been applied (cf. [15]). The basic idea of this approach is to search a limited number of partial solutions in parallel. If a beamwidth of k is applied, we select in each step the k best partial latin rectangles which form the beam. From the set of resulting partial schedules the k best ones are selected with respect to the objective function we use.

Considering Example 1 again, for beamwidth $k = 2$ in Step 1 both partial schedules LR^1 and LR^2 are considered. In both partial schedules we check all possible insertions for operation (1,2).

This yields six new partial schedules, i.e. LR^3, LR^4, LR^5 as well as

$$\overline{LR^3} = \begin{bmatrix} \cdot & 3 & \cdot & \cdot \\ 2 & 4 & 1 & 5 \\ \cdot & 5 & \cdot & \cdot \end{bmatrix}, \quad \overline{LR^4} = \begin{bmatrix} \cdot & 4 & \cdot & \cdot \\ 2 & 3 & 1 & 4 \\ \cdot & 5 & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \overline{LR^5} = \begin{bmatrix} \cdot & 5 & \cdot & \cdot \\ 2 & 3 & 1 & 4 \\ \cdot & 4 & \cdot & \cdot \end{bmatrix}.$$

From this set we select two partial schedules with the lowest costs. Then we proceed in this manner until k complete schedules have been generated. The schedule with the best objective value is taken as heuristic solution.

4. Iterative improvement algorithm

In this section we describe how the concept of Section 3 can be used for improving the constructive solution iteratively. This algorithm is a generalization of the procedure described in [20]. Is is also a variant of the path algorithms introduced by Werner [21]. Neighbourhood structures can be described by directed or undirected graphs where the vertex set is given by the set of all feasible solutions and we have an arc from a solution to all neighbours. If we have an arc between two vertices in both directions,

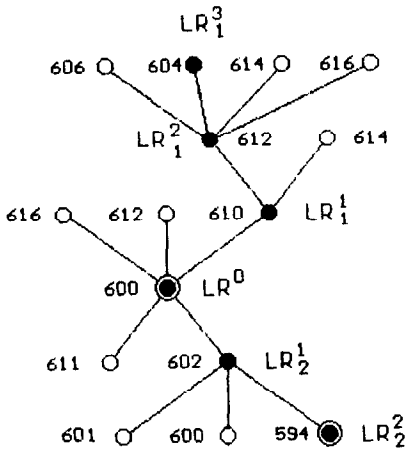


Fig. 3. Path search.

both arcs are replaced by an edge. In such a neighbourhood graph we do not generate all neighbours (i.e. all adjacent vertices) as in the case of usual local search, but we generate a limited number of paths. In each case the next vertex on the path is determined by choosing the schedule with the best objective value from a certain subset of neighbours. A generated schedule is accepted as new initial solution if for the first time a better objective value has been obtained on a path. The path search is illustrated in Fig. 3 with a maximal path length of 3.

Here LR_i^j denotes the jth neighbour within path i. Let the given integers denote the objective values for the corresponding schedules. Starting from LR^0 as initial solution, we do not find an improved solution on the first path (600–610–612–604). A better solution is found for the first time on the second path by LR_2^2. This schedule is taken as initial solution for the next iteration.

Next we answer the following two questions:

(1) Which neighbourhood structure is used in our algorithm?

(2) How should the path search be organized? Because we do not investigate the underlying neighbourhood completely, the question is which are the most "interesting" neighbours to generate the paths.

To answer the first question, let us first consider a permutation problem where the set of feasible solutions can be described by the set P_n of permutations of n jobs or by a subset of P_n. Here often shift neighbourhoods yield good results in local search algorithms (cf. [21]). In this case a neighbour is generated by choosing an arbitrary job of the current sequence and reinserting this job on a different position. The reinsertion can be restricted to a smaller position (left shift) or to a larger position (right shift). For $n = 3$, the shift graph $G^S(n)$ and the left shift graph $G^{LS}(n)$ are shown in Fig. 4.

