Metaheuristic Algorithms for Open Shop Scheduling to Minimize Mean Flow Time

Michael Andresen, Heidemarie Bräsel, Marc Mörig,
Jan Tusch, Frank Werner, Per Willenius

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

Abstract: This paper considers 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 and continues recent work by Bräsel et al. (2005) on constructive algorithms. For this strongly NP-hard problem, we compare different iterative metaheuristic algorithms, namely simulated annealing, tabu search and genetic algorithms. For the first two types of algorithms, a number of different neighborhoods are suggested and tested together with their control parameters. For the genetic algorithm, new genetic operators are suggested based on the representation of a solution by the rank matrix describing the job and machine orders. 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. For most of the problems, the genetic algorithm is superior when fixing the same number of 30000 generated solutions. However, in contrast to makespan minimization problems, where the focus is on problems with an equal number of jobs and machines, it turns out that problems with a larger number of jobs than machines are the hardest problems.

Key words: Open shop scheduling, mean flow time, metaheuristic algorithms, simulated annealing, tabu search, genetic algorithm

1 Introduction

The open shop scheduling problem can be described as follows. 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 job J_i on machine M_j is denoted as operation (i, j), and the sequence in which the operations of a job are processed on the machines is arbitrary. All processing times of the operations are assumed to be given in advance. 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 many applications. 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. Further 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. [13], Liu and Bulfin [19] and Prins [21].

Let C_i be the completion time of job J_i, $i = 1, 2, \ldots, n$, i.e. the time when the last operation of this job is completed. In this paper, the optimization criterion is the minimization of the sum of the completion times of the jobs, also known as mean flow time minimization. Using the well-known 3-parameter classification, this problem is known as $O||\sum C_i$.

Most papers in the literature dealt with the minimization of makespan. Gonzalez and Sahni [10] 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 [10]. However, slight generalizations of the two-machine nonpreemptive problem lead to NP-hard problems. In view of the NP-hardness of $O||C_{max}$, branch and bound and heuristic algorithms have been developed for this problem. The currently best exact algorithm is the one given by DORNDORF et al. [8]. The authors focus on constraint propagation methods for reducing the search space. Several benchmark problems from the literature have been solved to optimality for the first time. In particular, among the problems given by TAILLARD [25], 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.

BRÄSEL et al. [6] 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. Other constructive algorithms have been given e.g. by GUERET and PRINS [11].

Among metaheuristic algorithms, a tabu search algorithm for the open shop problem with minimizing makespan was developed by ALCAIDE et al. [2]. Using list scheduling algorithms a starting solution is constructed, and the tabu search algorithm has extensively been tested on randomly generated instances. LIAW [16] presented a hybrid genetic algorithm for solving the open shop problem with makespan minimization approximately. The hybrid algorithm incorporates a local improvement procedure based on tabu search into a basic genetic algorithm. 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. Another genetic algorithm has been suggested by PRINS [22], which yields excellent results for the benchmark problems from the literature. In [22], it has been pointed out that the use of good initial solutions in the genetic algorithm is important for the quality of the final solution. In addition, the following features turned out to be essential: generation of active schedules only, small populations with distinct makespan values and chromosome reordering to increase the efficacy of crossover.

Some years ago the investigation of multicriteria open shop scheduling problems has begun. Polynomially solvable special cases of the two-machine problem together with heuristic algorithms and some extensions for the open shop hierarchical criteria scheduling problems to find a schedule with minimum mean flow time subject to minimum makespan have been given by KYPARISIS and KOULAMAS [14] and GUPTA et al. [12].

While there are some recent papers on multicriteria scheduling problems where one objective is the minimization of mean flow time, there exist only a few papers dealing only with the minimization of mean flow time (although from a practical point of view, often the minimization of the latter criterion is more important). ACHUGBUE and CHIN [1] proved that problem $O2||\sum C_i$ is NP-hard in the strong sense. We note that the the two-machine preemptive problem is NP-hard in the ordinary sense [9] while the three-machine preemptive problem is NP-hard in the strong sense [18].

LIAW et al. [17] 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 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 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 been given which 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 algorithm 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 $O|pmtn|\sum C_i$, Bräsel and Hennes [5] 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 [23, 24]. They presented a 5.83-approximation algorithm for the nonpreemptive open shop problem of minimizing 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, a 3-approximation algorithm has been given.

Recently, a comparative study of heuristic constructive algorithms for mean flow time open shop scheduling has been given by Bräsel et al. [4]. This paper compares matching heuristics, priority dispatching rules as well as insertion and appending algorithms combined with beam search on problems with up to 50 jobs and 50 machines, respectively. From the latter paper, it follows that the choice of an appropriate constructive algorithm strongly depends on the ratio n/m. In particular, it turned out that for problems with $n > m$, the rather fast procedure Beam-Append is superior while for problems with $n < m$, the more time-consuming procedure Beam-Insert gives the best results. For the square problems with $n = m$, an overlapping can be observed: For small problems, the Beam-Insert procedure is slightly superior while for larger problems, variants of the Beam-Append procedure are better. However, the procedures are rather sensitive with respect to parameter settings. In our study, we will use some constructive procedures from this paper for finding the initial solutions for the metaheuristic algorithms under consideration.

