Using Simulated Annealing for Open Shop Scheduling with Sum Criteria

Michael Andresen, Heidemarie Bräsel, Mathias Plauschin, Frank Werner[1]

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

Abstract: This work considers the problem of scheduling n jobs on m machines in an open shop environment. For any job, there may be given a release date, a job weight and a due date. It is also assumed that the processing times of all non-preemptive operations of the jobs are given in advance. We consider several objective functions which are all special cases of the problem of minimizing total weighted tardiness subject to given release dates. This contribution continues and extends recent work by Andresen et al. [2] on the heuristic solution of open shop problems with minimizing mean flow time. The focus of this work is on simulated annealing algorithms. Several neighborhoods are suggested and tested together with the control parameters of the algorithm on different problem types. In particular, we investigate the influence of release dates, processing times, job weights, due dates and the ratio of n and m on the selection of an appropriate simulated annealing algorithm by comparing 96 simulated annealing variants relative to each other. Extensive computational results are presented for nearly 8,000 open shop instances with up to 30 jobs and 30 machines, respectively. Moreover, we also perform a comparison of simulated annealing with three variants of a genetic algorithm.

Key words: Open shop scheduling, sum criteria, simulated annealing, genetic algorithm, comparative study

1 Introduction

In this chapter, we consider the open shop scheduling problem which 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. Moreover, 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.

Such an open shop environment arises in many industrial 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 [20] and Prins [21].

For each job J_i, $i = 1, 2, \ldots, n$, there may be given a release date $r_i \geq 0$ which is the earliest possible time when the first operation of this job may start, a weight w_i and a due date $d_i \geq 0$ by which the job should be completed. The processing time of operation (i, j) is denoted as t_{ij}. It is assumed that the processing times of all operations are assumed to be given in advance.

Let C_i be the completion time of job J_i, i.e. the time when the last operation of this job is completed. Traditional optimization criteria are basically partitioned into two types: either the minimization of the maximum term $\max_{1 \leq i \leq n}\{f_i(C_i)\}$ or of the sum $\sum_{i=1}^{n} f_i(C_i)$ is considered,

[1]Corresponding author, E-mail: frank.werner@mathematik.uni-magdeburg.de

where $f_i(C_i)$ denotes the cost arising when job J_i is completed at time C_i. A typical example of an optimization criterion of the first type is the minimization of makespan $C_{max} = \max_{1 \leq i \leq n}\{C_i\}$, while a rather general example of a criterion of the second type is the minimization of total weighted tardiness $\sum_{i=1}^{n} w_i T_i = \sum_{i=1}^{n} w_i \max\{0, C_i - d_i\}$. If release dates of the jobs are given, the latter problem is also denoted as $O|r_i \geq 0| \sum w_i T_i$ which is the most general problem considered in this study.

In the following, we first give a few comments on the open shop problem with minimizing the makespan C_{max} and then a literature review on such problems with sum optimization criteria. Here we discuss only some papers dealing with arbitrary processing times.

Most papers in the literature dealt with the minimization of makespan. In view of the NP-hardness of problem $O||C_{max}$, branch and bound as well as heuristic algorithms have been developed for this problem. Among the exact algorithms, we only mention those given by Laborie [14] and Tamura et al. [25] which were able to solve open benchmark instances from the recent literature. In [14], a complete search for cumulative scheduling based on the detection and resolution of minimal critical sets was performed. The heuristic for selecting such sets relied on an estimation of the related reduction of the search space, where additionally an extension of the search procedure using a self-adapted shaving was proposed. This approach was implemented on the top of classical constraint propagation algorithms. The algorithm was able to solve the remaining 34 open instances out of the 80 instances with up to 10 jobs and 10 machines given by Gueret and Prins [11]. In [25], a method to encode constraint satisfaction problems with integer linear constraints into Boolean satisfiability problems was proposed. The effectiveness of this approach was tested on several benchmark instances for the open shop problem. In particular, this algorithm was able to solve all the 192 benchmark instances of three sets from the literature [9, 11, 24].