In [21] these considerations have been extended to the case that a feasible solution can be described by a set of permutations (note that the job shop problem belongs to

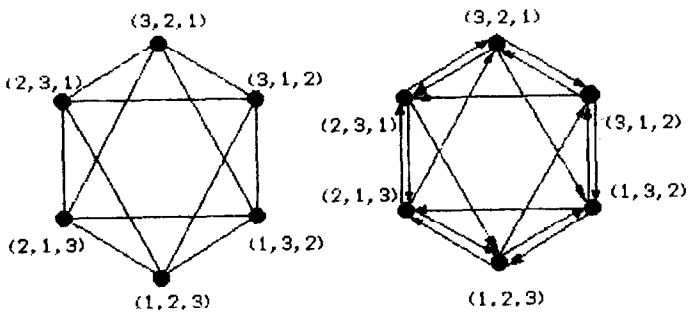


Fig. 4. $G^S(3)$ and $G^{LS}(3)$.

this class if we write the job order on each machine as a permutation of the processing order of the jobs on that machine). A similar shift neighbourhood graph $G^S(n, m, \mathrm{MO})$ can be defined for such problems. Two vertices (i.e. feasible schedules) are adjacent if there exists a job such that after deleting this job in both schedules the remaining schedules of $n-1$ jobs are identical, i.e. to generate a neighbour in this graph we select an arbitrary job and reinsert this job on an arbitrary position on the processing order of each machine such that a feasible schedule results. Analogously, a right or a left shift graph can be introduced. Here a neighbour is generated by selecting an arbitrary job and reinserting this job on a larger (smaller) position in some of the processing orders on the machines. In [21] it has been proven that these right and left shift graphs have the diameter n for $n \geqslant 3$. It can be easily shown that this is also true for the shift graph $G^S(n, m, \mathrm{MO})$. In our algorithm we apply the shift neighbourhood $G^S(n, m, \mathrm{MO})$. However, we either generate a right shift or a left shift neighbour, i.e. such neighbours where some of the operations of a selected job are shifted to the right in the processing order and other operations of this job are shifted to the left will not be considered.

We now consider the question of organizing the path search in the underlying neighbourhood. Because the number of neighbours is rather large, it is not recommendable to calculate the objective values of all neighbours. We use the approach by Werner [18] which is a slight generalization of the block approach by Grabowski [9]. To explain these ideas, we first consider the case that only a right or left shift of one operation of a job is allowed. Grabowski introduced blocks of operations of a critical path in the graph $G(\mathrm{LR})$ of the current schedule LR. A block is a maximal set of at least two operations of the critical path which are processed on the same machine without any idle time between these operations (see also Fig. 5). The main theorem by Grabowski is as follows.

Theorem 2. *Let* LR *be a feasible schedule. If* LR′ *is a feasible schedule with* $C_{\max}(\mathrm{LR}') < C_{\max}(\mathrm{LR})$, *then there exists an operation of a block of the critical path in* $G(\mathrm{LR})$ *that is processed in* LR′ *before the first operation of this block or there exists an operation of a block that is processed in* LR′ *after the last operation of this block.*

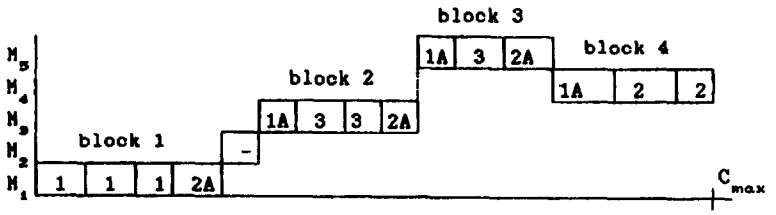


Fig. 5. Blocks of a critical path in G(LR).

Theorem 2 gives a necessary condition for an objective function improvement of LR′ with respect to LR. Werner [18] gave a slight generalization of this theorem. He considered only shifts of operations such that in the graph G(LR′) of the generated neighbour LR′ there does not exist any path with the same vertices as the critical path in the graph G(LR) of the current schedule LR (in the other case it is clear that the objective value of LR′ cannot be better than the objective value of LR). Then it is possible that further shifts of operations that belong to the first or to the last block can be excluded. More precisely, if a neighbour with the above necessary condition for an objective function improvement should be generated by a shift of one operation, we have to consider the following types of operations of a block:

Type 1: operations of this type have to be reinserted after the last operation of this block (but not necessarily immediately after this operation);

Type 1A: operations of this type have to be reinserted at least one position later in the processing order of the corresponding machine;

Type 2: operations of this type have to be reinserted before the first operation of this block (but not necessarily before this operation);

Type 2A: operations of this type have to be reinserted at least one position earlier in the processing order of the corresponding machine;

Type 3: operations of this type have to be reinserted before the first of after the last operation of the corresponding block.