The remainder of the paper is organized as follows. In Section 2, we introduce some basic notions and we give an estimate for the optimal objective function value. In Section 3 we describe the components of the metaheuristic 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 this paper, 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 put the operations of job J_i in row i and the operations on machine M_j in column j, we get the graph $G(MO, JO) = G(MO) \cup G(JO)$, 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. We call such an acyclic digraph $G(MO, JO)$ a *sequence graph*. Note that all above graphs represent partial orders on the set of operations. Similarly as in [6, 26], we describe a sequence graph $G(MO, JO)$ by its *rank matrix* $A = (a_{ij})$, i.e., the entry $a_{ij} = k$ means that a path to operation (i, j) with a maximal number of operations includes k operations. Due to this property, equality $a_{ij} = k$ implies that there is no other operation with rank k in row i and column j, and the so-called *sequence property* is satisfied: 'For each $a_{ij} = k > 1$, integer $k - 1$ occurs as entry in row i or column j (or both).' Now we assign the processing time t_{ij} as the weight to operation (i, j) in $G(MO, JO)$. The computation of a longest path

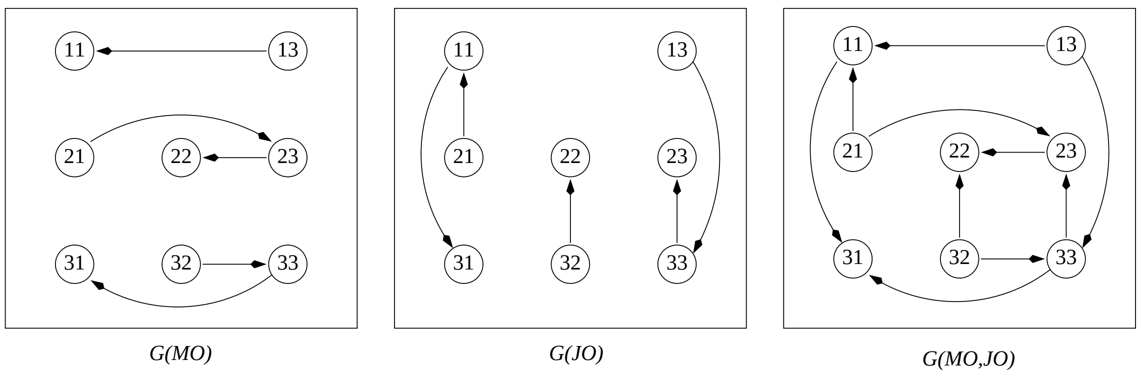

Figure 1: $G(MO)$, $G(JO)$ and $G(MO, JO)$

to the vertex (i, j) with (i, j) included in an acyclic digraph $G(MO, JO)$ gives the completion time c_{ij} of operation (i, j) in the semiactive schedule $C = (c_{ij})$. We remind that a schedule is called *semiactive* if no operation can start earlier without changing the underlying sequence graph. For the above example, 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:

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

In addition, 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, where vertex (i, j) is not included. It can be noted that the advantage of the use of the rank matrix in contrast to the usual description of a solution by a permutation (i.e. sequence) of the operations is the exclusion of redundancy: different rank matrices describe different solutions while different operation sequences may describe the same solution.

In [5], Bräsel and Hennes generalized the above model to the preemptive problem. They also derived a lower bound for problem $O|pmtn|\sum C_i$ which, of course, is also a lower bound for the nonpreemptive problem. This 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 assume that the jobs and machines are ordered such that

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

Clearly, $C_i \ge T_i$ holds in an arbitrary feasible schedule and any unavoidable waiting time of job J_i increases the completion time C_i. Now, the goal is to identify such unavoidable waiting times of jobs. Without loss of generality we can assume that $n = m$ (otherwise we introduce dummy jobs or machines). Assume first that $T_1 > \overline{T}_1$, 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 exist subintervals of a total length h_1 having the following property: at each time inside these subintervals, there can be no more than $n - 2$ jobs processed simultaneously with job J_1 since one machine different from M_1 is occupied by job J_1 and machine M_1 is idle. Therefore, one job (or several jobs) with

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

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

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

Now this procedure is repeated with other jobs satisfying inequality (2) until we get $h_1 = 0$.

If $T_1 < \overline{T}_1$ holds, then there is an unavoidable idle time on machine M_1, and the rest is symmetric to the first case. Now jobs $J_2, J_3, \ldots, J_n$ are considered in a similar way as describe above and finally, the lower bound for problem $O|pmtn|\sum C_i$ is equal to sum of the final T_i values (see also [5]).

3 Metaheuristic Algorithms

In the following subsections, we describe the three metaheuristic algorithms included into our comparative study: simulated annealing, tabu search and a genetic algorithm. At the end of the next section, we briefly report on some further metaheuristic algorithms that have been included into our experiments.

3.1 Simulated Annealing

Simulated annealing is an enhanced version of local optimization. Annealing refers to the process when physical substances are raised to a high energy level and then gradually cooled until some solid state is reached. The goal of this process is to reach the lowest energy state. In this process physical substances usually move from higher energy states to lower ones if the cooling process is sufficiently slow. However, there is some probability at each stage of the cooling process that a transition to a higher energy state will occur, but this probability of moving to a higher energy state decreases in this process.

A basic simulated annealing algorithms starts with generating an initial solution (rank matrix) R taken as the current starting solution. Then a neighbor (rank matrix) R^* of (rank matrix) R is generated and the difference in the objective function values $\Delta = f(R^*) - f(R)$ is calculated. If $\Delta < 0$, the new solution R^* is accepted as new starting solution since it has a better function value. If the objective function value increases (i.e. $\Delta \geq 0$), the generated neighbor may also be accepted with a probability $exp(-\Delta/T)$, where T is a control parameter called the temperature. This temperature is periodically reduced by a cooling scheme every NT iterations, where NT is a preset parameter called the epoch length. In our implementation, we investigated in particular the influence of the chosen neighborhood and the cooling scheme.