Among metaheuristic algorithms, we only discuss two papers presenting simulated annealing algorithms. The first algorithm by Liaw [15] used particular neighborhoods based on up to three pairwise interchanges of two adjacent operations belonging to the same job or being processed on the same machine such that the resulting neighbor satisfies a necessary condition for an improvement of the objective function value. The cooling scheme was of the geometric type and used an initial temperature of 15. The recommended variant had a low temperature reduction scheme (it used a reduction factor of 0.995 for the temperature). The number of iterations with a constant temperature was set to be equal to $30 \cdot n \cdot m$. Taking into account that at least 100 epochs with constant temperatures have been considered per run in [15] (usually even substantially more epochs), this means that e.g. for problems with 20 jobs and 20 machines, at least $30 \cdot 20 \cdot 20 \cdot 100 = 1,200,000$ iterations had to be performed. Moreover, since five runs were made for each instance and in one iteration of the algorithm, up to four neighbors were checked (see neighborhood NH_1 in [15]) and the best neighbor among them was then taken, much more than $6,000,000$ feasible solutions had to be evaluated per instance to get the results presented in [15]. Thus, extremely long runs of simulated annealing were considered in that paper (up to 3.5 hours per single run of an instance with $n = m = 30$). On the other side, the quality of the solutions obtained was comparable to the results obtained by the insertion algorithm combined with beam search given in [6]. In particular, comparing the best results of some beam-insert variant from [6] with the best of the five runs of the simulated annealing algorithm from [15] on the 30 benchmark instances with $n = m \in \{10, 20, 30\}$ given by Taillard [24], the results were equal for 18 instances, 8 times the simulated annealing algorithm was better and four times the beam-insert algorithm was better. A particle swarm algorithm combined with simulated annealing has been given by Yang et al. [28]. For the simulated annealing routine, a very small initial temperature of 2 was used. Computational results have been presented for some benchmark instances with up to 20 jobs and machines given by Taillard [24] (however, the values stated as best known solutions in [28] are rather far away from the real best known solutions so that also the results presented in that paper are not competitive). For a discussion of further exact and

heuristic algorithms for open shop problems with minimizing the makespan, the reader is referred to [2].

There exist only a few papers considering sum criteria. First, we discuss the papers dealing with minimization of mean flow time (or what is the same, total completion time) in an open shop. If preemptions are allowed, the two-machine problem is NP-hard in the ordinary sense [10] while the three-machine preemptive problem is NP-hard in the strong sense [19]. For problem $O|pmtn|\sum C_i$, Bräsel and Hennes [8] 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 non-preemptive problems, Achugbue and Chin [1] proved that problem $O2||\sum C_i$ is NP-hard in the strong sense. Liaw et al. [18] considered the problem of minimizing total completion time with a given sequence of jobs on one machine. This problem is NP-hard in the strong sense even in the case of two machines. 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. Moreover, 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 in 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 was about 4 % while for larger problems, the average percentage deviation from the lower bound was about 8 %.

Bräsel et al. [5] presented a computational study of heuristic constructive algorithms for mean flow time open shop scheduling. They compared 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 [5], it followed 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 algorithm beam-append was superior while for problems with $n < m$, the more time-consuming algorithm beam-insert gave the best results. For the square problems with $n = m$, an overlapping has been observed: For small problems, the beam-insert algorithm was slightly superior while for larger problems, variants of the beam-append algorithm were better. However, the algorithms were rather sensitive with respect to parameter settings.

Andresen et al. [2] presented a simulated annealing and a genetic algorithm for the problem of minimizing mean flow time. They tested their algorithms on problems with up to 50 jobs when performing short runs, where every algorithm may generate 30,000 solutions. It has been found that in contrast to makespan minimization, the hardest problems are those with $n > m$, while for problems with $n < m$, often a lower bound for the corresponding preemptive open shop problem [8] was reached. For the hard problems, it was essential to use a good constructive initial solution and to start the simulated annealing algorithm with an extremely small temperature.

Concerning approximation algorithms with a performance guarantee, the currently best result has been given by Queyranne and Sviridenko [22, 23]. They presented a 5.83-approximation algorithm for the non-preemptive open shop problem of minimizing weighted mean flow time which was based on linear programming relaxations in the operation completion times. This was used to generate precedence constraints. For the preemptive version of this problem, a 3-approximation algorithm has been given.

There exist some papers dealing with open shop problems and other optimization criteria than

makespan and mean flow time. Liaw [16] gave a dynamic programming algorithm for the two-machine preemptive problem of minimizing total weighted completion time. Moreover, a restricted variant was given as a heuristic which was based on pairwise interchanges in the job completion time sequence, i.e. the sequence in which the jobs are ordered according to non-decreasing completion times. Computational experience has shown that the dynamic programming algorithm can handle problems with up to 30 jobs and that the heuristic has an average percentage deviation of less than 0.5 % from the optimal value for these problems.