In Fig. 5 an example of a critical path in G(LR) is shown where we use the Gantt chart for the representation. Moreover, the type of each operation of a block is also given.

Next we describe the generation of a complete shift neighbour in the neighbourhood $G^S(n, m, \mathrm{MO})$ which we use in our algorithm. We already mentioned that we only generate such neighbours where the operations are shifted in the same direction (i.e. to the right or to the left). Our iterative algorithm is closely related to the constructive algorithm of the last section because we delete all operations of a job and then we reinsert these operations in a similar way. However, we have additional restrictions, i.e. some reinsertions are forbidden to ensure that a shift neighbour is generated that satisfies the necessary condition for an objective function improvement. Because we allow the reinsertion of an operation on its previous position, it is possible that only a subset of operations of a job is shifted.

First we describe the generation of a right shift neighbour. Our algorithm selects a job which contains at least one operation in a block of the critical path in $G(\mathrm{LR})$. To describe the generation of a right shift neighbour, we assume that a job J_i has been selected that has one operation, say O_{ik}, of type 1, 1A or 3 in a block, i.e. a right shift of this operation is allowed. Then we determine for each operation of J_i the minimal position of its reinsertion in the processing order of the machines. For operation O_{ik} the first possible reinsertion is after the corresponding block and the remaining operations of J_i can be reinserted beginning from the previous position in the processing order of the corresponding machine.

After determining all these minimal positions for reinserting the operations, we delete row i in the current schedule LR and we actualize the remaining rectangle such that property (1) of a schedule is satisfied, i.e. we have a feasible schedule which contains the operations of all jobs besides J_i. Now all operations of J_i will be reinserted according to the machine order of the job, i.e. first the operation of J_i with $\mathrm{mo}_{ij} = 1$ is reinserted, then the operation with $\mathrm{mo}_{ij} = 2$ and so on. For each position, beginning with the smallest possible one, the costs $r_{ij} + t_{ij} + q_{ij}$ for the inserted operation O_{ij} are calculated. If we have at least for one insertion $r_{ij} + t_{ij} + q_{ij} < C_{\max}(\mathrm{LR})$ and then 3 insertions on consecutive positions do not satisfy the above inequality, we do not consider further reinsertions on larger positions in the processing order because in this case a schedule with an objective value improvement becomes unlikely. As in the constructive phase, we can apply the principle of beam search. Again, in the case of beamwidth k, in each step the k best insertions of an operation will be considered for the next step.

Analogously, a left shift neighbour can be generated with respect to an operation of type 2, 2A or 3 of a block of the critical path in $G(\mathrm{LR})$. However, we take maximal positions of the operations into consideration and, because of symmetry, the operations are reinserted from the last to the first.

If a job J_i has been selected, each operation of J_i contained in a block of the critical path in $G(\mathrm{LR})$ is taken as "initial" operation for generating a right or left shift neighbour (according to its type) and, finally, from the set of generated neighbours the schedule with the best objective value is taken as neighbour within the actual path. Hence, the generation of a shift neighbour can be summarized as follows:

Algorithm 1. Generation of a shift neighbour

Input: *initial solution* LR^0 *of the iteration,*
actual schedule LR *within the path,*
job J_i *which has been selected for performing a shift transformation,*
beamwidth k;

Output: *generated shift neighbour* LR′

begin

determine the set OP_i of operations of J_i belonging to the blocks of the critical path in $G(\mathrm{LR})$; $\mathrm{SN} := \emptyset$; determine LR* by deleting row i in LR and actualizing the remaining rectangle such that property (1) holds;