Neighborhood:

First, we briefly discuss the generation of neighbors of a current solution described by a sequence graph $G(MO, JO)$ resp. the rank matrix R. In the case of a job shop problem, often a neighbor is generated by interchanging two adjacent jobs in exactly one machine order (this means that the ranks in the current rank matrix are changed in such a way that in exactly one machine order two adjacent jobs have been interchanged). We denote this neighborhood as machine oriented API neighborhood, abbreviated as API(MO). In an open shop problem we can, due to symmetry, consider also a neighborhood based on adjacent pairwise interchanges in the job order on a machine, abbreviated as API(JO). In our algorithms, we use the union of both neighborhoods, abbreviated as API.

A second neighborhood considered is crit-API which is a restricted API neighborhood, where a neighbor must satisfy a necessary condition for an improvement of the makespan value. This neighborhood is based on the so-called block approach originally introduced for shop scheduling problems with makespan minimization. It considers only such adjacent pairwise interchanges of two operations, where the current critical path is ‘destroyed’. Clearly, the latter neighborhood is a subneighborhood of the complete API neighborhood.

Moreover, we consider the neighborhood k-API, in which a neighbor is generated from the current sequence graph $G(MO, JO)$ resp. the corresponding rank matrix R by generating consecutively up to k neighbors in the API neighborhood (i.e. a path containing up to k arcs in the resulting neighborhood graph is generated).

As a generalization of the shift neighborhood for permutation problems we use a neighborhood SHIFT, where exactly one operation is changed in the relative order of operations, namely in such a way that either in the job order on one machine or in the machine order of one job exactly one operation is shifted left or right. Analogously, we use the pairwise interchange (PI) neighborhood, where exactly two operations belonging to the same job or to be executed on the same machine are changed in the relative order, namely these two chosen operations are interchanged either in the job order on one machine or in the machine order of one job. Similarly, we can define the crit-SHIFT neighborhood, where a neighbor generated in the SHIFT neighborhood must satisfy the necessary condition for an improvement of the objective function value (that is, crit-SHIFT is a subneighborhood of the SHIFT neighborhood).

Finally, we consider the h-reinsertion neighborhood. In the latter case, $g \in \{1, 2, \ldots, h\}$ operations of the same job are chosen. Then the first operation is deleted and reinserted at the best position, i.e. the insertion algorithm (see Algorithm Beam-Insert in [4]) is applied to this operation. Then this procedure is repeated for the other chosen operations.

Cooling scheme:

As the cooling scheme, a geometric, a Lundy-Mees and a linear reduction scheme have been considered. The geometric cooling scheme reduces the current temperature T^{old} to the new temperature T^{new} in the next epoch according to

$$T^{new} = \alpha \cdot T^{old},$$

where $0 < \alpha < 1$. The Lundy-Mees scheme (see [20]) is characterized by

$$T^{new} = T^{old}/(1 + \beta \cdot T^{old}),$$

where $0 < \beta < 1$. Finally, a linear (or arithmetic) reduction scheme reduces the temperature in the subsequent epoch according to

$$T^{new} = T^{old} - \gamma \cdot T^0,$$

where T^0 is the initial temperature. In our experiments we fixed the initial temperature T^0, the epoch length and set the parameter α, β and γ in such a way that the final temperature is close to zero taking into account that in our study, the number of generated solutions is settled in advance and therefore, the number of epochs is fixed.

In addition to the usual procedure of one cooling cycle, we also considered different numbers of cooling cycles, where the temperature reduction is done faster within one run such that, if the final temperature is reached, the procedure is restarted again with the initial temperature. The number NCC denotes the number of cooling cycles.

3.2 Tabu Search

Tabu search is an iterative procedure that moves in each iteration to the best neighbor investigated which has not necessarily a better objective function value. Nevertheless, subsequent moves may cause the search to move back to an already visited solution. To avoid cycling (i.e. coming back to a recently visited solution) and to escape from a poor local optimum, a tabu restriction is used that makes selected attributes of these moves forbidden (i.e. they are declared to be tabu). Tabu restriction is enforced by a tabu list L which stores the move attributes to avoid reversals of moves. The tabu list contains the moves describing the last LS solutions visited, where LS is the size of the tabu list. It controls the memory of the search process (we use a short-term memory in our implementation of tabu search). In one iteration, a tabu search algorithm may investigate all nontabu neighbors or only a certain number or percentage of nontabu neighbors. In the experiments, we mainly test the influence of the neighborhood, the size of the tabu list and the number of generated nontabu neighbors in one iteration.

Concerning the neighborhoods we used the same as tested for simulated annealing. For characterizing the moves, we used neighborhood-specific descriptions. Furthermore, we generated in each iteration a fixed number NN of nontabu neighbors. For the length LS of the tabu list, we tested constant values.

3.3 Genetic Algorithm

Genetic algorithms belong to the class of artificial intelligence techniques and they are based on Darwin's evolutionary theory about 'survival of the fittest and natural selection'. A genetic algorithm is characterized by a parallel search of the state space by keeping a set of possible solutions under consideration, called a population. A new generation is obtained from the current population by applying genetic operators such as mutation and crossover to produce new offspring. The application of a genetic algorithm requires an encoding scheme for a solution (also denoted as an individual), the choice of genetic operators, a selection mechanism and the determination of genetic parameters such as the population size and probabilities of applying the genetic operators.

