Heuristic Algorithms for Open Shop Scheduling to Minimize Mean Flow Time, Part I: Constructive Algorithms

Article · September 2005

CITATIONS
11

READS
163

6 authors, including:

Jan Tusch
None
13 PUBLICATIONS **111** CITATIONS
SEE PROFILE

Frank Werner
Otto-von-Guericke-Universität Magdeburg
397 PUBLICATIONS **4,013** CITATIONS
SEE PROFILE

Heuristic Algorithms for Open Shop Scheduling to Minimize Mean Flow Time, Part I: Constructive Algorithms

Heidemarie Bräsel, André Herms, Marc Mörig, Thomas Tautenhahn,
Jan Tusch, Frank Werner

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

Abstract: In this paper, we consider the problem of scheduling n jobs on m machines in an open shop environment so that the sum of completion times or mean flow time becomes minimal. For this strongly NP-hard problem, we develop and discuss different constructive heuristic algorithms. Extensive computational results are presented for problems with up to 50 jobs and 50 machines, respectively. The quality of the solutions is estimated by a lower bound for the corresponding preemptive open shop problem and by an alternative estimation of mean flow time. We observe that the recommendation of an appropriate constructive algorithm strongly depends on the ratio n/m. In part II of this paper, several iterative algorithms are discussed and compared.

Key words: Open shop scheduling, mean flow time, constructive heuristics

1 Introduction

In an open shop scheduling problem, a set of n jobs $J_1, J_2, \ldots, J_n$ has to be processed on a set of m machines $M_1, M_2, \ldots, M_m$. The processing of a job on a machine is denoted as an operation, and the sequence in which the operations of a job are processed on the machines is immaterial. It is assumed that the processing times of all operations are given. As usual, each machine can process at most one job at a time and each job can be processed on at most one machine at a time.

Open shop scheduling problems arise in several industrial situations. For example, consider a large aircraft garage with specialized work-centers. An airplane may require repairs on its engine and electrical circuit system. These two tasks may be carried out in any order but it is not possible to do these tasks on the same plane simultaneously. Other applications of open shop scheduling problems in automobile repair, quality control centers, semiconductor manufacturing, teacher-class assignments, examination scheduling, and satellite communications are described by Kubiak et al. [12], Liu and Bulfin [18] and Prins [19].

Most papers in the literature dealt with the minimization of makespan. Gonzalez and Sahni [8] presented an $O(n)$ algorithm for the two-machine open shop problem denoted as $O2||C_{max}$. The preemptive problem can be solved in polynomial time for an arbitrary number of machines [8]. However, slight generalizations of the two-machine nonpreemptive problem lead already to NP-hard problems. Achugbue and Chin [1] proved that the two-machine open shop problem with minimizing the sum of completion times (also denoted as mean or total flow time minimization) is already NP-hard in the strong sense. Even for the corresponding problem with allowed preemptions, the two-machine problem is NP-hard in the ordinary sense [7] and the three-machine problem is NP-hard in the strong sense [17].

In view of the NP-hardness of problem $O||C_{max}$, heuristic and branch and bound algorithms have been developed for the case of minimizing the makespan. Bräsel et al. [4] developed several constructive heuristics based on matching algorithms (which determine subsets of operations that can run simultaneously) as well as on the insertion of operations into partial schedules combined with beam search. Using the randomly generated instances given by Taillard [22], they found that for

most of the larger instances with $n = m$, the insertion algorithms combined with beam search often reached the trivial lower bound for an instance given by the maximum of the sum of processing times of the operations of a job and the machine loads. This stimulated some investigations in generating harder benchmark problems. BRUCKER et al. [5] generated a set of more difficult problems, for which the optimum value is greater than the lower bound mentioned above. The general idea was to construct processing time matrices in which all row and column sums are equal to the same value. The presented branch and bound algorithm was able to solve most benchmark instances with up to 7 jobs and 7 machines to optimality. GUERET et al. [9] improved the latter procedure by using intelligent backtracking. They can solve all instances with 7 jobs and 7 machines and most problems with 10 jobs and 10 machines given by TAILLARD [22] to optimality. The currently best exact algorithm is that given by DORNDORF et al. [6]. Instead of analyzing and improving the search strategies for finding solutions, the authors focus on constraint propagation methods for reducing the search space. Several benchmark problem instances are solved to optimality for the first time. In particular, among the problems given by TAILLARD [22], all instances with 10 jobs and 10 machines have been solved with an average running time of less than one minute. Moreover, all but one instances with 15 jobs and 15 machines and 7 instances with 20 jobs and 20 machines have been solved. Except for the unsolved instances, the running time is always less than 12 minutes for the 15×15 instances and about one hour for the 20×20 instances.

GUERET and PRINS [10] presented and compared several heuristics for the open shop makespan minimization problem. Several versions of list scheduling algorithms as well as matching algorithms have been proposed. The sequence of computed subsets by the matching algorithm gave a preliminary schedule which can be improved by a packing procedure. Local search algorithms are presented using two types of neighborhoods in which either a matching is moved to another position or two matchings are interchanged. Computational results have been presented for problems with up to 20 jobs and 20 machines.

LIAW [14] presented an iterative approach based on Bender's decomposition which separates the sequencing and scheduling components of the problem allowing each to be treated individually. The scheduling component is the dual of a longest path problem which can be efficiently solved by the label correcting algorithm. The sequencing component has led to a heuristic sequence improvement procedure. Computational experience has shown that most solutions of randomly generated problems are within 1 % deviation from the lower bound.

A tabu search algorithm for the open shop problem with minimizing makespan has been given by ALCAIDE et al. [2]. List scheduling algorithms have been used for constructing a starting solution, and the tabu search algorithm has extensively been tested on randomly generated instances. LIAW [15] presented a hybrid genetic algorithm for solving the open shop problem with makespan minimization approximately. The hybrid algorithm incorporated a local improvement procedure based on tabu search into a basic genetic algorithm. This enabled the algorithm to perform genetic search over the subspace of local optima. The algorithm has been tested both on randomly generated as well as benchmark problems of the literature with up to 20 jobs and 20 machines and often reached an objective function value close to the optimum.

Recently also the investigation of multicriteria open shop scheduling problems has begun. KYPARISIS and KOULAMAS [13] described polynomially solvable special cases of the two- and three-machine open shop hierarchical criteria scheduling problems to find a schedule with minimum mean flow time subject to minimum makespan, and they proposed a heuristic algorithm to solve the problem with a dominant machine. GUPTA et al. [11] derived and compared several heuristic algorithms for the hierarchical criteria two-machine problems, where the primary objective was the minimization of makespan and the secondary criterion was one of minimizing total completion time, total weighted completion time and total weighted tardiness. The constructive and iterative algorithms developed have been evaluated on problem instances with up to 80 jobs. Moreover, two polynomially solvable special cases

have been given.

While there are some papers on multicriteria scheduling problems where one objective is the minimization of mean flow time, there exist only a few papers dealing exclusively with the minimization of mean flow time although from a practical point of view, often the minimization of the latter criterion is more important. Liaw et al. [16] considered the problem of minimizing total completion time with a given sequence of jobs on one machine (also denoted as $O|GS(1)|\sum C_i$). This problem is already NP-hard in the strong sense even in the case of two machines. First, a lower bound has been derived based on the optimal solution of a relaxed problem in which the operations on every machine may overlap except for the machine with a given sequence of jobs. Although the relaxed problem is NP-hard in the ordinary sense, it can nevertheless be rather quickly solved via a decomposition into subset-sum problems. Then a branch and bound algorithm has been presented and tested on square problems with $n = m$. The algorithm was able to solve all problems with 6 jobs in 15 minutes on average and most problems with 7 jobs within a time limit of 50 hours with an average computation time of about 15 hours for the solved problems. A heuristic algorithm has also been proposed which is an iterative dispatching procedure. This procedure consists of two major components: a one-pass heuristic generating a complete schedule at each iteration, and an adjustment strategy to adjust the parameter used at each iteration. This procedure has been tested on square problems with up to 30 jobs and 30 machines. For the small problems with at most 7 jobs, the average percentage deviation from the optimal value is about 4 % while for larger problems, the average percentage deviation from the lower bound is about 8 %. For the preemptive problem with an arbitrary number of machines and minimizing total completion time, Bräsel and Hennes [3] derived lower bounds and heuristics which have been tested on problems with up to 50 jobs and 50 machines. For problems with a small number of jobs, the results with the heuristics have been compared to the optimal solutions found by an exact algorithm.

Concerning approximation algorithms with performance guarantee, the currently best result has been given by Queyranne and Sviridenko [20, 21]. They presented a 5.83-approximation algorithm for the nonpreemptive open shop problem with minimization of weighted mean flow time which is based on linear programming relaxations in the operation completion times. This is used to generate precedence constraints. For the preemptive version of this problem, it is shown that a simple job-based greedy algorithm using directly the LP relaxation yields a 3-approximation algorithm.

In this paper, we consider the non-preemptive open shop scheduling problem with minimizing mean flow time. The remainder of the paper is organized as follows. In Section 2, we introduce some basic notions and we derive estimations for the optimal objective function value. In Section 3 we describe several constructive algorithms. A detailed computational comparison of the algorithms is discussed in Section 4. Section 5 contains some conclusions and summarizing recommendations.