while $OP_i \neq \emptyset$ **do**
begin determine the operation $(i,j) \in OP_i$ with the greatest processing time;
ty := type of operation (i,j);
Generation of a right shift neighbour of LR
if ty $\in \{1, 1A, 3\}$ **then**
begin determine for all operations of J_i by means of LR the minimal integer in JO such that a right shift neighbour with the above additional restriction results; $S := \{LR^*\}$;
for $j := 1$ **to** m_i **do**
begin for each LR $\in S$ **do**
determine all possible reinsertions of the jth operation of J_i (by considering row i in JO), actualize these schedules such that property (1) holds and put the resulting schedules in S^*;
if $|S^*| > k$ **then** remove all LR $\in S^*$ except the (partial) schedules with the k lowest costs;
$S := S^*$;
end;
SN := SN $\cup$ S;
end;
Generation of a left shift neighbour
if ty $\in \{2, 2A, 3\}$ **then**
generate a left shift neighbour in the same manner and add the k best complete schedules to SN;
$OP_i := OP_i \setminus \{(i,j)\}$;
end;
determine the schedule LR′ $\in$ SN with the lowest C_{max} value;
end.

One possible modification of the above algorithm consists in accepting an improvement directly (i.e. it is not necessary to consider first all operations in OP_i before accepting an improvement).

Then the iterative path algorithm for the heuristic solution of the job shop problem is as follows:

Algorithm 2. Generation of the paths
Input: *initial schedule* LR^0, *maximal path length* l, *beamwidth* k, *maximal number of iterations* u_{max};
Output: *heuristic solution* LR^0
begin $u := 0$;
M1: determine the blocks of the critical path in $G(LR^0)$;
determine the set A of jobs having at least one operation in a block of the critical path in $G(LR^0)$;
while $A \neq \emptyset$ **do**

```
begin determine the job J_i ∈ A which has an operation with the greatest
      processing time in a block;
      generate a shift neighbour LR^1 of LR^0 by applying Algorithm 1 with
      respect to J_i and beamwidth k;
      if C_max(LR^1) < C_max(LR^0) then
      begin LR^0 := LR^1; u := u + 1;
            if u = u_max then STOP else go to M1;
      end;
      r := 1; A := A \ {J_i}; B := {J_i};
      while r < l do
      begin determine the set A* of jobs having at least one operation in
            a block of the critical path in G(LR^r);
            if A* \ B = ∅ then go to M2;
            select randomly a job J_i ∈ A* \ B;
            generate a shift neighbour LR^(r+1) of LR^r by applying Algorithm
            1 to J_i and beamwidth k;
            B := B ∪ {J_i};
            if C_max(LR^(r+1)) < C_max(LR^0) then
            begin LR^0 := LR^(r+1); u := u + 1;
                  if u = u_max then STOP
                               else go to M1;
            end;
            r := r + 1;
      end;
M2: end;
end.
```

In our tests we always applied $u_{max} = 4n$. Note that the above algorithm organizes the path search in such a way that within one path different jobs are selected to generate the shift neighbours. Moreover, our algorithm is a randomized one in the case of $l > 1$.

5. Computational results

A Turbo-Pascal implementation of the developed insertion algorithms was tested on a PS2 computer in the version 55 SX. The problems were taken from the literature.

We considered the following sets of test problems from [1]:

I: 5 problems with 10 jobs, 5 machines (2-1 to 2-5 where *i-j* means problem *j* in table *i* in [1]);
II: 5 problems with 20 jobs, 5 machines (2-11 to 2-15);
III: 5 problems with 10 jobs, 10 machines (2-16 to 2-20);
IV: 5 problems with 15 jobs, 10 machines (2-21 to 2-25);
V: 5 problems with 20 jobs, 10 machines (2-26 to 2-30);
VI: 5 problems with 15 jobs, 15 machines (2-36 to 2-40).

Moreover we included the problems 1-3, 1-5, 1-6 (10 jobs, 10 machines) and 1-4 (20 jobs, 5 machines) from [1] into our test.

For our algorithms we use the following abbreviations:

$C(k_c)$ – constructive insertion algorithm with beamwidth k_c;

$I(k_c, k_i, l)$ – iterative algorithm where the initial solution is obtained by $C(k_c)$ and in the iterative phase a beamwidth of k_i has been applied; the maximal path length is l.

Because our iterative algorithm is a randomized one for $l > 1$, we performed 3 runs in this case and the stated values refer to the average value of these 3 runs. If values in parentheses are given, they refer to the best value of the 3 runs for each example. Furthermore, CBEST represents the best value of the variants of our constructive algorithm and IBESTl gives the best result of all tested iterative variants with a maximal path length of l.