For the representation of an individual we again use the rank matrix of a sequence graph. While often the initial population is created randomly, we also used the constructive algorithms from [4] to generate an initial population. The size of the population is fixed by the parameter *popsize*.

Mutation:

Mutation serves to prevent that all solutions in the population fall into a local optimum. In our algorithm, a mutation is performed as follows. Initially the rank of exactly one randomly chosen operation is changed in the chosen individual described by the rank matrix R. Assume that (i, j) is the chosen operation. Let $r_{ij} = l$ and k be the maximal rank in row i and column j of R, respectively. Then we may set $r_{ij} = k^*$, where $k^* \in \{1, 2, \dots, k, k+1\} \setminus \{l\}$. By this change, the resulting protochild is usually not feasible. Now the relative order of the remaining operations in R is maintained and we modify the ranks in such a way that a rank matrix is obtained again. To this end, we put a linear order on the operations in such a way that a smaller number in the linear order indicates that the rank of this operation is not greater than the rank of an operation with a larger number. In the case of ties, a lexicographical order is applied with one exception: The mutated operation gets the smallest number in the linear order among all operations with the same rank. Finally, from this linear order of operations, the resulting rank matrix is constructed.

To illustrate, consider the individual described by the rank matrix

$$R = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 1 & \mathbf{3} \\ 3 & 4 & 1 \end{pmatrix}$$

Assume that operation $(2, 3)$ has to be chosen for applying a mutation. We have $r_{23} = 3$ (drawn in bold face above). There are four possibilities for performing a mutation in this case, namely due to $k = r_{13} = 4$, we can set the rank $r_{23} \in \{1, 2, 4, 5\}$. Let us consider the case $r_{23} = k^* = 1$. We get the following protochild PC_1, then the corresponding linear order OP_1 of the operations (note that the mutated operation has now a smaller number than the other operations $(1, 1)$ and $(2, 2)$ with rank 1).

Finally transforming OP_1 into the corresponding rank matrix gives the offspring OFF_1:

$$PC_1 = \begin{pmatrix} 1 & 2 & 4 \\ 2 & 1 & \mathbf{1} \\ 3 & 4 & 1 \end{pmatrix} \implies OP_1 = \begin{pmatrix} 2 & 5 & 8 \\ 6 & 3 & \mathbf{1} \\ 7 & 9 & 4 \end{pmatrix} \implies OFF_1 = \begin{pmatrix} 1 & 3 & 4 \\ 3 & 2 & \mathbf{1} \\ 4 & 5 & 2 \end{pmatrix}.$$

The application of a mutation is controlled by a probability parameter *pmuta*.

Crossover:

The crossover is applied to two randomly chosen individuals in the current population. It exchanges the ranks of a randomly chosen set of operations. This yields two (usually infeasible) protochildren. The rest works in a similar way as the mutation described above. This means that the relative order of the remaining operations from the parent is maintained, and both parts (i.e. the relative orders of the operations with exchanged ranks and the relative orders of the remaining operations) are now combined to a feasible rank matrix. To this end, again a linear order is put on the operations in such a way that a smaller number in the linear order indicates that the rank of this operation is not greater than the rank of an operation with a larger number. In the case of ties, a lexicograhical order is applied with the following exception: The operations included in the exchange of ranks get the smallest possible numbers among all operations with the same rank (this guarantees that the operations with a new rank have a 'higher priority'). Finally, from this linear order of operations, the resulting rank matrix is constructed.

To illustrate the crossover, we consider the individuals described by the rank matrices

$$R_1 = \begin{pmatrix} 1 & \mathbf{2} & 4 \\ 2 & \mathbf{1} & \mathbf{3} \\ 3 & 4 & 1 \end{pmatrix} \quad \text{and} \quad R_2 = \begin{pmatrix} 2 & \mathbf{1} & 4 \\ 1 & \mathbf{3} & \mathbf{2} \\ 4 & 2 & 3 \end{pmatrix}.$$

Assume that the operations $(1,2),(2,2)$ and $(2,3)$ marked in bold face above have been chosen for performing the crossover. This yields the two protochildren

$$PC_1 = \begin{pmatrix} 1 & \mathbf{1} & 4 \\ 2 & \mathbf{3} & \mathbf{2} \\ 3 & 4 & 1 \end{pmatrix} \quad \text{and} \quad PC_2 = \begin{pmatrix} 2 & \mathbf{2} & 4 \\ 1 & \mathbf{1} & \mathbf{3} \\ 4 & 2 & 3 \end{pmatrix}.$$

and the corresponding linear orders of operations

$$OP_1 = \begin{pmatrix} 2 & \mathbf{1} & 8 \\ 5 & \mathbf{6} & \mathbf{4} \\ 7 & 9 & 3 \end{pmatrix} \quad \text{and} \quad OP_2 = \begin{pmatrix} 4 & \mathbf{3} & 8 \\ 2 & \mathbf{1} & \mathbf{6} \\ 9 & 5 & 7 \end{pmatrix}$$

from which the resulting two offspring can be constructed:

$$OFF_1 = \begin{pmatrix} 2 & \mathbf{1} & 4 \\ 3 & \mathbf{4} & \mathbf{2} \\ 4 & 5 & 1 \end{pmatrix} \quad \text{and} \quad OFF_2 = \begin{pmatrix} 3 & \mathbf{2} & 5 \\ 2 & \mathbf{1} & \mathbf{3} \\ 5 & 3 & 4 \end{pmatrix}$$

The application of a crossover is controlled by a probability parameter *pcross*.