2 Basic Notions and Evaluation of Schedules

In the following, we use the digraph $G(MO, JO)$ with operations as vertices and arcs between two immediately succeeding operations of a job or on a machine. If we place the operations of job J_i in the i-th row and the operations on machine M_j into the j-th column, then $G(MO, JO) = G(MO) \cup G(JO)$ where $G(MO)$ contains only horizontal arcs (describing the machine order of the jobs) and $G(JO)$ contains only vertical arcs (describing the job orders on the machines).

Figure 1 shows $G(MO)$, $G(JO)$ and $G(MO, JO)$, if the machine orders of the jobs are given by

$$J_1 : M_3 \to M_1; \qquad J_2 : M_1 \to M_3 \to M_2; \qquad J_3 : M_2 \to M_3 \to M_1$$

and, moreover, the job orders on the machines are as follows:

$$M_1 : J_2 \to J_1 \to J_3; \qquad M_2 : J_3 \to J_2; \qquad M_3 : J_1 \to J_3 \to J_2.$$

A combination of machine orders and job orders (MO, JO) is feasible, if $G(MO, JO)$ is acyclic. An acyclic digraph $G(MO, JO)$ is called *sequence graph*. In this case all above graphs represent partial

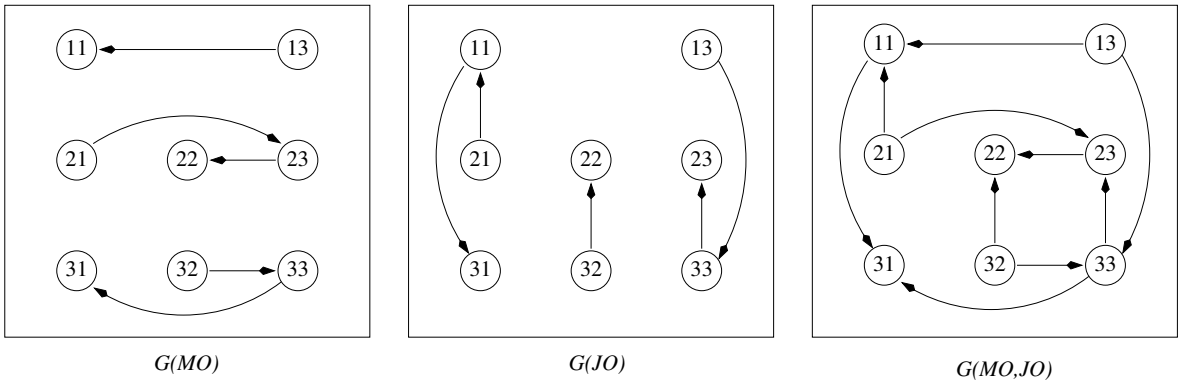


Figure 1: $G(MO)$, $G(JO)$ and $G(MO, JO)$

orders on the set of operations. Similarly as in [4, 23], we describe a sequence graph $G(MO, JO)$ by its *rank matrix* $A = (a_{ij})$, i.e., the entry $a_{ij} = l$ means that a path to operation (i, j) with maximal number of operations has l operations. Notice that for each $a_{ij} = k > 1$, integer $k - 1$ occurs as entry in row i or column j (or both). Now we assign the weight t_{ij} to operation (i, j) in $G(MO, JO)$. The computation of a longest path to the vertex (i, j) with (i, j) included in an acyclic digraph $G(MO, JO)$ yields the completion time c_{ij} of operation (i, j) in the semiactive schedule $C = (c_{ij})$. In the example considered, we obtain the following schedule C from the given matrix of processing times $T = (t_{ij})$ and the rank matrix $A = (a_{ij})$ corresponding to $G(MO, JO)$ in Figure 1, where $\sum C_i = 9 + 13 + 14 = 36$ holds:

$$T = \begin{pmatrix} 4 & . & 5 \\ 2 & 3 & 3 \\ 5 & 1 & 2 \end{pmatrix}, \qquad A = \begin{pmatrix} 2 & . & 1 \\ 1 & 4 & 3 \\ 3 & 1 & 2 \end{pmatrix}, \qquad C = \begin{pmatrix} 9 & . & 5 \\ 2 & 13 & 10 \\ 14 & 1 & 7 \end{pmatrix}. \tag{1}$$

The head r_{ij} of an operation (i, j) is defined as its earliest possible starting time according to $G(MO, JO)$, i.e., as the longest path to vertex (i, j) in this graph with vertex (i, j) not included.

In [3], Bräsel and Hennes generalized the model to the preemptive case and derived a lower bound for the open shop problem $O|pmtn|\sum C_i$ which can also be taken for the nonpreemptive problem. This lower bound is as follows. Let

$$T_i = \sum_{j=1}^{m} t_{ij} \qquad \text{and} \qquad \overline{T}_j = \sum_{i=1}^{n} t_{ij}$$

and suppose that jobs and machines are ordered such that

$$T_1 \leq T_2 \leq \ldots \leq T_n \qquad \text{and} \qquad \overline{T}_1 \leq \overline{T}_2 \leq \ldots \leq \overline{T}_m.$$

Obviously, we have $C_i \geq T_i$ in a feasible schedule and any unavoidable waiting time of job J_i increases the completion time C_i. The goal is to identify such unavoidable waiting times of jobs. We can assume that $n = m$ (otherwise we introduce dummy jobs or machines). First, let $T_1 > \overline{T}_1$, i.e., the total processing time of job J_1 is greater than the machine load on M_1 and let $h_1 := T_1 - \overline{T}_1$. Considering the intervals in which job J_1 is processed, there are subintervals of total length h_1 with the following property: at each time inside these subintervals, there can be at most $n - 2$ jobs processed simultaneously with J_1 since one machine different from M_1 is occupied by job J_1 and machine M_1 is idle. Thus, one (several) job(s) with

$$T_j < \overline{T}_j \tag{2}$$

must wait until value T_j has been increased to $\overline{T}_j$. Therefore, one chooses the longest job J_k with $T_k < \overline{T}_k$ and performs the following computations:

$$H := \min\{T_k + h_1, \overline{T}_k\}; \quad h_1 := h_1 - H + T_j; \quad T_j := H.$$

This procedure is repeated with other jobs satisfying inequality (2) until $h_1 = 0$. If $T_1 < \overline{T}_1$, then there is an unavoidable idle time on machine M_1, and the rest is symmetric to the first case. Then jobs $J_2, J_3, \ldots, J_n$ are considered in a similar way and finally, the lower bound for the preemptive problem is equal to sum of the final T_i values (see also [3]).

In addition, we use the following measure for evaluating schedules generated by the subsequent procedures. We transform the given problem into a problem with constant processing times

$$\bar{t} := \bar{t}_{ij} = \frac{T}{n \cdot m},$$

where

$$T = \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij}.$$

The modified problem can be polynomially solved by applying an algorithm for solving problem $O|t_{ij} = 1| \sum C_i$. This algorithm forms blocks of m jobs subsequently performed on the machines, i.e., each block requires $m\bar{t}$ time units in which the m jobs are feasibly processed. In the following, we use the optimal function value $\overline{C}$ of the modified problem for estimating the optimal function value of the original problem. It is worth noting that $\overline{C}$ is not a lower bound for the optimal objective function value of the original problem.

3 Constructive Algorithms

Among constructive algorithms we consider matching algorithms, priority dispatching rules (generation of active and nondelay schedules), insertion and appending procedures combined with beam search. In the following, we describe these algorithms more in detail.

(a) Matching algorithms

The first type of algorithms is based on matching procedures. They have been suggested by Bräsel et al. [4] for the makespan minimization problem and generate so-called *rank-minimal schedules*. Without loss of generality, we assume that we have a square problem with $n = m$ (otherwise we introduce dummy jobs or dummy machines such that $n = m$). The algorithm successively determines n operations having rank 1 in the graph $G(MO, JO)$, n operations having rank 2, and so on. This is done by solving weighted bipartite maximum cardinality matching problems. Let G^* be the complete bipartite graph with the set of jobs and machines, respectively, as vertices. To each edge (i, j) of G^*, we assign the weight t_{ij}. Then the basic algorithm is as follows.

Algorithm MATCHING

for $k := 1$ **to** n **do**
begin
1. determine a perfect matching M on G^* with an appropriate objective function and assign to all operations $(i, j) \in M$ rank k;
2. delete all edges (i, j) from G^*;
end

It remains to describe appropriate objective functions to determine perfect matchings M. For instance, among all perfect matchings, choose one with

(1)	$\sum_{(i,j)\in M} t_{ij}$	$\rightarrow$	min!
(2)	$\sum_{(i,j)\in M} t_{ij}$	$\rightarrow$	max!
(3)	$\sum_{(i,j)\in M} \{t_{ij} + r_{ij}\}$	$\rightarrow$	min!
(4)	$\max_{(i,j)\in M} t_{ij}$	$\rightarrow$	min!
(5)	$\min_{(i,j)\in M} t_{ij}$	$\rightarrow$	max!
(6)	$\max_{(i,j)\in M} \{t_{ij} + r_{ij}\}$	$\rightarrow$	min!

where r_{ij} denotes the head of operation (i, j). While the first three variants use a sum objective function, the other ones use a bottleneck function. The algorithms underlying criteria (1) - (2) and (4) - (5) are static heuristics. The main idea behind them is to choose operations with approximately the same processing time in each step. The algorithms underlying criteria (3) and (6) are dynamic heuristics, where in each step a matching is chosen which minimizes $\sum C_i$ and C_{max}, respectively, for the resulting partial solution.