Liaw [17] presented a branch and bound algorithm for the preemptive open shop problem to minimize total tardiness. Computational results for the two-machine problem showed that the algorithm can handle problems with up to 30 jobs. A heuristic procedure was also given which determined in the q-th iteration the job to be placed in position q in the sequence of the jobs ordered according to non-decreasing completion times. This was done by means of the repeated solution of linear programs. The solutions obtained by the heuristic algorithm had an average deviation of less than 2 % from the optimal value.

Blazewicz et al. [3] considered open shop problems with a common due date, where the goal is to minimize total weighted late work, i.e. the weighted portion processed after the common due date. In addition to some complexity results, a polynomial algorithm for the two-machine problem of minimizing total late work and a pseudo-polynomial algorithm for the corresponding weighted case have been given.

In this chapter, we investigate the application of simulated annealing to open shop scheduling problems with different sum criteria. The most general problem considered in this work deals with the minimization of total weighted tardiness subject to given release dates. Preemptions of operations are forbidden. The remainder of the chapter is organized as follows. In Section 2, we introduce the mathematical model used for describing feasible solutions. In Section 3, we discuss the components of the simulated annealing algorithms considered in our study. A detailed comparative study for the different types of problems is presented in Section 4. In particular, we discuss the influence of the initial solution, the parameters of the algorithms and the problem type in terms of n and m, release dates, processing times, weights and due dates of the jobs and compare the results for short and longer runs. Moreover, a comparison with a genetic algorithm is performed to test the influence of the use of a population. Section 5 contains some conclusions and summarizing recommendations.

2 Basic Notions

Next, we describe the mathematical model for representing feasible solutions for the open shop problem. 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 in a rectangular array, where the operations of job J_i are sequenced in row i and the operations on machine M_j in column j and draw an arc between immediately succeeding operations of the same job or on the same machine, 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).

Example 1: *Let the machine orders of the jobs be chosen as*

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

and, moreover, let the job orders on the machines be as follows:

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

Figure 1 shows the graphs $G(MO)$, $G(JO)$ and $G(MO, JO)$ (with the pair ij of job and machine indices of the operations given inside the vertices).

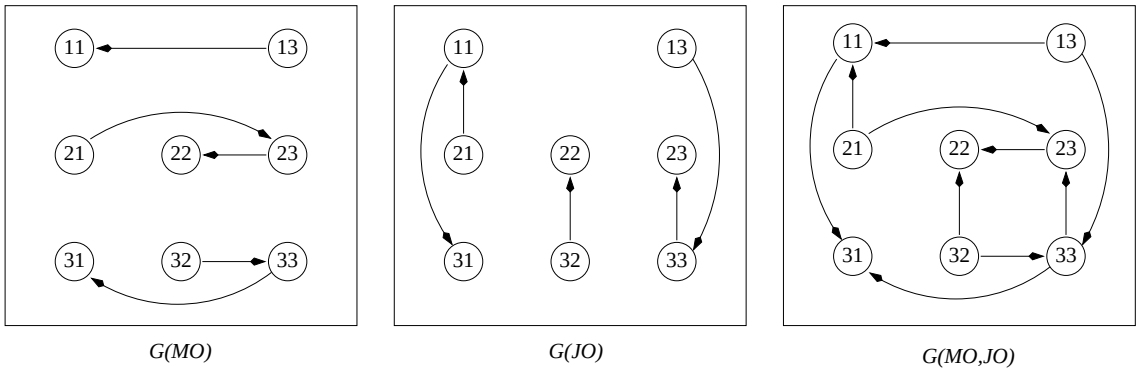

Figure 1: $G(MO)$, $G(JO)$ and $G(MO, JO)$

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 [4, 6, 27], 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 $a_{ij} = k > 1$, the 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 to the vertex (i, j) with (i, j) included in an acyclic digraph $G(MO, JO)$, i.e. a path for which the sum of the vertex weights is maximal, 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.

Example 2: *Consider an open shop problem with $n = 3$ jobs and $m = 3$ machines. Let the release dates of the jobs be given as follows: $r_1 = 3, r_2 = 1, r_3 = 6$. The job weights are $w_1 = 1, w_2 = 4, w_3 = 2$. Moreover, the due dates of the jobs are given as follows: $d_1 = 10, d_2 = 13, d_3 = 18$. The matrix T of the processing times of the operations is given as*

$$T = \begin{pmatrix} 4 & \cdot & 5 \\ 2 & 3 & 3 \\ 5 & 1 & 2 \end{pmatrix}$$

(note that job J_1 has to be processed only on machines M_1 and M_3). Assume that the job and machine orders are chosen as in Example 1. The resulting graph $G(MO, JO)$ corresponds to the rank matrix

$$A = \begin{pmatrix} 2 & \cdot & 1 \\ 1 & 4 & 3 \\ 3 & 1 & 2 \end{pmatrix}.$$

For this instance, we obtain the following schedule C from the given matrix of processing times $T = (t_{ij})$ and the rank matrix $A = (a_{ij})$:

$$C = \begin{pmatrix} 12 & \cdot & 8 \\ 3 & 16 & 13 \\ 17 & 7 & 10 \end{pmatrix}.$$