Moreover we considered the following algorithms from [1]:

PDR – best value of 10 runs with different priority dispatching rules;

RPDR – each of the 10 dispatching rules was randomized, the run is then repeated until ten consecutive runs produce no improvement, and the best result obtained is reported;

SBI – shifting bottleneck procedure by Adams et al.;

SBII – partial enumeration version of the shifting bottleneck procedure (for the sets I–VI a time limit set to the CPU time required by RPDR was imposed).

In Tables 1 and 2 we use the following abbreviations:

APD – average percentage of deviation from the optimal or best known solution, and

ACT – average computation times in seconds.

We note that all computation times of our algorithms refer to a PS/2 computer in the version 55 SX and the computation times of the algorithms from [1] (i.e. PDR, RPDR, SBI and SBII) refer to a VAX 780/11 computer.

Table 1 compares the results for some constructive algorithms. We tested our algorithm against priority dispatching rule heuristics. Our algorithm yields better results for larger beamwidth k. If we take the best value of three runs ($k = 1, 2, 3$), we obtain better results on average than algorithm PDR which generates 10 schedules. This is especially obvious for the larger problems. The randomized algorithm RPDR yields slightly better results than variant CBEST but it is a very time consuming algorithm. Nevertheless, we mention that CBEST obtained in 13 of 30 problems of the sets I–VI a better $C_{\max}$ value.

In Table 2 the results of some representative variants of our iterative algorithm and the versions of the Adams et al. algorithm are summarized for the sets I–VI. First, let us consider the variants of our algorithm with $l = 1$. The deviation ranges from 5% to 7% depending on k_c and k_i. The best variant ($k_c = k_i = 3$) yield similar results as SBI (especially we note the good results for the first four sets of problems). Clearly, if we take the best result of the runs with $l = 1$ in each case, then SBI is already outperformed.

Moreover, as can be seen from Table 2, the results are better for our iterative algorithm with $l = 3$ than with $l = 1$ (however, the computation times increase in this

Table 1
Results for the constructive algorithms for the problems of the sets I–VI

Problem	Objective values						
	OPT[a]	PDR	RPDR	$C(1)$	$C(2)$	$C(3)$	CBEST
2-1	666	679	679	671	671	666	666
2-2	655	792	727	722	707	690	690
2-3	597	673	634	691	701	657	657
2-4	590	670	621	693	681	621	621
2-5	593	594	594	593	593	593	593
APD	–	9.87	5.04	8.84	8.31	4.13	4.13
ACT	–	4	127	1	1	2	–
2-11	1222	1223	1223	1309	1247	1322	1247
2-12	1039	1041	1040	1074	1047	1039	1039
2-13	1150	1151	1151	1198	1150	1150	1150
2-14	1292	1293	1293	1348	1292	1292	1292
2-15	1207	1320	1314	1448	1346	1322	1322
APD	–	1.96	1.76	7.79	2.89	3.54	2.32
ACT	–	8	354	2	4	6	–
2-16	945	1036	1036	1032	1064	1016	1016
2-17	784	857	857	915	905	889	889
2-18	848	897	897	1016	995	935	935
2-19	842	926	898	913	950	965	913
2-20	902	1001	942	1048	1043	1003	1003
APD	–	9.14	7.16	14.07	14.60	11.23	10.16
ACT	–	7	237	2	4	6	–
2-21	1050	1208	1198	1272	1171	1218	1171
2-22	927	1085	1038	1112	1101	1112	1101
2-23	1032	1163	1108	1183	1160	1195	1160
2-24	935	1142	1048	1106	1058	1032	1032
2-25	977	1259	1160	1155	1107	1120	1107
APD	–	19.16	12.85	18.45	13.83	15.35	13.27
ACT	–	14	412	3	6	10	–
2-26	1218	1373	1373	1434	1414	1387	1387
2-27	1269[b]	1472	1417	1619	1518	1530	1518
2-28	1250[b]	1475	1402	1610	1466	1481	1466
2-29	1195[b]	1539	1382	1438	1377	1362	1362
2-30	1355	1604	1508	1513	1493	1513	1493
APD	–	18.78	12.70	21.22	15.68	15.71	14.99
ACT	–	26	838	6	13	25	–
2-36	1268	1517	1385	1538	1617	1450	1450
2-37	1423[b]	1670	1551	1519	1572	1577	1519
2-38	1184[b]	1405	1388	1519	1474	1477	1474
2-39	1233	1436	1341	1506	1438	1473	1438
2-40	1233	1477	1383	1468	1554	1479	1468
APD	–	18.38	11.28	19.51	21.03	17.87	16.26
ACT	–	25	844	9	17	30	–