(b) Generation of active and nondelay schedules

A schedule is called *active* if no operation can be started earlier without delaying some other operation. A schedule is called *nondelay* if no machine is left idle provided that it is possible to process some job. Obviously, any nondelay schedule is an active schedule.

The algorithms for constructing active and nondelay schedules repeatedly append operations to a partial schedule. Starting with an empty schedule (which is obviously a nondelay one), operations are appended as follows:

- To construct a nondelay schedule, we determine the minimal head r of all unscheduled operations. At time r, there exist both a free machine and an available job. To maintain the nondelay property of the schedule, we have to append an operation which can start at time r. Among all operations (i, j) with $r_{ij} = r$, choose one according to some priority dispatching rule.
- To construct an active schedule, we determine the minimal possible completion time EC of all unscheduled operations with respect to their heads and processing times. The next operation to append is chosen among all unscheduled operations which can start before EC. Again, that choice is made by a priority dispatching rule.

In our tests, we have used the following priority dispatching rules:

- $FCFS$ - first come first served;
- ECT - earliest completion time;
- SPT - shortest processing time;
- LPT - longest processing time.

We note that in the case of generating a nondelay schedule, the application of rules ECT and SPT leads to the same schedule.

(c) Insertion algorithms combined with beam search

Insertion algorithms combined with beam search have been suggested in connection with shop scheduling problems, e.g. by BRÄSEL et al. [4] for the open shop problem and by WERNER and WINKLER [23] for the job shop problem. They are restricted branch and bound procedures which are based on an enumerative algorithm for schedules. The insertion algorithm works as follows. According to some chosen insertion order, operations are successively inserted into partial sequence graphs. When a new operation is inserted, all precedence constraints previously established are maintained. Let v_k operations of job J_k and u_l operations on machine M_l be already sequenced. Then there exist $(v_k + 1) \cdot (u_l + 1)$ possibilities to insert operation (k, l), namely operation (k, l) can be inserted on the i-th position in the machine order of job J_k, $1 \leq i \leq v_k + 1$ and on the j-th position in the job order on machine M_l, $1 \leq j \leq u_l + 1$ (however, not every combination of positions leads to a sequence graph). For the following three cases, it is clear that the resulting graph is acyclic:

I1: Let $a^*_{kl} = 1$, i.e., choose M_l as the first machine in the machine order of J_k and J_k as the first job in the job order of M_l.

I2: Let the rank $a^*_{kl} = b + 1$, where entry b occurs in the k-th row or in the l-column of the rank matrix A, i.e., operation (k, l) becomes a direct successor of one of the operations of job J_k or on machine M_l.

I3: Let there exist a pair of operations (k, w) and (v, l) such that there does not exist any path between them in the current graph $G(MO, JO)$. If $a_{kw} \leq a_{vl}$, we set $a^*_{kl} = a_{vl} + 1$ and $a^*_{kw} = a_{vl} + 2$, i.e., operation (k, l) is chosen as the direct successor of operation (v, l) and as the direct predecessor of (k, w). If $a_{kw} \geq a_{vl}$, we set $a^*_{kl} = a_{kw} + 1$ and $a^*_{vl} = a_{kw} + 2$, i.e., operation (k, l) is chosen as the direct successor of operation (k, w) and as the direct predecessor of (v, l). Notice that, if $a_{kw} = a_{vl}$, both cases yield a new rank matrix.

In all cases above, the ranks of all successors of the inserted operation have to be updated. It can be easily proved that only the above cases I1 - I3 lead to sequence graphs when inserting operation (k, l).

Consider rank matrix A according to (1) and let us insert operation $(1, 2)$. According to I1, we can generate the rank matrix

$$A^1 = \begin{pmatrix} 3 & \mathbf{1} & 2 \\ 1 & 5 & 4 \\ 4 & 2 & 3 \end{pmatrix}.$$

According to I2, we can generate three rank matrices (namely using $b \in \{1, 2, 4\}$):

$$A^2 = \begin{pmatrix} 3 & \mathbf{2} & 1 \\ 1 & 4 & 3 \\ 4 & 1 & 2 \end{pmatrix}, \qquad A^3 = \begin{pmatrix} 2 & \mathbf{3} & 1 \\ 1 & 4 & 3 \\ 3 & 1 & 2 \end{pmatrix}, \qquad A^4 = \begin{pmatrix} 2 & \mathbf{5} & 1 \\ 1 & 4 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

According to I3, we can generate the following rank matrices:

$$A^5 = \begin{pmatrix} \mathbf{2} & \mathbf{3} & 1 \\ 1 & 7 & 6 \\ 6 & \mathbf{4} & 5 \end{pmatrix}, \qquad A^6 = \begin{pmatrix} \mathbf{6} & \mathbf{5} & 1 \\ 1 & \mathbf{4} & 3 \\ 7 & 1 & 2 \end{pmatrix}, \qquad A^7 = \begin{pmatrix} 3 & \mathbf{2} & \mathbf{1} \\ 1 & 6 & 5 \\ 5 & \mathbf{3} & 4 \end{pmatrix}, \qquad A^8 = \begin{pmatrix} 4 & \mathbf{2} & \mathbf{3} \\ 1 & 6 & 5 \\ 5 & \mathbf{1} & 4 \end{pmatrix}$$

(notice that the corresponding elements a^*_{kl}, a^*_{kw} and a^*_{vl} are drawn in bold face in $A^5 - A^8$).

The following algorithm Beam-Insert is determined by fixing the insertion order IO, the estimation LB for the objective function value of a partial schedule, the beam width k and the son selection procedure SEL, i.e., the partial sequence graphs that are selected from the set of generated graphs for

further considerations. Typically, the insertion order IO is determined by some well-known priority rule (see generation of active and nondelay schedules). In our tests, we have considered the rules $RANDOM, SPT$, LPT and $FCFS$. Concerning the estimation of a partial schedule, this is usually done by means of a rough lower bound for the objective function value of an arbitrary completion of the current partial sequence graph and schedule, respectively. Here we always use the following bound:

LBP: For a partial schedule, the value $\sum C_i$ is estimated by the sum of the completion times of the currently last scheduled operation of each job.

It should be noted that in an insertion algorithm, the unscheduled operations cannot be used for strengthening the estimation LBP.

The beam width k denotes the number of paths that are considered in the branching tree. To select k partial sequence graphs, we have considered the following two variants for the son selection procedure SEL:

SEL1: Select in each step the k best sons from the whole set of partial sequence graphs, i.e., the selected sons do not necessarily have different fathers.

SEL2: For each of the k fathers select the best son, i.e., all sons have different fathers (as long as we do not have k fathers the first criterion is applied).

Algorithm Beam-Insert is as follows:

Algorithm BEAM-INSERT

1. fix the insertion order $IO = (i_1, j_1); (i_2, j_2); \ldots; (i_{mn}, j_{mn})$, the estimation LB for the objective function value, the beam width k, and the son selection procedure SEL;
2. let $SR = \{A\}$, where A is the empty sequence graph;
   for $v := 1$ **to** mn **do**
   begin
3. $S^* := \emptyset$;
   for all $A \in SR$ **do**
   begin
4. generate the set S_A of sons of A by inserting operation (i_v, j_v) into A according to I1 up to I3;
5. $S^* := S^* \cup S_A$;
   end
6. $k^* := \min\{k, |S^*|\}$;
7. prune the search tree by removing from S^* all but the k^* best (according to LB) sequence graphs complying with son selection procedure SEL;
8. $SR := S^*$;
   end
9. select the schedule with smallest mean flow time value as the generated heuristic solution.

We emphasize that the above insertion algorithm is a generalization of that suggested in [4] since here all possible cases for inserting a selected operation are considered.

(d) Appending procedures combined with beam search

In some initial tests we have found that for certain types of problems, even very simple nondelay schedules produced better results than the Beam-Insert algorithm. This leads to the idea to combine the generation of nondelay schedules with a new beam search strategy. This means that in each step, a partial sequence graph is extended by appending some operation. For selecting this operation, the smallest possible head of some unscheduled operation is determined. In the case when this operation is not uniquely determined (which is in particular the case at the beginning of the procedure), we apply some tie-breaking rule TBR to select an operation (i, j). We take as tie-breaking rule a specific priority dispatching rule. In our tests, we used $TBR \in \{RANDOM, SPT, LPT, FCFS\}$.

In order to generate several successors from a partial sequence graph, we considered two son generation procedures SG for generating the successors of the current partial sequence graph(s), namely a machine-oriented ($SG = MO$) and a job-oriented ($SG = JO$) appending of the next operation. This means that, considering the selected operation (i, j), either an unscheduled operation on machine M_j or an unscheduled operation of job J_i is sequenced such that a nondelay schedule results. From a partial sequence graph, there can be generated up to n and m sequence graphs for $SG = MO$ and $SG = JO$, respectively.