Thus, we obtain the completion times $C_1 = 12, C_2 = 16$ *and* $C_3 = 17$. *For the optimization criterion* $F = \sum w_i T_i$, *we get the objective function value*

$$\begin{aligned} F &= w_1 \cdot \max\{0, C_1 - d_1\} + w_2 \cdot \max\{0, C_2 - d_2\} + w_3 \cdot \max\{0, C_3 - d_3\} \\ &= 1 \cdot \max\{0, 12 - 10\} + 4 \cdot \max\{0, 16 - 13\} + 2 \cdot \max\{0, 17 - 18\} \\ &= 1 \cdot 2 + 4 \cdot 3 + 2 \cdot 0 = 14. \end{aligned}$$

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. For example, both the permutations

$$OP^1 = \Big((1,3),(3,2),(2,1),(1,1),(3,3),(2,3),(3,1),(2,2)\Big)$$

and

$$OP^2 = \Big((2,1),(3,2),(1,3),(3,3),(1,1),(3,1),(2,3),(2,2)\Big)$$

represent the same sequence graph given as $G(MO, JO)$ in Fig. 1. Considering e.g. the operations to be processed on machine M_3, we have in both permutations OP^1 and OP^2 the same sequence

$$\Big((1,3),(3,3),(2,3)\Big),$$

i.e. both permutations represent the same chosen job order on $M_3 : J_1 \rightarrow J_3 \rightarrow J_2$. This is also true for the remaining job orders and all machine orders of the jobs. Moreover, there exist at least $3! \cdot 2! \cdot 2! \cdot 1! = 24$ permutations of the operations which represent the same job and machine orders as the rank matrix A since there are three operations with rank 1, two operations with rank 2, two operations with rank 3 and one operation with rank 4 in A. In general, the problem of counting possible extensions of a partial order (as it is given e.g. by a rank matrix of a sequence graph) is $\#P$-complete (see [7]).

3 Simulated Annealing Algorithms

In this chapter, we focus on the application of simulated annealing algorithms for solving open shop problems with different sum criteria. One of the major goals of this study consists in finding similarities and differences in the recommendations for the parameters of the algorithms for the different types of problems.

It is well-known that simulated annealing is an enhanced version of local search. 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.

In terms of our open shop model, a basic simulated annealing algorithm starts with generating an initial solution (rank matrix) A. Then a neighbor (rank matrix) A^* of rank matrix A is generated and the difference $\Delta = F(A^*) - F(A)$ in the objective function values of both schedules is calculated. If $\Delta < 0$, the neighbor A^* is accepted as the new starting solution in the next iteration since it has a better function value. If the objective function value does not decrease (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 temperature. This temperature is periodically reduced by a cooling scheme every EL iterations, where EL is a preset parameter called the *epoch length*. As a stopping criterion, one may use e.g. a given number of iterations, a time limit or a given number of iterations without an improvement of the best objective function value. In the first two cases, one must adjust the cooling scheme in such a way that the algorithm stops with a sufficiently small temperature. In our tests, we investigate in particular the influence of the chosen neighborhood and the cooling scheme.

3.1 Neighborhoods

First, we briefly discuss the generation of neighbors of a current solution described by the rank matrix A of a sequence graph $G(MO, JO)$. 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, also consider 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. This means that, in order to generate a neighbor, the rank matrix is modified such that exactly in one job or machine order, two adjacent operations are interchanged. Thus, in order to generate a neighbor, an operation (i, j) is randomly selected and then it is interchanged with the predecessor or successor operation on machine M_j or of job J_i. One of these (at most) four possibilities is randomly chosen. If the pairwise interchange leads to a feasible schedule, it is accepted as the generated neighbor, otherwise another second operation is chosen to perform an adjacent pairwise interchange in a job or machine order. Note that the adjacent pairwise interchange always leads to a feasible solution if the ranks of the two chosen operations differ only by one. As a consequence, if the first operation has been chosen, one of the at most four possibilities for generating a neighbor in the API neighborhood always leads to a feasible solution which follows from the sequence property stated in Section 2.