Table 1 (Continued)

Problem	Objective values						
	OPT[a]	PDR	RPDR	$C(1)$	$C(2)$	$C(3)$	CBEST
total							
APD	–	12.88	8.47	14.98	12.72	11.31	10.19
ACT	–	14	469	4	7	12	–

[a]OPT – optimal or best known value.
[b]Means that optimality has not been proven.

Table 2
Results for the iterative algorithms for the problems of the sets I–VI

Problem	SBI	SBII	$I(3,1,1)$	$I(3,3,1)$	$I(3,1,3)$	$I(3,2,3)$	IBEST1	IBEST3
2-1	666	666	666	666	666	666	666	666
2-2	720	669	690	686	680.7	673.3	684	655
2-3	623	605	639	639	641.7	630.7	617	617
2-4	593	593	602	602	599.3	600.7	602	598
2-5	593	594	593	593	593	593	593	593
APD	3.09	0.80	2.88	2.76	2.60 (2.32)	2.06 (1.31)	1.96	0.80
ACT	2	30[a]	2	4	6	9	–	–
2-11	1222	1222	1222	1222	1222	1222	1222	1222
2-12	1039	1039	1039	1039	1039	1039	1039	1039
2-13	1150	1150	1150	1150	1150	1150	1150	1150
2-14	1292	1292	1292	1292	1292	1292	1292	1292
2-15	1207	1207	1263	1263	1207	1211.7	1207	1207
APD	0.00	0.00	0.00	0.00	0.00 (0.00)	0.35 (0.00)	0.00	0.00
ACT	2	–[a]	6	10	18	26	–	–
2-16	1021	978	1016	1016	1016	1016	1016	979
2-17	796	787	820	820	799.7	795.7	789	787
2-18	891	859	935	935	935	906.7	864	857
2-19	875	860	875	863	872	859.7	857	852
2-20	924	914	946	913	913.7	915	911	911
APD	4.20	1.73	6.74	2.70	4.93 (4.65)	3.89 (3.15)	2.51	1.50
ACT	8	237	7	13	33	58	–	–
2-21	1172	1084	1153	1108	1144	1117.7	1090	1091
2-22	1040	944	959	963	977	969.7	959	946
2-23	1961	1032	1054	1073	1053.7	1059.7	1054	1039
2-24	1000	976	993	993	990	989.7	993	976
2-25	1048	1017	1096	1077	1068.3	1057.3	1077	1046
APD	8.13	2.71	6.75	5.96	6.33 (5.37)	5.56 (4.27)	5.17	3.55

Table 2 (Continued)

Problem	SBI	SBII	$I(3,1,1)$	$I(3,3,1)$	$I(3,1,3)$	$I(3,2,3)$	IBEST1	IBEST3
ACT	24	374	30	97	127	223	–	–
2-26	1304	1224	1367	1367	1274.7	1255	1286	1244
2-27	1325	1291	1367	1409	1365.7	1362.7	1356	1318
2-28	1256	1250	1388	1362	1371	1309	1332	1289
2-29	1294	1239	1341	1341	1292.7	1297.7	1277	1235
2-30	1403	1355	1435	1456	1378.3	1408.7	1369	1362
APD	6.14	1.18	9.74	10.29	6.36 (4.60)	5.63 (3.78)	5.38	2.57
ACT	42	785	94	197	392	573	–	–
2-36	1351	1305	1397	1397	1380	1376	1357	1330
2-37	1485	1423	1552	1506	1482.3	1478.3	1470	1448
2-38	1280	1255	1396	1317	1320.3	1301.7	1317	1290
2-39	1321	1273	1360	1340	1348.3	1319.3	1319	1297
2-40	1326	1269	1379	1342	1327	1290.7	1296	1263
APD	6.74	3.02	11.86	8.87	8.29 (6.58)	6.73 (5.22)	6.73	4.64
ACT	63	844	78	243	355	673	–	–
total APD	4.72	1.57	6.33	5.10	4.75 (3.92)	4.04 (2.95)	3.62	2.18
ACT	24	491	36	94	155	260	–	–