The following algorithm Beam-Append is characterized by the tie-breaking rule TBR, the son generation procedure SG, the estimation LB of the partial schedule, the beam width k, and the son selection type SEL. Parameters k and SEL are similar to the corresponding parameters for procedure Beam-Insert. For the evaluation of the generated sons, we have considered the following estimations for the objective function value:

LBP: For a partial schedule, the value $\sum C_i$ is estimated by the sum of the completion times of the currently last scheduled operation of each job (see also Beam-Insert).

LBC: For the Beam-Append procedure, it is known that all unscheduled operations of a job must be sequenced later than the currently last scheduled operation of this job. Therefore, we estimate the completion time C_i of job J_i by the largest completion time of the scheduled operations of this job plus the sum of the processing times of the unscheduled operations of job J_i.

Let $R_i, 1 \leq i \leq n$, be the sum of the processing times of all currently unscheduled operations of job J_i. Then

$$LBC = LBP + \sum_{i=1}^{n} R_i,$$

i.e. LBC is a stronger lower bound.

Algorithm Beam-Append is as follows.

Algorithm BEAM-APPEND

1. fix the tie-breaking rule TBR, the son generation procedure SG, the estimation LB for the objective function value, the beam width k, and the son selection procedure SEL;
2. let $SR = \{A\}$, where A is the empty sequence graph;
   for $v := 1$ **to** mn **do**
   begin
3. $S^* := \emptyset$;
   for all $A \in SR$ **do**
   begin
4. determine operation (i, j) with the smallest head r_{ij} among

all unscheduled operations in A, if this is not uniquely determined, apply the tie-breaking rule TBR to determine operation (i, j);

5. generate the set S_A of sons by appending to A all possible operations on machine M_j (if $SG = MO$) or of job J_i (if $SG = JO$) such that a nondelay schedule results;

6. $S^* := S^* \cup S_A$;

end

7. $k^* := \min\{k, |S^*|\}$;

8. prune the search tree by removing from S^* all but the k^* best (according to LB) sequence graphs complying with son selection procedure SEL;

9. $SR := S^*$;

end

10. select the schedule with smallest mean flow time value as the generated heuristic solution.

4 Computational Results

In this section, we present computational results for the constructive algorithms discussed in Section 3.

4.1 Initial Tests for Constructive Algorithms

In initial investigations we have found that the choice of an appropriate algorithm strongly depends on the structure of the problem. For instance, square problems with $n = m$ show a rather different behavior than problems in which the number of jobs is substantially larger than the number of machines or vice versa.

We have tested the constructive algorithms first on the six following problem types:

n	30	30	30	20	20	10
m	10	20	30	20	30	30

For all problem types, we considered 25 instances with short processing times (type S: processing times are integers randomly taken from the interval $[1, 20]$) and 25 instances with long processing times (type L: processing times are integers randomly taken from the interval $[1, 100]$). We have used the following algorithms:

- **Active/Nondelay**: Generation of an active and a nondelay schedule, respectively. The notation $ND(PDR)$ denotes the generation of a nondelay schedule according to priority dispatching rule $PDR \in \{SPT, LPT, FCFS\}$. Analogously, the notation $A(PDR)$ denotes the generation of an active schedule according to priority dispatching rule PDR.

- **Beam-Insert**: Application of the insertion algorithm combined with beam search. The notation $BI(IO, LB, k, SEL)$ denotes the Beam-Insert procedure with insertion order IO, the estimation LB of a partial schedule, beam width k and son selection procedure SEL.

- **Beam-Append**: Application of the appending procedure combined with beam search. The notation $BA(TBR, SG, LB, k, SEL)$ denotes the Beam-Append procedure with the tie-breaking rule TBR, the son generation procedure SG, the estimation LB of a partial schedule, beam width k and son selection procedure SEL.

Before evaluating the results, we give a few comments on the computation times for the particular algorithms on an AMD Athlon XP 1800+. For the large problems with $n = m = 30$, the average computation times for Beam-Append are less than 1 s for $k = 1$, between 2 and 3 s for $k = 3$ and between 4 and 5 s for $k = 5$ in dependence on the used bound LB (for LBC times are slightly larger than for LBP). For Beam-Insert and the large problems with $n = m = 30$, the average computation times are approximately linear in k, namely between 60 and 75 s for $k = 1$, between 180 and 230 s for $k = 3$ and between 300 and 380 s for $k = 5$. For smaller problems, e.g. $n = 20, m = 20$, these times are 6 s for $k = 1$, between 16 and 20 s for $k = 3$ and between 28 and 32 s for $k = 5$. We have obtained the following results.

(a) General observations:

First, we observed that the processing times did not have a substantial influence on the results, i.e., for both types S and L of problems we have obtained comparable conclusions. However, the problem size, i.e., the relationship between the values of n and m, turned out to have a substantial influence on the recommendation of an appropriate algorithm.

Among the matching algorithms, criterion (3) gave the best results for all square problems with $n = m$. For nonsquare problems, criteria (3) and (6) delivered the best results each in approximately half of the instances. However, all variants of the matching algorithm work bad (usually 15 - 20 % worse than an appropriate Beam-Append variant) so that they are not considered in further tests.

Considering the son generation procedure in Beam-Append, we found that for nonsquare problems (i.e., n and m are different) a job-oriented appending is substantially preferable while for square problems, both a job and a machine-oriented procedure worked with a similar quality. Therefore, we decided to apply exclusively a job-oriented appending in all further tests, i.e., $SG = JO$.

(b) Specific Observations:

In the following, we discuss the results separately for the problems with $n > m$, $n = m$ and $n < m$:

Problems with $\mathbf{n > m}$, i.e., $n = 30,\ m = 20$ and $n = 30,\ m = 10$:

For problems with $m = 20$, we have observed that among active and nondelay schedules, variant $ND(FCFS)$ is clearly the best. Moreover, the Beam-Append procedure is clearly superior to the Beam-Insert procedure. The recommended tie-breaking rule is $FCFS$ which is clearly superior to all other tie-breaking rules. Son selection procedures $SEL1$ and $SEL2$ yield comparable results. Concerning the used lower bound, both variants LBP and LPC yield results of the same quality and should be considered further: for problems of type L, the best value has been obtained 11 times for LBC and 14 times for LBP while for problems of type S, the best value has been obtained 12 times for variant LBC and 13 times for variant LBP.

For the problems with $m = 10$ and priority dispatching rules, all best values have been obtained for the $FCFS$ rule, where both the generation of a nondelay and an active schedule have to be considered (for the majority of problems, nondelay schedules are superior, however, also some active schedules contribute with best values). For the Beam-Append procedure, recommendations are identical to the case $m = 20$ (e.g. the best value has been obtained 10 times for variant LBC and 15 times for variant LBP).

Problems with $\mathbf{n = m}$, i.e., $n = 20,\ m = 20$ and $n = 30,\ m = 30$:

First, we can emphasize that results for both problem sizes show a very similar behavior with a few exceptions. For the problems of type S with $n = m = 20$, among active and nondelay schedules the generation of a nondelay schedule by variant $FCFS$ is superior to the other rules, however, the generation of a nondelay schedule by the SPT/ECT rule also leads sometimes to the best value. For

problems of type L, nondelay schedules according to the rules $FCFS$ and SPT/ECT are comparable, but also a nondelay schedule according to the LPT rule leads for three instances to the best value.
Concerning Beam-Append, we have found that both variants LBP and LBC deliver schedules of approximately comparable quality (bound LBC is slightly preferable), and also no particular tie-breaking rule is superior to the others.
Concerning Beam-Insert, for all problems insertion order LPT has obtained the best results while both son selection procedures $SEL1$ and $SEL2$ and all three considered beam widths contribute with best values among the Beam-Insert variants. However, algorithm Beam-Insert is not competitive to Beam-Append.

For the problems with $n = m = 30$ and algorithm Beam-Append, lower bound LBC is preferable (for problems of type L, the best solution was obtained 18 times by LBC while only 8 times by LBP and concerning the beam width, only for $k \in \{3, 5\}$). All tie-breaking rules contribute with best values, however SPT is preferable (again for the problems of type L, the best solutions was 16 times obtained by SPT, 4 times by $RANDOM$, 4 times by $FCFS$ and one time by LPT).

Problems with $\mathbf{n < m}$, i.e., $n = 20,\ m = 30$ and $n = 10,\ m = 30$:
First, we emphasize that for both problem sizes recommendations are very similar. For the problems with $m = 20$, among active and nondelay schedules, variant $ND(LPT)$ is clearly the best. It has obtained the best function value for all 20 instances of type L and for 16 instances of type S.
Among the Beam-Insert procedures, all best values have been obtained for insertion order LPT while both son selection procedures $SEL1$ and $SEL2$ have their merits. Typically, with an increasing value of k, the results become slightly better. However, any variant $k \in \{1, 2, 3\}$ contributed with best values.
Concerning algorithm Beam-Append, the results are considerably better with the rough bound LBP in comparison with bound LBC. Both son selection procedures $SEL1$ and $SEL2$ contribute approximately in the same amount with best values. Concerning the tie-breaking rule, $FCFS$ is the best followed by rule SPT. In contrast to Beam-Insert, tie-breaking rule LPT works not well in this case. However, Beam-Append procedures are not competitive to Beam-Insert variants and often even worse than a good nondelay schedule.
For the problems with $n = 10$, all best values with Beam-Append have been obtained for LBP while both son selection procedures $SEL1$ and $SEL2$ and all tie-breaking rules contribute with best values, and there is no superiority of one variant over the others. However, again Beam-Append has no superiority over good nondelay schedules and is worse than good Beam-Insert variants.