Moreover, we consider the neighborhood k-API, in which a neighbor is generated from the current sequence graph $G(MO, JO)$, respectively the corresponding rank matrix A, 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). When generating a neighbor, the number $s \in \{1, 2, \ldots, k\}$ of interchanges of two adjacent operations of a job or on a machine is randomly chosen. Note also that the neighborhood used in [15] is a subneighborhood of the 3-API neighborhood, where one or up to four neighbors with specific properties have been generated per iteration.

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. In order to generate a neighbor, an operation (i, j) is randomly chosen. Then another operation belonging to the same job or to be processed on the same machine is selected. Consider the first case (the second one is analogue), and let (i, k) be the other chosen operation. If the rank a_{ik} is smaller than a_{ij}, the rank a_{ij} is modified such that operation (i, j) appears immediately before operation (i, k) (it corresponds to a left shift of machine M_j in the machine order of job J_i). If the rank a_{ik} is larger than a_{ij}, the rank a_{ij} is modified such that operation (i, j) appears immediately after operation (i, k) (it corresponds to a right shift of machine M_j in the machine order of job J_i). Notice that usually the ranks of some other operations have to be modified in order to maintain all established precedence relations. If the chosen shift leads to an infeasible solution, this shift is not performed, and two other operations for performing a shift are

randomly chosen.

Another neighborhood considered is a restricted SHIFT neighborhood denoted as crit-SHIFT. Here only such neighbors in the SHIFT neighborhood are considered which satisfy a necessary condition for an improvement of the makespan value, namely a critical path (i.e. a longest path among all paths ending in a sink of the corresponding sequence graph) in the starting solution is 'destroyed', and there does not exist a path in the graph describing the generated neighbor which contains the same vertices as this critical path of the current starting solution. This neighborhood is based on the so-called block approach originally introduced for shop scheduling problems with makespan minimization. Clearly, the crit-SHIFT neighborhood is a subneighborhood of the complete SHIFT neighborhood.

In our experiments, we always randomly generate one neighbor in the chosen neighborhood in each iteration. In particular, we do not consider such variants which investigate in one iteration all or several neighbors of the current starting solution in a particular neighborhood and select the best neighbor as the generated one to which the acceptance criterion of simulated annealing is applied.

Example 3: *We illustrate the API, SHIFT and crit-SHIFT neighborhoods discussed above by the following example with* $n = 3$ *and* $m = 4$*. Let the current sequence graph be described by the rank matrix*

$$A = \begin{pmatrix} 2 & 1 & 7 & 3 \\ 3 & 2 & 6 & 1 \\ 4 & 3 & \mathbf{5} & 2 \end{pmatrix}.$$

Assume that operation $(3,3)$ *with* $a_{33} = 5$ *(given in bold face above) has been chosen randomly for generating a neighbor in the API neighborhood. This operation* $(3,3)$ *is contained*
(a) in the machine order of job $J_3 : M_4 \to M_2 \to M_1 \to M_3$ *and*
(b) in the job order on machine $M_3 : J_3 \to J_2 \to J_1$*.*
Using this operation $(3,3)$*, one can generate two neighbors in the API neighborhood (note that we cannot generate four neighbors since machine* M_3 *is the last one in the machine order of* J_3 *and* J_3 *is the first job in the job order on machine* M_3*). If we interchange the machines* M_1 *and* M_3 *in the machine order of job* J_3 *(see (a) above), we get the rank matrix*

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

as the generated neighbor in the API neighborhood. If we interchange the jobs J_3 *and* J_2 *in the job order on machine* M_3 *(see (b) above), we get the rank matrix*

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

as the generated neighbor.

Next, we consider the generation of a neighbor in the SHIFT neighborhood. Let again $(3,3)$ *be the operation chosen first and assume that operation* $(3,2)$ *is selected as the second operation. Operation* $(3,2)$ *is performed earlier and belongs to job* J_3 *too. This means that operation* $(3,3)$ *will be shifted left in the machine order of job* J_3 *so that it is rescheduled directly before operation* $(3,2)$*. This gives the rank matrix*

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

of the generated neighbor (notice that the entries of some operations have to be changed in order to maintain all precedence relations). Assume now that operation $(1,3)$ *is chosen as the second operation. This operation is performed later than* $(3,3)$ *on the same machine which means that operation* $(3,3)$ *is shifted right in the job order on machine* M_3 *so that it is rescheduled directly after operation* $(1,3)$*. This gives the rank matrix*