[a] SBII was only started when SBI did not meet the lower bound.

case). This confirms that it often seems to be necessary to consider not only shift neighbours to overcome local optima. Because it is not useful to extend the cardinality of the neighbourhood and to compute the objective values of all neighbours, the concept of path search turned out to be a suitable alternative to usual local search schemes. In the case $l = 3$ the variant with $k_c = 3$ and $k_i = 2$ yields good results. The results show that the enumerative version SBII is an excellent heuristic but in 16 of 30 cases IBEST3 obtained at least the same value.

In addition to Table 2 we only mention that our iterative algorithm generates a very small number of schedules. In the case of $l = 1$, the average number of generated schedules ranges from 79 for $k_i = 1$ to 250 for $k_i = 3$. The variants with $l = 3$ generate on average 200–350 schedules depending on the chosen value of k_i in each case.

Table 3 summarizes the results for the problems 1-3 to 1-6. We note that 1-3 is the famous (10, 10) problem by Muth and Thompson [13]. Unfortunately, the results for PDR and RPDR are not stated for these problems in [1]. We only emphasize the excellent C_{max} value of 979 of the constructive version of our algorithm for problem 1-3. Usual local search algorithms often do not obtain such a good value. In [12] it is reported that within 6000 CPU seconds with deterministic local search more than

Table 3
Further results for benchmark problems

Problem:	1-3	1-4	1-5	1-6
Optimal value:	930	1165	1234	943
$C(1)$	1006	1270	1338	1104
$C(2)$	1001	1267	1338	1012
$C(3)$	979	1350	1381	1060
$I(1,2,1)$	1006	1188	1323	996
$I(2,2,1)$	964	1197	1288	996
$I(3,2,1)$	962	1181	1332	967
$I(1,2,3)$	956.3 (949)	1187.3 (1178)	1261.3 (1249)	981.7 (970)
$I(2,2,3)$	954.3 (949)	1193.0 (1179)	1265.0 (1260)	971.7 (958)
$I(3,2,3)$	956.3 (951)	1224.0 (1192)	1276.7 (1249)	974.3 (967)
SBI	1015	1290	1306	962
SBII	930	1178	1239	943
Computation times on a PS 2/55 SX [s]				
$C(1)$	2	2	2	2
$C(2)$	4	4	4	4
$C(3)$	6	6	6	6
$I(1,2,1)$	1	41	12	22
$I(2,2,1)$	19	38	17	8
$I(3,2,1)$	28	75	18	21
$I(1,2,3)$	78	121	60	65
$I(2,2,3)$	76	124	66	48
$I(3,2,3)$	66	145	82	61
SBI[a]	10	4	6	13
SBII[a]	851	80	1503	1101

[a] CPU seconds on a VAX 780/11.

9000 local optima have been generated with a best objective value of 1006. Our iterative variants with $l = 3$ obtained a best value of 949 in about one minute on a PS 2. In [12] some experiments with simulated annealing have been mentioned. It is reported that for the proposed standard variant an average length of 942.4 had been obtained, however, each run took about 6000 CPU seconds. Five runs of simulated annealing with a much slower cooling schedule produced an average schedule length of 933.4 but each run required 16 hours on average.

The above computational study did not include such algorithms as simulated annealing or tabu search. The reason for doing so was that our experience is that these mentioned algorithms require more computation time to produce similar results as the compared algorithms. This is due to the fact that simulated annealing or tabu search do not take advantage of structural properties of the special problem. Therefore, such algorithms have to generate more solutions than our iterative algorithm. So we compared only specific algorithms for the job shop problem.

6. Concluding remarks

Motivated by good results of insertion algorithms for a number of combinatorial optimization problems, we applied such techniques for the heuristic solution of the job shop problem. The developed constructive algorithm yields better results than the majority of the usual priority dispatching rules for generating an active schedule. In the iterative phase, the principle of path search developed by Werner [21] has been applied. This turns out to be an alternative way to simulated annealing or tabu search to overcome difficulties connected with local search. However, it requires some theoretical insight into the structure of the problem to make the path search efficient. Especially it is necessary to decide which are "suitable" neighbours for the path generation. In the case of classical shop problems in the field of machine scheduling the consideration of the combinatorial structure of feasible solutions turned out to be a suitable approach.