Therefore, we decided to use the following algorithms for the comparative study in the next subsection:

- **Problems with $\mathbf{n > m}$:**
  $ND(FCFS)$ [and for comparison purposes $A(FCFS)$];
  $BA(FCFS, JO, LB, k, SEL)$ with $LB \in \{LBP, LBC\}; k \in \{1, 3, 5\}$ and $SEL \in \{SEL1, SEL2\}$.

- **Problems with $\mathbf{n = m}$:**
  $ND(PDR)$ [and for comparison purposes $A(PDR)$] with $PDR \in \{SPT, LPT, FCFS\}$;
  $BI(LPT, LBP, k, SEL)$ with $k \in \{1, 3, 5\}$ and $SEL \in \{SEL1, SEL2\}$;
  $BA(TBR, JO, LB, k, SEL)$ with $TBR \in \{RANDOM, SPT, LPT, FCFS\}, LB \in \{LBP, LBC\}$, $k \in \{1, 3, 5\}$ and $SEL \in \{SEL1, SEL2\}$.

- **Problems with $\mathbf{n < m}$:**
  $ND(PDR)$ [and for comparison purposes $A(PDR)$] with $PDR \in \{SPT, LPT\}$;

$BI(LPT, LBP, k, SEL)$ with $k \in \{1, 3, 5\}$ and $SEL \in \{SEL1, SEL2\}$.

This means that for problems with $n > m$, we consider 11 variants of algorithms (remind that for $k = 1$, $SEL = SEL1$ and $SEL = SEL2$ yield identical results). For problems with $n = m$, we consider 51 variants (six variants of priority dispatching rules, five variants of algorithm BI and 40 variants of algorithm BA). Finally, for problems with $n < m$, we consider nine variants (four variants of priority dispatching rules and five variants of algorithm BI). Note also that recommendations for $n > m$ and $n < m$ are substantially different while for square problems with $n = m$, there is a mixture of both extremes (therefore, we also used substantially more variants for this type of problems, in particular we included also an appropriate Beam-Insert variant due to its good performance for the problems with $n < m$).

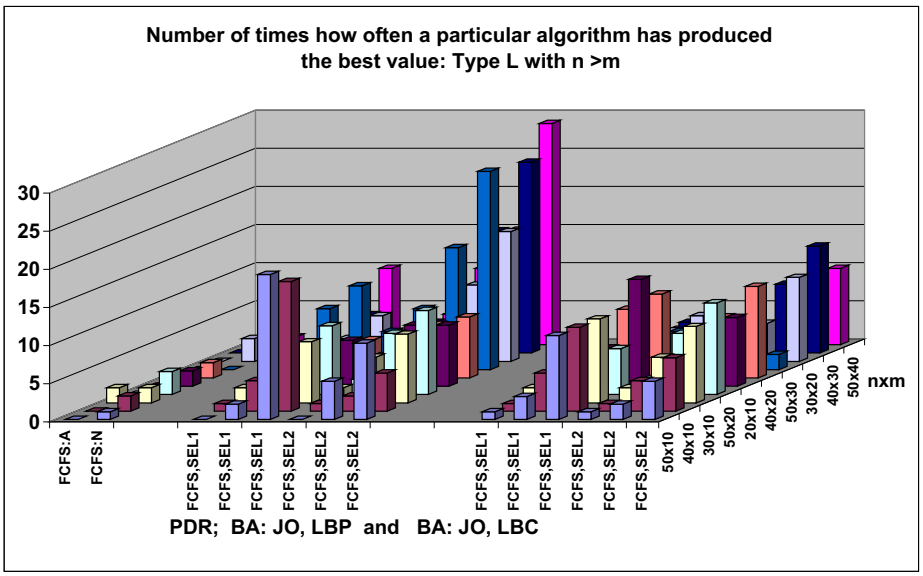
4.2 Comparative Study of Constructive Algorithms

For the comparative study, we have considered all pairs (n, m), $n \neq m$, with $n \in \{10, 20, 30, 40, 50\}$ and $m \in \{10, 20, 30, 40, 50\}$. Additionally, we have considered square problems with $n = m \in \{10, 15, 20, 25, 30, 35, 40\}$. For any pair (n, m), we have generated 50 instances with both types S and L of the processing times yielding a total of 2700 instances. We again discuss the results separately for the cases $n > m$, $n = m$ and $n < m$.

Problems with n > m:

In Table 1, we present the **a**verage **p**ercentage **d**eviation $APD_{\overline{C}}$ from the comparison value $\overline{C}$ (see Section 2) of the best values obtained by the particular types of algorithms (i.e., the third column PDR refers to the best value of the priority dispatching rules for generating active and nondelay schedules, and the fourth and fifth columns refer to the best values of all variants of Beam-Append with $LB = LBP$ and $LB = LBC$, respectively. Moreover, the last column presents the average percentage deviation $APD_{\overline{C}}$ of the best value from $\overline{C}$ obtained by some constructive algorithm. It is noted that for this type of problems, we do not refer to the preemptive lower bound since it is rather weak for these problems. In Figure 2, we present the number of times how often a particular value has obtained the best value. At the x-axis, the different procedures are marked: first priority dispatching rules, then Beam-Append with $LB = LBP$, and finally Beam-Append with $LB = LBC$. Within a particular type of Beam-Append, the left entry represents beam width $k = 1$, the middle entry $k = 3$ and the right entry $k = 5$. At the y-axis, the types of problems are marked (here they are ordered according to decreasing ratios n/m).

From Table 1 and Figure 2, we observe the following results for problems with $n > m$. The Beam-Append procedure with $k = 5$ yields clearly the best results. The results for problems of types S and L are similar. It can also be observed that for the problems with large ratio n/m (in particular $n/m \geq 2$, i.e., problems with $m = 10$ and $n = 50, m = 20$), son selection procedure $SEL1$ is superior while for the problems with small ratio $n/m \leq 2$, obviously son selection type $SEL2$ is preferable. Using these son selection procedures and comparing the two lower bounds LBP and LBC, it turns out that for most problems the bound $LB = LBP$ is superior. In particular, for almost all problems of type L, the rough bound LBP leads to better results while for problems of type S, bound LBC becomes better as the ratio of n/m increases (in particular for $n = 50$ and $m = 10$, the use of $LB = LBC$ is superior). Due to the smaller computational times, also the corresponding variants with $k = 3$ are preferable. This means that $BA(FCFS, JO, LBP, 5, SEL2)$ and $BA(FCFS, JO, LBP, 3, SEL2)$ are recommended for problems with approximately $n/m \leq 2$ while for problems with approximately $n/m \geq 2$, algorithm $BA(FCFS, JO, LBP, 5, SEL1)$ and also $BA(FCFS, JO, LBC, 5, SEL1)$ can be recommended. It should also be noted that for problems with n/m close to one, the average percentage deviation of the best value from $\overline{C}$ becomes very small.