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

of the generated neighbor. So both rank matrices A^3 *and* A^4 *describe feasible neighbors of rank matrix* A *in the SHIFT neighborhood.*

Now assume that the processing times of all operations are equal to one. In this case, the makespan value of rank matrix A *is equal to 7, and a critical path contains e.g. the vertices*

$$(1,3),(2,3),(3,3),(3,1),(2,1),(2,2),(2,4)$$

(note that the critical path is not uniquely determined for this instance). In this case, both rank matrices A^3 *and* A^4 *are also a neighbor of rank matrix* A *in the crit-SHIFT neighborhood (because in both cases operation* $(3,3)$ *is shifted to a position 'outside' the chosen critical path. In fact, both neighbors lead indeed to an improvement of the makespan value:* $C_{max}(A^3) = 5$ *and* $C_{max}(A^4) = 6$*. Considering e.g. the objective function* $F = \sum C_i$ *and assuming that all release dates are equal to zero, the starting solution described by* A *has the function value* $F = 7 + 6 + 5 = 18$*, and both generated neighbors lead to an improvement of the objective function value:* $F(A^3) = 5 + 4 + 5 = 14$ *and* $F(A^4) = 5 + 4 + 6 = 15$*.*

3.2 Cooling schemes

Typical cooling schemes used in a simulated annealing algorithm are a geometric, a Lundy-Mees and a linear reduction scheme. The three cooling schemes have been tested for open shop problems with mean flow time minimization in [2]. It has been found that often the geometric scheme is slightly superior. In most other applications to scheduling problems, a geometric cooling scheme is also preferred. Therefore, in the following we test exclusively geometric schemes.

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 $0 < \alpha < 1$.

In our experiments we fix the initial temperature T^0, the epoch length EL and set the temperature reduction factor α in such a way that the final temperature is close to zero (we always use $T^{end} = 0.01$ as the final temperature) taking into account that in our study, the maximal number of generated solutions is settled in advance and therefore, the maximal number of epochs with a constant temperature is fixed. Based on the experiments in [2], we fix the epoch length as $EL = 100$.

In addition to the usual procedure of one cooling cycle, we also consider variants of simulated annealing with several cooling cycles in one run, 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. This requires that the (maximal) number of solutions to be generated in one run is settled in advance. The number CC denotes the number of cooling cycles in one run of the algorithm.

4 Computational Results

In this section, we present the computational results with the tested algorithms. First, we describe the generation of the open shop instances in Section 4.1. Then we give some comments on the generation of the initial solution in Section 4.2. In Section 4.3, we describe the design of the comparative study. A detailed comparison of the simulated annealing algorithms is made in Section 4.4. Finally, we compare the fast simulated annealing algorithms with genetic algorithms from [2] in Section 4.5.

4.1 Generation of Instances

For the comparative study, we consider all pairs (n, m), with $n \in \{10, 15, 20, 30\}$ and $m \in \{10, 15, 20, 30\}$ yielding a total of 16 combinations of m and n. In particular, there are four pairs (n, m) with $n = m$, six pairs with $n > m$ and six pairs with $n < m$.

For each pair (n, m), we generated several problem types differing in the job weights, the processing times, the release dates and the due dates.

For the job weights, we considered the following two variants:

w1: All weights are equal to one: $w_i = 1$ for $i = 1, 2, \ldots, n$.

w2: The weights are uniformly distributed integers from the interval $[1, 10]$.

For the processing times of the operations, we also consider two variants:

t1: The processing times are uniformly distributed integers from the interval $[1, 100]$.

t2: The processing times are uniformly distributed integers from the interval $[35, 66]$.

For the above two cases, we have chosen two uniform distributions having the same expectation value of 50.5, but in the second case the standard deviation is substantially smaller, namely a bit less than one third of the standard deviation in the first case.

For the release dates, we consider two different variants

r1: All release dates are equal to zero: $r_i = 0$ for $i = 1, 2, \ldots, n$.

r2: The release dates are uniformly distributed integers from the interval $[0, r_{max}]$, where

$$r_{max} = \frac{1}{2n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ij}.$$

In case r2, the value of r_{max} has been settled in such a way that it is equal to the half of the average total processing time of a job.