We note that the mutation and crossover described above is rather different from those operators usually applied in genetic algortihms for shop scheduling problems when the algorithm works with a linear order of the operations.

Selection:

We use an elitist strategy combined with 2/4 selection. This means that the best individual is immediately included into the next generation. To generate new individuals, two individuals of the current generation are selected and by means of the genetic operators mutation and selection two offspring are generated. Among these four individuals, the two individuals with the highest fitness value (i.e. with the smallest objective function value) are included into the next generation. This procedure is continued until the new generation has been obtained. We used this 2/4 selection in contrast to e.g. roulette wheel selection due to the very good performance for the makespan minimization open shop problem (see [16]).

4 Computational Results

In this section, we present computational results for the metaheuristic algorithms discussed in Section 3. It is our goal to present relatively fast algorithms. Therefore, for the study presented in this paper, we allowed each procedure to generate the same fixed number of only 30000 solutions. Note that this does not necessarily mean that computational times are very similar.

4.1 Initial Tests for Metaheuristic Algorithms

All initial tests have been performed for the following problem sizes:

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

For all problem types, we considered 25 instances in the initial tests with processing times from the interval $[1, 100]$). These are problems of type L in [4] (we did not include problems of type S from [4] since in that paper, it was found that results do not substantially depend on the interval from which processing times are taken). We discuss the results for the three metaheuristic algorithms.

Simulated Annealing:

Among the neighborhoods, the SHIFT neighborhood is clearly on the first rank while the API neighborhood is the second best. In particular, the reinsertion neighborhood works weak, but also the pairwise interchange (PI) neighborhood is clearly not competitive to the SHIFT neighborhood. In addition, the restriction to the crit-neighborhoods is mostly worse than the original complete neighborhood (this results from the fact that they try to reduce the makespan value what, however, often does not reduce mean flow time).

For the cooling schemes, we have used an epoch length of 100 and consequently, there are 300 cycles (i.e. epochs) with a constant temperature in the search. We have found that an extremely low initial temperature is strongly required. In particular, we observed that in the case of an improving generated neighbor, the reduction in the objective function value is usually only very small (mostly less than 5 time units). We found that cooling schemes with an initial temperature T^0 with $2 \leq T^0 \leq 10$ can be recommended and with this choice, the influence of the cooling scheme is then only marginal, where a geometric cooling scheme is slightly preferred. The variants with one and five cooling cycles yield similar results. Sometimes the corresponding variant with $NCC = 1$ is superior while sometimes $NCC = 5$ is better. For the comparative study, we included eight variants of simulated annealing: $T^0 \in \{2, 10\}$ and $NCC \in \{1, 5\}$, each parameter combination applied to the SHIFT and the API neighborhoods.

Tabu Search:

We have found that the API neigbourhood produced the best results. The 3-API neighborhood is on the second rank. It can be observed that for tabu search, neighborhoods are preferrred which change the current solution only slightly. For the length of the tabu list, no variant is clearly preferred while for the number of neighbors, a larger value of parameter NN is superior (this also underlines that for the problem under consideration, a more detailed investigation of the neighborhood is preferred so that a possible increase in the objective function value is kept as small as possible). Concerning the length of the tabu list and the number of generated neighbors, we recommend $LS = 10$ and $NN = 50$ so that for the comparative study, four variants are included (the latter parameter setting applied to the API and 3-API neighborhoods).

Genetic Algorithm:

First, we have found that the sum of the probabilities for applying the genetic operators mutation and crossover should be around 1.0. Therefore, we performed tests with five variants $pcross \in \{0.2, 0.3, 0.4, 0.5, 0.6\}$ combined with $pmuta = 1 - pcross$. Moreover, we worked with a population size $popsize \in \{5, 10, 15, 20, 30, 50, 100\}$. Since we fixed 30000 generated solutions, this means that for the number $Ngener$ of generations, we have $Ngener \in \{6000, 3000, 2000, 1500, 1000, 600, 300\}$. This gives 35 variants (for the initial tests we used a randomly generated initial population). For problems with $n \leq m$, we observed that a larger population size is favorable and that larger mutation rates are slightly preferred. For these reasons, we included four variants of the genetic algorithm into the comparative study: $popsize \in \{50, 100\}$ and $pmuta \in \{0.6, 0.8\}$. For problems with $n > m$, the only difference in the recommendation concerns the population size of the genetic algorithm. Here smaller population sizes are preferred. The difference in the recommended population sizes follows from the restriction to a rather small number of generated solutions and the observation that problems with $n > m$ appear to be much harder than the other ones, see also later in Section 4.2. This means that it can be expected that also for problems with $n > m$ larger population sizes can be recommended provided that a considerably larger number of generated solutions is allowed. For $n > m$, we included variants with $popsize \in \{10, 50\}$ into the comparative study (the values of parameter $pmuta$ do not change) so that again four variants of a genetic algorithm are considered.

4.2 Comparative Study of Metaheuristic 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 processing times from the interval $[1, 100]$ yielding a total of 1350 instances.