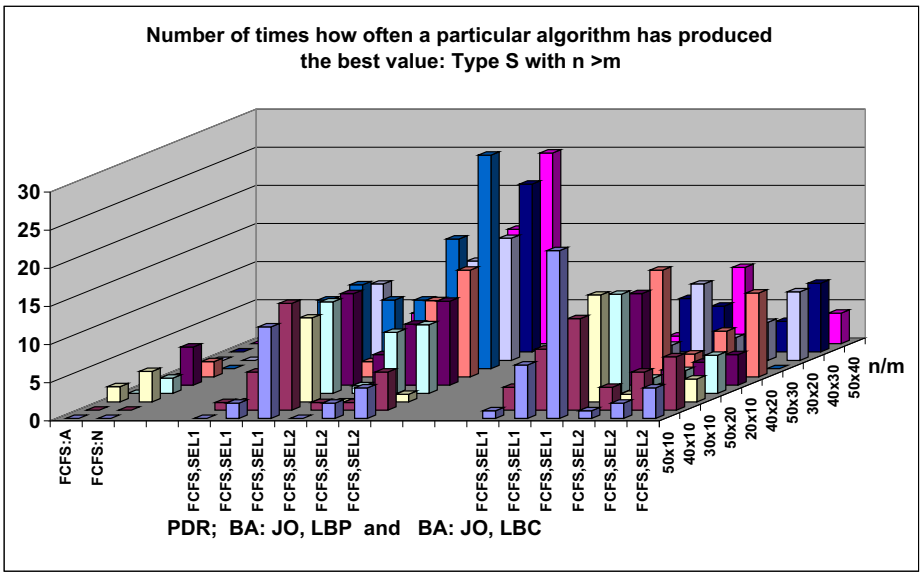


Figure 2: Evaluation of constructive heuristics for problems with $n > m$

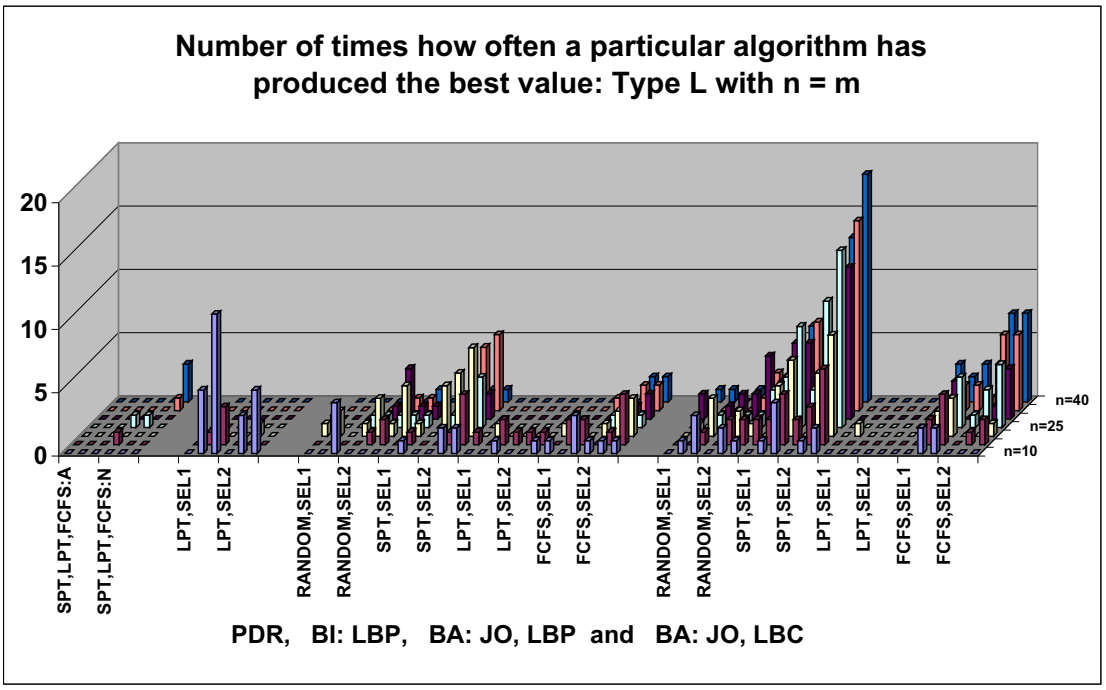
Table 1: Comparison of best values for problems with $n > m$

(n,m)	type	PDR	Beam-Append		$APD_{\overline{C}}$ of
			LBP	LBC	best value
(20,10)	S	11.294	9.840	9.863	9.479
	L	12.276	10.743	10.694	10.381
(30,10)	S	9.184	8.217	8.183	7.975
	L	10.126	9.330	9.235	9.049
(40,10)	S	8.133	7.422	7.413	7.269
	L	8.177	7.532	7.545	7.366
(50,10)	S	6.893	6.340	6.227	6.154
	L	7.985	7.455	7.521	7.356
(30,20)	S	3.933	2.853	2.941	2.718
	L	4.470	3.574	3.647	3.393
(40,20)	S	7.915	7.181	7.100	6.974
	L	9.124	8.400	8.270	8.154
(50,20)	S	3.952	3.361	3.385	3.283
	L	4.879	4.326	4.346	4.200
(40,30)	S	2.453	1.718	1.795	1.626
	L	2.830	2.294	2.366	2.192
(50,30)	S	4.235	2.939	3.631	2.928
	L	4.603	3.715	4.147	3.695
(50,40)	S	2.522	1.370	1.531	1.306
	L	2.274	1.789	1.874	1.719

For the large problems with $n = 50$ and $m = 40$, computational times on an AMD Athlon XP 2200+ (this computer has been used for the comparative study) are less than 0.4 s for the priority dispatching rules, less than 2.5 s for the Beam-Append procedures with $k = 1$ and less than 14.5 s for the Beam-Append procedures with $k = 5$.

Problems with n = m:

The results for square problems with $n = m$ are presented in Table 2 and Figures 3 and 4. In Table 2, we first present the **a**verage **p**ercentage **d**eviations APD_{LB} of the best variant of each type of algorithm from the preemptive lower bound (columns 3 - 6). In column 7, we present the average percentage deviation APD_{LB} of the best constructive solution from the preemptive lower bound, and in column 8 the average percentage deviation $APD_{\overline{C}}$ of the best constructive solution from the comparison value $\overline{C}$. In Figure 3, we give the number of times how often a particular algorithm has produced the best value. At the x-axis, the corresponding algorithms are described. Each notation represents three entries. The first group of algorithms considers priority dispatching rules (three active and three nondelay schedules), the next group considers Beam-Insert variants, and then Beam-Append variants with $LB = LBP$ and $LB = LBC$, respectively, are considered. For each type, the left entry represents the beam width $k = 1$, the middle entry represents $k = 3$ and the right one $k = 5$. At the y-axis, the corresponding values of $n(= m)$ are given. In Figure 4, we compress the results given in Figure 3 by showing how many times a particular type of algorithm has obtained the best value (note, however, that PDR represents 6 variants, Beam-Insert represents 5 variants and Beam-Append with $LB = LBP$ and $LP = LPC$, respectively, represent 20 variants each).



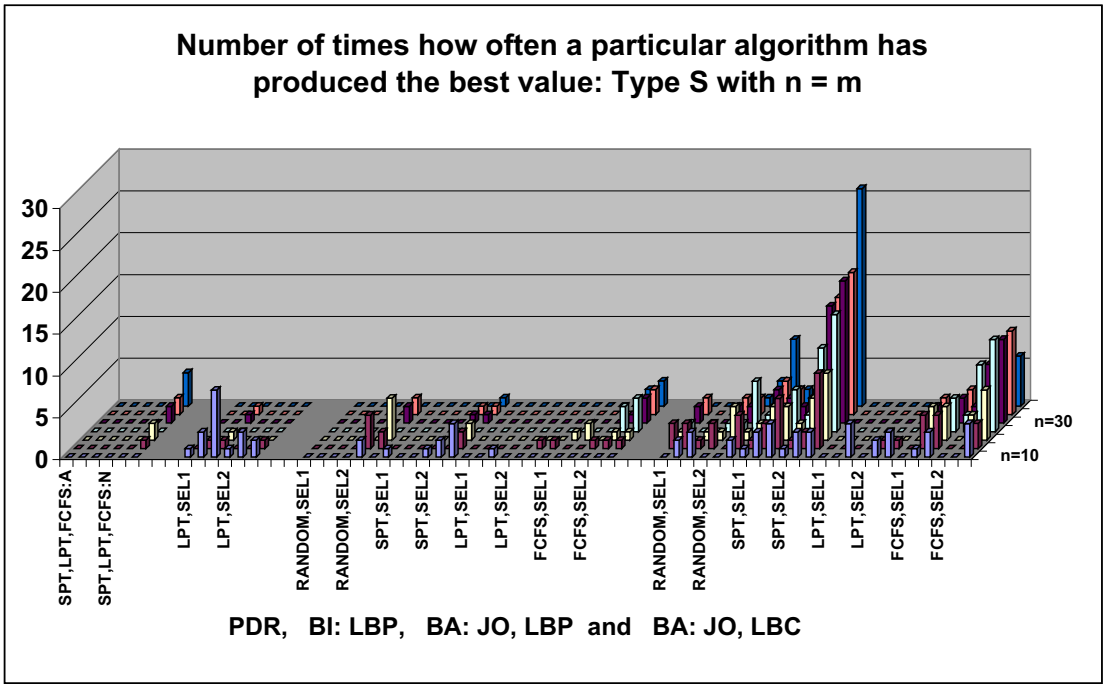
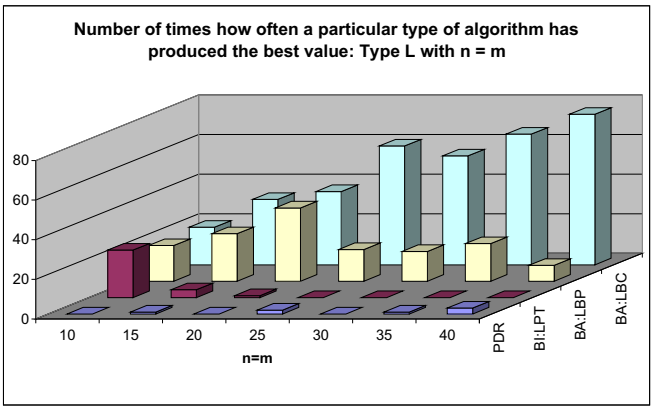