For the due dates of the jobs, we considered the following three variants:

d1: The due dates of all jobs are equal to zero: $d_i = 0$ for $i = 1, 2, \ldots, n$ (in this case, we have the objective function $\sum w_i C_i$ or its special case $\sum C_i$).

d2: The due dates of the jobs are generated as follows:

$$d_i = r_i + TF \cdot \sum_{j=1}^{m} t_{ij}, \quad i = 1, 2, \ldots, n$$

with the tightness factor $TF = 1.0$ for problems with $n \leq m$ and $TF = 1.25$ for the problems with $n > m$.

d3: The due dates of the jobs are generated as follows:

$$d_i = r_i + TF \cdot \sum_{j=1}^{m} t_{ij}, \quad i = 1, 2, \ldots, n$$

with the tightness factor $TF = 1.1$ for the problems with $n \leq m$ and $TF = 1.5$ for the problems with $n > m$.

While for the second variant d2 due dates are more tight, they are more lose for the third variant d3. We have found that problems with $n \leq m$ and $TF \geq 1.2$ tend to become rather easy in the sense that often the best of the constructive procedures has an objective function value of zero which means that the optimal solution has already been found. On the other hand, larger tightness factors are of interest for the problems with $n > m$. So we decided to use different tightness factors for the problem types with $n \leq m$ and $n > m$ under consideration.

Each problem type is described by a 4-tuple (w,t,r,d), For instance, the 4-tuple (w1,t1,r1,d1) characterizes the open shop problem $O||\sum C_i$ of minimizing mean flow time when all release dates are equal to zero and processing times are taken from the interval $[1, 100]$ (this was the only problem type investigated in [2]). In our tests, we considered problems of all possible 4-tuples. This gives altogether $2^3 \cdot 3 = 24$ different types of problems. For each of these types and any of the 16 pairs (n, m), we generated 20 instances, giving a total of $24 \cdot 16 \cdot 20 = 7,680$ instances.

4.2 Generation of the Initial Solution

Often initial solutions for shop scheduling problems are obtained by generating active or nondelay schedules. A schedule is called *active* if no operation can be started earlier without changing the underlying sequence graph and 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, and any active schedule is a semiactive one. Similarly as in [5] for mean flow time minimization, we have found in initial tests that nondelay schedules are superior to active schedules for the problems under consideration. Therefore, we exclusively used the generation of nondelay schedules as fast constructive procedures.

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

In our tests, we have used the following priority dispatching rules for generating a nondelay schedule:

- RND (an operation is randomly selected)
- FCFS (first come first served, i.e. the operation that entered the queue first is chosen),
- SPT (shortest processing time),
- WSPT (weighted shortest processing time, i.e. the operation with smallest quotient t_{ij}/w_i is chosen) and
- LPT (longest processing time),
- EDD (earliest due date)

4.3 Design of the Comparative Study

For each of the instances generated as described in Section 4.1, we first tested the different simulated annealing variants. In particular, we have used the following simulated annealing algorithms, differing in the construction of the initial solution, the stopping criterion, the neighborhood and the cooling scheme.

Initial Solution: We consider one variant with a weak initial solution and one variant with a better initial solution:

I1: The initial solution is determined by the generation of a nondelay schedule according to the rule RND.

I2: The initial solution is determined as the best nondelay schedule obtained by the application of all priority dispatching rules mentioned in Section 4.2.

Stopping criterion: We consider two variants with an a priori fixed number of iterations (i.e. the number of generated solutions) and additionally one variant, where the algorithm stops if no improvement of the best function value has been obtained for a certain number of iterations. In particular, we use the following stopping criteria:

S1: The algorithm performs 30,000 iterations.

S2: The algorithm performs 200,000 iterations.

S3: The algorithm performs at most 200,000 iteration but stops, if no improvement of the best objective function value has been obtained for 10,000 iterations.

Neighborhoods: The simulated annealing algorithm uses one of the four neighborhoods discussed in Section 3:

N1: The algorithm uses the API neighborhood.

N2: The algorithm uses the 3-API neighborhood.

N3: The algorithm uses the SHIFT neighborhood.

N4: The algorithm uses the crit-SHIFT neighborhood.

Cooling Scheme: The geometric cooling scheme tested in our algorithms is characterized by the initial temperature and the number of cooling cycles. For the initial temperature, we used the following two variants:

IT1: The initial temperature is equal to 2.

IT2: The initial temperature is equal to 15.

For the number of cooling cycles, we considered the following two variants:

CC1: The number of cooling cycles is equal to 1.

CC2: The number of cooling cycles is equal to 5.

In our tests, we considered any possible combination of an initial temperature and the number of cooling cycles, yielding four different cooling schemes.