The derived path algorithm would also be appropriate for the implementation on a parallel computer. Instead of generating successively the paths, this could be done in parallel. Thus, first we generate the neighbours in the shift graph, then the schedules with a shortest length of 2 from the initial schedule LR^0 and so on. In such an implementation a schedule with a shortest length k from the LR^0 can only be accepted if no schedule with a shorter length from LR^0 had been accepted.

In the paper we considered the case that each job has to be processed on each machine at most once. However, all the ideas presented in this paper can also be applied to the more general case that a job has to be processed more than once on a machine. In this case we have to replace the two-dimensional representation of an operation by a three-dimensional one.

Finally we only mention that a similar constructive insertion algorithm has been derived for the open shop problem which also yields excellent results [3].

References

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

[2] H. Bräsel, Lateinische Rechtecke und Maschinenbelegung, Dissertation B, TU Magdeburg (1990).

[3] H. Bräsel, T. Tautenhahn and F. Werner, Constructive heuristic algorithms for the open shop problem, TU Magdeburg, preprint (1991).

[4] H. Bräsel and F. Werner, The job-shop problem – modelling by latin rectangles, exact and heuristic solution, in: H.J. Sebastian and K. Tammer, eds., System Modelling and Optimization, Proceedings of the 14th IFIP Conference Leipzig (1989).

[5] P. Brucker, B. Jurisch and B. Sievers, A fast branch and bound algorithm for the job-shop problem, Universität Osnabrück, preprint (1990).

[6] J. Carlier and E. Pinson, An algorithm for solving the job-shop problem, Management Sci. 35 (1989) 164–176.

[7] M. Florian, P. Trepant and G. Mc Mahon, An implicit enumeration algorithm for the machine sequencing problem, Management Sci. 17 (1971) B782–B792.

[8] F. Glover, Tabu search – part I, ORSA J. Comput. 1/3 (1989) 190–206.

[9] J. Grabowski, A new algorithm of solving the flow-shop problem, in: G. Feichtinger and P. Kall, eds., Operations Research in Progress (1982) 57–75.
[10] R. Haupt, A survey of priority rule-based scheduling, OR Spektrum 11 (1989) 3–16.
[11] P.J.M. Van Laarhoven and E.H.L. Aarts, Simulated Annealing: Theory and Applications (Reidel, Dordrecht, 1987).
[12] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, Report BS-R8909, Department of Operations Research, Statistics and System Theory (1989).
[13] J.F. Muth and G.L. Thompson, Industrial Scheduling (Prentice Hall, Englewood Cliffs, NJ, 1963).
[14] M. Nawaz, E.E. Enscore and I. Ham, A heuristic algorithm for the m-machine, n-job flow shop sequencing problem, OMEGA 11 (1983) 91–95.
[15] P.S. Ow and T.E. Morton, The single machine early/tardy problem, Management Sci. 35 (1989) 177–191.
[16] D.J. Rosenkrantz, R.E. Stearns and P.M. Lewis, An analysis of several heuristics for the traveling salesman problem, SIAM J. Comput. 6 (1977) 563–581.
[17] A.S. Spachis and J.R. King, Job-shop scheduling heuristics with local neighbourhood search, Internat. J. Prod. Res. 17 (1979) 507–526.
[18] F. Werner, Zu einigen Nachbarschaftsstrukturen für Iterationsverfahren zur näherungsweisen Lösung spezieller Reihenfolgeprobleme, Wiss. Z. Tech. Univ. Magdeburg 31 (5) (1987) 48–54.
[19] F. Werner, Zu einigen Nachbarschaftsstrukturen für Iterationsverfahren zur näherungsweisen Lösung spezieller Reihenfolgeprobleme, Optimization 19 (1988) 539–556.
[20] F. Werner, Iterative heuristics for the job-shop problem, TU Magdeburg (1988).
[21] F. Werner, Zur Struktur und näherungsweisen Lösung ausgewählter kombinatorischer Optimierungsprobleme, Dissertation B, TU Magdeburg (1989).