Figure 3: Evaluation of constructive heuristics for problems with $n = m$



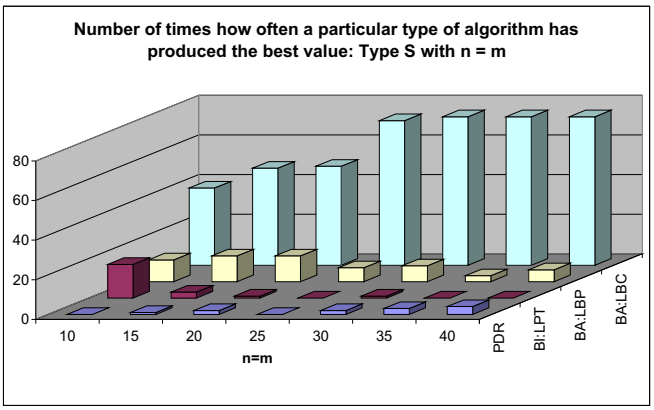


Figure 4: Comparison of the best variants of different types of heuristics

Table 2: Comparison of best values for problems with $n = m$

(n, m)	type	PDR	Beam-Insert	Beam-Append		APD_{LB} of	$APD_{\overline{C}}$ of
				LBP	LBC	best value	best value
(10,10)	S	14.973	12.321	12.523	11.904	11.449	9.027
	L	15.361	12.880	13.031	13.002	12.152	9.491
(15,15)	S	12.093	11.552	10.836	10.277	10.178	8.521
	L	12.478	11.875	11.033	10.808	10.538	8.848
(20,20)	S	10.498	10.559	9.571	8.992	8.923	7.841
	L	10.863	11.009	9.819	9.531	9.390	8.071
(25,25)	S	9.510	10.049	8.865	8.354	8.341	7.394
	L	9.795	10.253	8.981	8.670	8.601	7.585
(30,30)	S	8.946	9.737	8.508	8.317	7.878	6.920
	L	8.500	9.578	7.932	7.567	7.527	6.783
(35,35)	S	7.921	9.008	7.389	6.867	6.846	6.240
	L	8.331	9.556	7.820	7.494	7.464	6.635
(40,40)	S	7.823	9.026	7.265	6.820	6.785	6.127
	L	7.676	9.207	7.236	6.939	6.916	6.137

From Table 2 and Figures 3 and 4, we observe the following results for square problems with $n = m$. Priority dispatching rules give very seldom the best value. There are some instances for larger values of n, where algorithm $ND(FCFS)$ yields the best value. Procedure Beam-Insert is the best only for

small problems with $n = 10$. Here, variant $BI(LPT, LBP, 5, SEL1)$ can be recommended. For most problems, Beam-Append produces the best results (however, one should emphasize again that for fixed LB variant, 20 algorithms have been run in contrast to Beam-Insert, where $IO = LPT$ has been fixed and thus only 5 variants have been run). First, we see that $TBR = SPT$ is the best tie-breaking rule followed by $FCFS$ while $TBR = LPT$ works bad. For most of the problems, the use of $LB = LBC$ and $SEL = SEL2$ is recommended. Using the above settings, both the variants with $k = 5$ and $k = 3$ can be recommended (i.e., algorithms $BA(SPT, JO, LBC, 5, SEL2)$ and $BA(SPT, JO, LBC, 3, SEL2)$ and as a further variant $BA(FCFS, JO, LBC, 5, SEL2)$. The superiority of bound LBC over LBP increases with n (see Table 2 and Figure 4). The average computational times for the large problems with $n = m = 40$ are 0.3 s for priority dispatching rules, up to 12 min for Beam-Insert procedures with $k = 1$, up to 66 min for Beam-Insert procedures with $k = 5$, less than 1 s for Beam-Append with $k = 1$ and less than 6 s for Beam-Append with $k = 5$. Moreover, at least for the Beam-Insert variants, computational times are about 30 % lower for problems of type S in comparison with type L.

Problems with n < m:

The results for problems with $n < m$ are given in Tables 3 and 4. For each algorithm, we present the following values:

row 1: average percentage deviation APD_{LB} from the preemptive lower bound;
row 2: variation coefficient (i.e., standard deviation divided by APD);
row 3: number of times how often a particular algorithm has produced the best value;
row 4: number of times how often the lower bound has been obtained.

First, consider the results for the problems of type S. For small values of n/m, in particular $n = 10$ and $m \in \{30, 40, 50\}$, some variant of Beam-Insert reaches the lower bound for the majority of instances. For all problem classes with $n < m - 10$, both Beam-Insert procedures with beam width $k = 5$ yield average percentage deviations from the lower bound not greater than 0.34 %. For problems with a large number of machines (particularly for $m = 50$), son selection procedure $SEL1$ is slightly preferable while for small numbers of machines, variant $SEL2$ is slightly superior. The hardest problems are those with ratio n/m close to one (in our test the problems with $n = 40, m = 50$ and $n = 30, m = 40$. For small ratio n/m, among the dispatching rules algorithm $ND(SPT)$ works best while for values of n/m close to one, algorithm $ND(LPT)$ works best. For small values of n/m (approximately up to 2/3) algorithm Beam-Insert is clearly the best procedure. For problems of type S, the quality slightly increases with the beam width k. Both beam widths $k \in \{3, 5\}$ can be recommended and, due to the shorter running times, for several problem classes Beam-Insert with $k = 1$ is certainly recommended, too. In the case of $k > 1$ and problems of type S, both son selection procedures $SEL1$ and $SEL2$ produce a similar solution quality.

For the problems of type L, the conclusions from Table 4 are similar. There is a tendency that the results are slightly better than for problems of type S in terms of the average percentage deviation APD_{LB}. Among the dispatching rules, the tendency of best results with $ND(LPT)$ for problems with n/m close to one is even stronger (see (30,40) and (40,50) problems). To the contrary, the tendency of best results with $ND(SPT)$ for problems with $n = 10$ is weaker than for the problems of type S. Among the Beam-Insert variants, son selection procedure $SEL2$ is superior to $SEL1$ for most problem classes. It follows from Tables 3 and 4 that the results are becoming better as the ratio n/m decreases.

For the large problems with $n = 10$ and $m = 40$, computational times for the priority dispatching rules are less that 0.3 s, for the Beam-Insert variants with $k = 1$ up to 17 min and with $k = 5$ up to 85 min. For the Beam-Insert variants, computational times for problems of type of type S are approximately 20 % lower.

Table 3: Results for problems of type S with $n < m$

(n,m)	Priority dispatching rules				Beam-Insert				
					$SEL1$			$SEL2$	
	$A(SPT)$	$ND(SPT)$	$A(LPT)$	$ND(LPT)$	$k=1$	$k=3$	$k=5$	$k=3$	$k=5$
(10,20)	14.823	2.254	1.911	0.908	0.273	0.233	0.179	0.179	0.123
	0.149	0.442	0.445	0.585	0.685	0.756	0.707	0.750	0.771
	0	1	0	1	7	12	18	15	28
	0	1	0	0	1	4	6	3	5
(10,30)	8.995	0.763	0.685	0.475	0.061	0.036	0.021	0.031	0.019
	0.205	0.725	0.389	0.545	1.037	1.242	1.506	1.443	1.664
	0	3	0	0	13	26	33	26	35
	0	3	0	0	13	22	29	22	31
(10,40)	6.248	0.517	0.286	0.219	0.015	0.009	0.009	0.008	0.006
	0.191	0.969	0.494	0.613	2.108	0.000	0.000	0.000	0.000
	0	5	0	1	31	37	36	39	41
	0	5	0	1	31	37	36	39	41
(10,50)	4.866	0.223	0.203	0.170	0.007	0.005	0.001	0.006	0.006
	0.195	1.071	0.623	0.720	0.000	0.000	0.000	0.000	0.000
	0	16	0	3	36	42	48	37	37
	0	16	0	3	36	42	48	37	37
(20,30)	18.252	3.038	2.027	1.238	0.912	0.765	0.723	0.750	0.688
	0.085	0.272	0.285	0.357	0.269	0.255	0.266	0.219	0.220
	0	0	0	2	2	14	14	12	22
	0	0	0	0	0	0	0	0	0
(20,40)	12.103	1.402	0.795	0.566	0.228	0.169	0.164	0.178	0.150
	0.102	0.437	0.342	0.399	0.416	0.418	0.386	0.397	0.422
	0	0	0	0	6	16	14	12	20
	0	0	0	0	0	0	0	0	0
(20,50)	9.236	0.770	0.437	0.340	0.078	0.052	0.048	0.057	0.052
	0.098	0.419	0.289	0.416	0.702	0.608	0.659	0.555	0.608
	0	0	0	0	10	15	21	14	21
	0	0	0	0	0	3	1	0	1
(30,40)	19.677	3.194	2.044	1.284	1.385	1.261	1.166	1.232	1.193
	0.082	0.212	0.194	0.269	0.183	0.159	0.156	0.126	0.124
	0	0	0	15	5	4	19	8	13
	0	0	0	0	0	0	0	0	0
(30,50)	13.731	1.697	0.872	0.662	0.457	0.352	0.330	0.362	0.340
	0.059	0.335	0.232	0.291	0.240	0.270	0.235	0.195	0.186
	0	0	0	0	1	17	22	6	11
	0	0	0	0	0	0	0	0	0
(40,50)	19.917	3.172	2.011	1.327	1.817	1.656	1.594	1.651	1.617
	0.050	0.174	0.187	0.243	0.117	0.138	0.119	0.086	0.094
	0	0	1	37	2	1	8	3	3
	0	0	0	0	0	0	0	0	0