For the comparative study, we used as initial solution for simulated annealing and tabu search the best constructive solutions from the procedures recommended in [4]: In particular, for the problems with $n = m$, we took the best from 23 variants (6 priority dispatching rules, 5 Beam-Insert procedures and 12 Beam-Append procedures). For the problems with $n > m$, we took the best solution among 14 variants (2 priority dispatching rules and 12 Beam-Append procedures). Finally, for the problems with $n < m$, we took the best solution among 9 procedures (4 priority dispatching rules and 5 Beam-Insert variants). The number of variants differs for the problem sizes since we did only include variants that contributed good values among the constructive algorithms (and for the square problems more variants yielded for a specific problem good values in comparison with the nonsquare problems). When applying a genetic algorithm, we used the best of these constructive algorithms as a part of the initial population (possibly filled with further randomly generated nondelay schedules described by the corresponding rank matrices). In the case of a random initial population, we randomly generated rank matrices representing nondelay schedules.

We have used the following five groups of algorithms (remind that each of these groups includes four variants according to the settlements in Section 4.1):

- *GA-R* - genetic algorithm with randomly determined initial population;
- *GA-C* - genetic algorithm with an initial population including some solutions determined by constructive algorithms;
- *TS* - tabu search with the best constructive solution as initial solution;
- $SA(S)$ - simulated annealing using the SHIFT neighborhood with the best constructive solution as initial solution;
- $SA(API)$ - simulated annealing using the API neighborhood with the best constructive solution as initial solution.

According to our recommendations, each group of algorithms is run with the recommended four variants (see subsection on initial tests above). Moreover, *GA*-Best and *SA*-Best refer to the best value among all considered groups of the genetic algorithm and simulated annealing, respectively. Moreover, 'Best' refers to the best value among all algorithms included into the comparative study. We present the results separately for the cases $n < m$, $n = m$ and $n > m$.

Problems with $\mathbf{n < m}$:

The results for problems with $n < m$ are given in Table 1. For each algorithm, we present the following values:

- row 1: average percentage deviation of the best solution obtained by the algorithms of the corresponding group from the preemptive lower bound;
- row 2: number of times some variant of the corresponding group of algorithms has produced the best value;
  (number of times the lower bound has been obtained by some variant of the corresponding group of algorithms);
- row 3: average computation time in seconds on an AMD Athlon XP 3200+.

First, from [4] it is already known that the best constructive algorithm has a rather small percentage deviation from the preemptive lower bound for this type of problems. Therefore, also percentage deviations of the metaheuristic algorithms from the lower bound are rather small. Indeed, it can be seen that for the majority of groups of algorithms, the best variant reaches the lower bound for most of the problems with 10 jobs. In particular, the problems become easy with reducing the ratio n/m. In most cases, the genetic algorithm outperforms the other algorithms, but for several problem types simulated annealing is on the first rank. However, the computational times for the genetic algorithms are considerably larger than for simulated annealing (although the same number of solutions has been generated).

Problems with $\mathbf{n = m}$:

The results for square problems are given in Table 2. The meaning of the rows is as follows:

Table 1: Results of the comparative study for problems with $n < m$

	GA			*TS*	*SA*			Best
(n, m)	*GA-R*	*GA-C*	*GA*-Best		*SA(S)*	*SA(API)*	*SA*-Best	
(10,20)	0.006	0.008	0.002	0.059	0.027	0.053	0.025	0.001
	41 (38)	38 (37)	48 (44)	10 (10)	19 (18)	12 (12)	23 (22)	50 (46)
	53.71	55.11	54.41	9.75	3.01	9.08	6.04	26.13
(10,30)	0.000	0.000	0.000	0.002	0.001	0.001	0.000	0.000
	50 (50)	50 (50)	50 (50)	41 (41)	45 (45)	46 (46)	48 (48)	50 (50)
	108.66	113.76	111.21	19.92	4.84	18.83	11.83	53.20
(10,40)	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000
	50 (50)	50 (50)	50 (50)	48 (48)	50 (50)	48 (48)	50 (50)	50 (50)
	182.15	190.74	186.44	31.81	7.06	30.17	18.62	88.39
(10,50)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	50 (50)	50 (50)	50 (50)	48 (48)	50 (50)	49 (49)	50 (50)	50 (50)
	278.93	292.34	285.63	50.99	9.75	47.43	28.59	135.89
(20,30)	0.550	0.435	0.415	0.529	0.501	0.515	0.489	0.407
	9 (0)	29 (0)	38 (0)	0 (0)	9 (0)	7 (0)	15 (0)	50 (0)
	372.25	389.73	380.99	46.57	15.39	43.57	29.48	173.50
(20,40)	0.287	0.084	0.084	0.086	0.093	0.083	0.082	0.079
	0 (0)	25 (0)	25 (0)	13 (0)	13 (0)	26 (0)	34 (0)	50 (0)
	616.64	641.34	628.99	76.00	21.82	71.81	46.81	285.52
(20,50)	0.158	0.018	0.018	0.019	0.022	0.017	0.016	0.015
	0 (0)	24 (1)	24 (1)	11 (0)	9 (0)	32 (0)	34 (0)	50 (1)
	921.17	988.62	954.89	117.67	29.93	112.23	71.08	433.92
(30,40)	0.971	0.650	0.643	0.712	0.702	0.680	0.656	0.625
	4 (0)	29 (0)	33 (0)	0 (0)	9 (0)	11 (0)	20 (0)	50 (0)
	1521.76	1397.56	1459.66	125.47	46.38	114.26	80.32	641.08

- row 1: average percentage improvements of the best constructive solution in comparison with the difference 'best constructive function value minus preemptive lower bound' (so 100 would mean that for all instances, the lower bound has been reached), where the results again refer to the best solution obtained within the corresponding group of algorithms;
- row 2: average percentage deviation of the best solution obtained by the algorithms of the corresponding group from the premptive lower bound;
- row 3: number of times some variant of the corresponding group of algorithms has produced the best value;
- row 4: average computation time in seconds on an AMD Athlon XP 3200+.

For the majority of problems, the genetic algorithms yield clearly the best results. Both variants, i.e. the initial population filled with random solutions as well as including some solutions obtained by the constructive algorithms contribute best values. For the simulated annealing algorithms, both neighborhoods SHIFT and API have to be considered. For the small problems, the SHIFT neighborhood is better while with increasing problem size, the API neighborhood becomes superior. The tabu search algorithm is not competitive and yields only marginal improvements of the best constructive solution. It can also be observed that the average percentage deviation of the best solution obtained substantially decreases with increasing problem size (from 10.51 % for $n = 10$ to 6.50 % for $n = 40$). Summarizing, the genetic algorithms are clearly the best among all algorithms tested, however, for generating 30000 schedules, they need the largest CPU time (in contrast to the much faster simulated annealing algorithm).

Table 2: Results of the comparative study for problems with $n = m$

	GA			*TS*	*SA*			Best
(n, m)	*GA-R*	*GA-C*	*GA*-Best		*SA*(*S*)	*SA*(*API*)	*SA*-Best	
(10,10)	14.75	12.66	16.76	2.57	6.98	1.61	7.03	17.03
	10.83	11.02	10.54	12.21	11.65	12.32	11.65	10.51
	29	16	45	1	5	0	5	50
	19.13	19.09	19.11	4.24	1.64	3.75	2.70	9.57
(15,15)	13.52	14.27	15.90	1.72	6.42	2.59	6.56	15.97
	9.45	9.37	9.18	10.73	10.25	10.64	10.23	9.18
	21	26	47	0	3	0	3	50
	73.27	73.16	73.21	11.84	4.83	10.78	7.80	34.77
(20,20)	13.64	13.97	15.28	1.55	4.97	4.76	6.08	15.28
	8.60	8.61	8.46	9.82	9.51	9.50	9.38	8.46
	19	31	50	0	0	0	0	50
	198.40	196.18	197.29	23.51	11.35	21.50	16.42	90.19
(25,25)	11.50	11.62	12.80	0.63	2.88	4.49	4.62	12.80
	7.92	7.91	7.81	8.91	8.71	8.57	8.54	7.81
	24	25	49	0	0	1	1	50
	218.43	217.50	217.96	28.91	13.68	26.13	19.90	100.93
(30,30)	11.69	11.75	12.95	0.60	1.81	5.17	5.18	12.97
	7.05	7.04	6.95	7.93	7.84	7.56	7.55	6.95
	21	27	48	0	0	2	2	50
	754.81	748.19	751.50	73.73	39.17	66.71	52.94	336.52
(35,35)	9.00	9.38	10.10	0.23	1.40	4.96	5.03	10.29
	7.09	7.06	7.01	7.77	7.68	7.39	7.39	6.99
	19	26	45	0	0	5	5	50
	1804.29	1519.32	1661.81	162.67	80.36	152.05	116.20	743.74
(40,40)	8.11	7.95	9.11	0.15	0.98	4.70	4.71	9.31
	6.58	6.60	6.52	7.17	7.11	6.84	6.84	6.50
	27	17	44	0	0	6	6	50
	2401.88	2398.48	2400.18	213.05	121.94	192.29	157.11	1065.53

Problems with n $>$ m:

The results for the problems with $n > m$ are given in Table 3. The three rows for each problem size correspond to the first, third and fourth rows in Table 2. For these problems, we do not refer to the deviations from the lower bound since the preemptive bound is extremely weak in this case. It can be observed that the percentage improvements over the best contructive solution from [4] are rather small. If a random initial population is used in the genetic algorithm, the objective function values of the solutions finally obtained are even far away from those of the best constructive solutions (see the negative values in the first rows for each problem size). We have also observed that a moderate increase in the number of generated solutions does not change this behavior substantially. We also note that, although percentage improvements are small, the objective function value of the constructive algorithms are typically improved by about 100 - 300 time units (which requires a considerable number of improvements - see the comment in the initial tests). This indicates that in contrast to makespan minimization problems (where usually only square problems are considered), instances with $n > m$ are the hardest ones. Generating only 30000 solutions, the genetic algorithm with an initial population using some of the constructive algorithms gives the best results, followed by simulated annealing. For this type of problems, the use of the SHIFT neighborhood is superior for small problems with $m = 10$, while for the other problems the API neighborhood is superior for simulated annealing. The tabu search algorithm is again not competitive (this is even true when increasing the number of generated solutions substantially). The results also show that for this type of problems (i.e. $n > m$), the development of specific algorithms is necessary.