Table 4: Results for problems of type L with $n < m$

(n,m)	Priority dispatching rules				Beam-Insert				
					$SEL1$			$SEL2$	
	$A(SPT)$	$ND(SPT)$	$A(LPT)$	$ND(LPT)$	$k=1$	$k=3$	$k=5$	$k=3$	$k=5$
(10,20)	15.323	1.916	2.068	0.941	0.337	0.200	0.190	0.171	0.122
	0.0185	0.397	0.433	0.413	0.751	0.922	0.832	0.762	0.820
	0	0	0	0	3	15	10	15	25
	0	0	0	0	0	2	0	0	0
(10,30)	9.933	0.882	0.777	0.391	0.051	0.038	0.032	0.018	0.013
	0.170	0.571	0.478	0.518	1.754	1.177	1.398	0.000	0.000
	0	1	0	0	6	19	21	20	27
	0	1	0	0	4	13	16	12	17
(10,40)	6.844	0.513	0.356	0.250	0.014	0.007	0.006	0.011	0.009
	0.193	0.732	0.510	0.607	0.000	0.000	0.000	0.000	0.000
	0	4	0	1	12	30	34	19	22
	0	4	0	1	12	23	29	18	19
(10,50)	5.418	0.277	0.235	0.164	0.007	0.004	0.004	0.005	0.003
	0.151	1.008	0.587	0.668	0.000	0.000	0.000	0.000	0.000
	0	12	0	2	30	31	38	36	39
	0	12	0	2	27	30	34	33	35
(20,30)	19.047	2.591	2.151	0.871	0.842	0.766	0.731	0.683	0.630
	0.087	0.291	0.209	0.264	0.235	0.274	0.256	0.207	0.181
	0	0	0	8	2	8	13	12	20
	0	0	0	0	0	0	0	0	0
(20,40)	13.083	1.363	0.948	0.465	0.188	0.147	0.144	0.130	0.113
	0.109	0.285	0.335	0.289	0.391	0.373	0.380	0.344	0.396
	0	0	0	0	7	10	12	18	29
	0	0	0	0	0	0	0	0	0
(20,50)	9.855	0.790	0.507	0.326	0.064	0.044	0.043	0.042	0.034
	0.095	0.708	0.299	0.307	0.494	0.000	0.735	0.000	0.000
	0	0	0	0	5	12	16	14	23
	0	0	0	0	0	0	0	0	0
(30,40)	20.778	2.883	2.134	0.855	1.345	1.278	1.239	1.183	1.113
	0.068	0.200	0.180	0.245	0.174	0.170	0.157	0.131	0.102
	0	0	0	41	0	1	3	3	6
	0	0	0	0	0	0	0	0	0
(30,50)	14.864	1.677	0.964	0.451	0.354	0.325	0.300	0.284	0.261
	0.067	0.228	0.164	0.186	0.282	0.275	0.279	0.223	0.171
	0	0	0	0	7	5	19	15	26
	0	0	0	0	0	0	0	0	0
(40,50)	21.017	3.029	2.222	0.917	1.698	1.637	1.600	1.567	1.533
	0.050	0.191	0.158	0.234	0.100	0.089	0.081	0.078	0.068
	0	0	0	48	1	1	0	1	1
	0	0	0	0	0	0	0	0	0

5 Concluding Remarks

From our computational experiences, we can give the following conclusions:

- For the open shop problem with mean flow time minimization, the choice of an appropriate constructive solution procedure strongly depends on the relationship between the number n of jobs and the number m of machines while the processing times have less influence.
- Constructive heuristics based on matching procedures do not work well for mean flow time minimization. Among priority dispatching algorithms, the generation of nondelay schedules is superior to the generation of active schedules.
- For problems with $n > m$, procedure Beam-Append works clearly best. The choice of a concrete variant depends on the ratio of n/m, where approximately a ratio of $n/m = 2$ is the borderline for the individual selections.
- For square problems with $n = m$, the Beam-Insert procedure is superior only for small problems (however, computational times are higher) while for larger problems, Beam-Append is clearly superior. For most problem types, the use of $FCFS$ as tie-breaking rule, LBC as lower bound for the schedule evaluation and $SEL2$ as son selection procedure can be recommended.
- For problems with $n < m$, procedure Beam-Insert yields the best results. For many instances, even the lower bound is met for an appropriate variant, and the average percentage deviations are small for problems with $n/m \leq 2/3$. For larger ratios of n/m, a nondelay schedule according to the $FCFS$ rule works often best. Surprisingly, Beam-Append procedures work bad in this case. However, the computational times for procedure Beam-Insert are substantially larger than for the other procedures.

Most constructive algorithms presented in this paper have already been included into the program package LiSA - A Library of Scheduling algorithms (see http://lisa.math.uni-magdeburg.de), and the remaining ones will be included into the next version.

Acknowledgement: This work has been supported by INTAS (project 03-51-5501).

References

[1] Achugbue, J.O.; Chin, F.Y.: Scheduling the Open Shop to Minimize Mean Flow Time. SIAM J. on Computing, Vol. 11, 1982, 709 - 720.

[2] Alcaide, D.; Sicilia, J.; Vigo, D.: A Tabu Search Algorithm for the Open Shop Problem, Top, Vol. 5, 1997, 283 - 286.

[3] Bräsel, H.; Hennes, H.: On the Open-Shop Problem with Preemption and Minimizing the Average Completion Time, European J. Oper. Res., Vol. 157, 2004, 607 - 619.

[4] Bräsel, H.; Tautenhahn, T.; Werner, F.: Constructive Heuristic Algorithms for the Open-Shop Problem, Computing, Vol. 51, 1993, 95 - 110.

[5] Brucker, P.; Hurink, J.; Jurisch, B.; Wöstmann, B.: A Branch-and-Bound Algorithm for the Open-Shop Problem, Discrete Applied Mathematics, Vol. 76, 1997, 43 - 59.

[6] Dorndorf, U.; Pesch, E.; Phan-Huy, T.: Solving the Open Shop Scheduling Problem, Journal of Scheduling, Vol. 4, 2001, 157 - 174.

[7] Du, J.; Leung, J.Y.T.: Minimize Mean Flow Time in Two-Machine Open-Shops and Flow-Shops, Journal of Algorithms, Vol. 14, 1990, 24 - 44.

[8] Gonzalez, S.; Sahni, T.: Open Shop Scheduling to Minimize Finish Time, J. Assoc. of Comput. Mach., Vol. 23, 1976, 665 - 679.

[9] Gueret, C.; Jussien, N.; Prins, C.: Using Intelligent Backtracking to Improve Branch-and-Bound Methods: An Application to Open-Shop Problems, European J. Oper. Res., Vol. 127, 2000,344 - 354.

[10] Gueret, C.; Prins, C.: Classical and New Heuristics for the Open-Shop Problem: A Computational Evaluation, European J. Oper. Res., 107, 1998, 306 - 314.

[11] Gupta, J.N.D.; Werner, F.; Wulkenhaar, G.: Two-Machine Open Shop Scheduling with Secondary Criterion, Intl. Trans. Oper. Res., Vol. 10, 2003.

[12] Kubiak, W.; Sriskandarajah, C.; Zaras, K.: A Note on the Complexity of Open Shop Scheduling Problems, INFOR, Vol. 29, 1991, 284 - 294.

[13] Kyparisis, G.J.; Koulamas, C.: Open Shop Scheduling with Makespan and Total Completion Time Criteria, Comput. Oper. Res., Vol. 27, 2000, 15 - 27.

[14] Liaw, C.-F.: An Iterative Improvement Approach for the Nonpreemptive Open Shop Scheduling Problem, European J. Oper. Res., Vol. 111, 1998, 509 - 517.

[15] Liaw, C.-F.: A Hybrid Genetic Algorithm for the Open Shop Scheduling Problem, European J. Oper. Res., Vol. 124, 2000, 28 - 42.

[16] Liaw, C.-F.; Cheng, C.-Y.; Chen, M.: The Total Completion Time Open Shop Scheduling Problem with a Given Sequence of Jobs on One Machine, Comput. Oper. Res., Vol. 29, 2002, 1251 - 1266.

[17] Liu, C.Y.; Bulfin, R.L.: On the Complexity of Preemptive Open-Shop Scheduling Problems, Oper. Res. Lett., Vol. 4, 1985, 71 - 74.

[18] Liu, C.Y.; Bulfin, R.L.: Scheduling Ordered Open Shops, Comput. Oper. Res., Vol. 14, 1987, 257 - 264.

[19] Prins, C.: An Overview of Scheduling Problems Arising in Satellite Communications, Journal Oper. Res. Soc., Vol. 40, 1994, 611 - 623.

[20] Queyranne, M.; Sviridenko, M.: New and Improved Algorithms for Minssum Shop Scheduling, Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco/USA, 2000, 871 - 878.

[21] Queyranne, M.; Sviridenko, M.: Approximation Algorithms for Shop Scheduling Problems with Minsum Objective, Journal of Scheduling, Vol. 5, 2002, 287 - 305.

[22] Taillard, E.: Benchmarks for basic scheduling problems, European J. Oper. Res., Vol. 64, 1993, 278 - 285.

[23] Werner, F.; Winkler, A.: Insertion Techniques for the Heuristic Solution of the Job Shop Problem, Discrete Appl. Math., Vol. 50, 1995, 191 - 211.