Table 3: Results of the comparative study for problems with $n > m$

	GA			TS	SA			Best
(n, m)	GA-R	GA-C	GA-Best		$SA(S)$	$SA(API)$	SA-Best	
(20,10)	-12.37	1.83	1.83	0.78	1.71	0.70	1.82	1.97
	0	28	28	1	21	0	21	50
	58.25	69.96	64.10	12.11	6.91	10.84	8.87	31.61
(30,10)	-15.34	0.82	0.82	0.41	0.71	0.39	0.72	0.85
	0	35	35	1	14	1	15	50
	118.77	173.32	146.05	27.89	19.38	25.26	22.32	72.88
(30,20)	-14.82	0.99	0.99	0.30	0.55	0.59	0.69	1.00
	0	45	45	0	1	4	5	50
	410.64	515.38	463.01	67.56	47.69	63.02	55.36	220.86
(40,10)	-18.40	0.54	0.54	0.30	0.45	0.33	0.48	0.56
	0	38	38	1	9	2	11	50
	202.56	347.89	275.22	47.56	43.35	42.90	43.12	136.85
(40,20)	-22.27	0.48	0.48	0.05	0.23	0.32	0.38	0.51
	0	39	39	0	3	9	12	50
	679.31	1081.19	880.25	120.63	116.61	114.62	115.61	422.47
(40,30)	-13.72	0.63	0.63	0.07	0.24	0.53	0.54	0.66
	0	36	36	0	0	14	14	50
	1786.27	1955.15	1870.71	203.42	176.06	194.61	185.33	863.10
(50,10)	-19.07	0.52	0.52	0.36	0.36	0.39	0.44	0.53
	0	41	41	3	4	4	8	50
	418.23	802.66	610.44	102.22	99.13	91.49	95.31	302.74
(50,20)	-24.50	0.25	0.25	0.03	0.14	0.23	0.25	0.28
	0	31	31	0	4	15	19	50
	1017.24	2054.36	1535.80	191.18	240.33	185.12	212.37	737.64

We note that we have also implemented an ant colony algorithm. The basic idea of such an algorithm comes from the ability of ants to find shortest paths from their nest to food locations. In a combinatorial problem, the ant iteratively builds a solution of the problem. This procedure is conducted using at each step a probability distribution which corresponds to the pheromone trail in real ants. Once a solution is completed, pheromone trails are updated according to the quality of the solution constructed (i.e. cooperation between ants is performed by the common structure which is the shared pheromone matrix). In our implementation, we followed the strategy given e.g. in [3], which turned out to work particularly good for the open shop makespan minimization problem. At each iteration, a number of ants probabilistically construct solutions. In particular, by means of a randomized Beam-Append procedure, rank matrices representing nondelay and active schedules are constructed. Then an iterative improvement procedure is applied to improve the constructed solutions. Finally, some of the constructed solutions are used for performing an update of the pheromone values which aim at increasing the probability to generate high quality solutions. The pheromone values encode for any two operations belonging to the same job or being processed on the same machine the desirability of performing a particular operation before the other one. However, we did not include this algorithm into this study. First, it is difficult to fix a priori a number of e.g. 30000 generated solutions as a stopping criterion and moreover, our prelimimary experience was that the algorithm is in the current version not competitive.

Moreover, we have experimented with hybrid genetic algorithms. To any generated offspring by the genetic algorithm, a fast iterative improvement procedure is applied. Even if these additionally generated solutions are not counted (i.e. 30000 solutions are generated by the genetic algorithms and a multiple of this number by an iterative improvement routine), the improvements in the objective function values are only marginal so that the additional expense is not justified, and we did not include this variant into our final study.

5 Concluding Remarks

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

- In contrast to constructive algorithms, appropriate parameter settings do not strongly depend on the size of n and m (compared with [4]).
- For the simulated annealing algorithm, it is essential to use extremely small initial temperatures (if the initial temperature is too high, it may happen that even the initial objective function value will not be reached again during the search). Among the neighborhoods, the SHIFT neighborhood is superior for most of the problem sizes, often followed by the API neighborhood.
- The tabu seach algorithm is not competitive and turned out to be the worst among the three included metaheuristic algorithms (we have also found that modifying the control parameter of tabu search did not change this observation). The reason for this behavior is that, independently of the difference of the increase in the objective function value, a move to a worse neighbor is accepted. Therefore, all or a considerable part of neighbors in a small neighborhood should be investigated to avoid a large increase in the objective function value.
- The genetic algorithm turned out to be the best algorithm for most problems with $n \leq m$. While for many instances with $n < m$, the (preemptive) lower bound is reached, the average percentage deviations from the lower bound range for square problems with $n = m$ from 6 to 11 % (where the percentage deviations are smaller for the large problems). For $n < m$, the genetic algorithm with 30000 generated solutions is faster than the good constructive beam-insert algorithm from [4].
- In contrast to makespan problems, where usually only problems with $n = m$ are considered, the most difficult mean flow time problems are those with $n > m$. Here the development of further specific algorithms is necessary. On the other hand, problems with $n < m$ are easy in the sense that often heuristic solutions equal or very close to the lower bound are found.

Most iterative 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] Blum, C.: Beam-Aco – Hybridizing Ant Colony Optimization with Beam Search: An Application to Open Shop Scheduling, Comp. Oper. Res., Vol. 32, 2005, 1565 - 1591.

[4] Bräsel, H.; Herms, A.; Mörig, M.; Tautenhahn, T.; Tusch, T.; Werner, F.: Heuristic Algorithms for Open Shop Scheduling to Minmize Mean Flow Time, Part I: Constructive Algorithms, Preprint 30/05, FMA, Otto-von-Guericke-University Magdeburg, 2005.

[5] 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.

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

[7] 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.

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

[9] 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.

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

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

[12] Gupta, J.N.D.; Werner, F.; Wulkenhaar, G.: Two-Machine Open Shop Scheduling with Secondary Criterion, Intl. Trans. Oper. Res., Vol. 10, 2003, 267 - 294.

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

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

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

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

[17] 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.

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

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

[20] Lundy, M.; Mees, A.: Convergence of an Annealing Algorithm, Math. Programming, Vol. 34, No. 1, 1986, 111 - 124.

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

[22] Prins, C.: Competitive Genetic Algorithms for the Open-Shop Scheduling Problem, Math. Meth. Oper. Res., Vol. 52, 2000, 389 - 411.

[23] 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.

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

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

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