Since we fixed the epoch length as $EL = 100$, this means that for variant CC1, the number of epochs is equal to 300 for stopping criterion S1. Moreover, since we fixed the final temperature as $T^{end} = 0.01$, the reduction factor α in the geometric scheme is equal to $\alpha = 0.983$ for an initial temperature of 2 corresponding to IT1 and $\alpha = 0.976$ for an initial temperature of 15 corresponding to IT2. For variant CC2, the number of epochs per cooling cycle is equal to 60. As a consequence, in each run the reduction factor α is equal to $\alpha = 0.916$ for an initial temperature of 2 and $\alpha = 0.887$ for an initial temperature of 15.

For the long runs with stopping criterion S2, the number of epochs is 2,000 (for S3, the maximal number of epochs is 2,000). Therefore, for variant CC1, the reduction factor α is equal to $\alpha = 0.998$ for an initial temperature of 2 and $\alpha = 0.997$ for an initial temperature of 15. For variant CC2, the number of epochs per cooling cycle is equal to 400. As a consequence, the reduction factor α is equal to $\alpha = 0.987$ for an initial temperature of 2 and $\alpha = 0.982$ for an initial temperature of 15.

A particular simulated annealing variant is described by a 5-tuple. For instance, algorithm (I2,S2,N3,IT1,CC2) means that the the best constructive solution is taken as initial solution, 200,000 iterations are performed, the SHIFT neighborhood is used and the cooling scheme is characterized by an initial temperature of 2 and five cooling cycles. We have run simulated annealing for any possible combination of a stopping criterion, use of a particular initial solution, a neighborhood and a cooling scheme. This yields $3 \cdot 2 \cdot 4 \cdot 4 = 96$ different simulated annealing algorithms.

Concerning computational times we only mention that for the large problems with $n = m = 30$, the average computational time per instance for a variant with stopping criterion S2 is 198.3 s on an AMD Athlon XP 3200+. For smaller problems with $n = 10$ and $m = 20$, this average computational time for a long run per instance is 19.8 s while for the corresponding problems with $n = 20$ and $m = 10$, this average time is 22.2 s. We also note that one computer of this type would require about 4,250 hours to perform all runs done in our study.

4.4 Comparative Study of Simulated Annealing

Before comparing the simulated annealing variants, we give a few comments on the performance of the constructive algorithms. For $n < m$, we have found that the LPT rule is clearly the best algorithm. It is followed by the rules RND, SPT and WSPT which yield solutions of approximately the same quality. In particular, the rather good quality of the RND rule is surprising. This rule is clearly better than the ECT and FCFS rules which are the weakest constructive algorithms for problems with $n < m$. Problems with d3 tend to become easy. In this case, the majority of the dispatching rules yield the best constructive solutions, and many objective function values are equal or very close to zero. For the problems with $n > m$, the LPT rule works bad. The best results have been obtained with the EDD, WSPT and FCFS rules. If $n = 30$ and $m = 10$, the WSPT rule works good for problems with w2. However, for problems with w1, the FCFS rule is clearly the best for problems with d1 and the EDD rule is superior for the problems with d2 and d3. The observed trends are most obvious for a large ratio of n/m (although, if the ratio n/m decreases, the observations are similar but not so strong). For problems with $n = m$, all dispatching rules contribute best values. In general, there is an overlapping of the observations for the problems with $n < m$ and $n > m$. We observed that the EDD rule is good for problems with r1 and d3 while the LPT rule works well for problems with r2.

For evaluating the 96 simulated annealing variants, we use a *performance index* defined as follows. Let F^A be the heuristic function value obtained for a particular instance by algorithm A, F^{CON} be the best function value obtained by some of the constructive procedures mentioned in Section 4.2, and F^{BEST} be the best function value obtained by one or several of the 96 tested simulated

annealing variants. In the case of $F^{CON} > 0$, the performance index PI of algorithm A for this particular instance is given by

$$PI = \begin{cases} \frac{F^{CON} - F^A}{F^{CON} - F^{BEST}} \cdot 100 & \text{if } F^{BEST} \leq F^A < F^{CON} \\ 0 & \text{if } F^A \geq F^{CON} \end{cases}$$

If $F^{CON} = 0$, we define the performance indices of all simulated annealing algorithms with the corresponding constructive initial solution to be equal to 100. Moreover, let $PI(k)$ be the percentage of the instances, for which a particular algorithm has obtained a performance index of at least k. That is, $PI(95) = 80$ means that the algorithm under consideration has obtained a performance index greater than or equal to 95 for 80 % of the instances. In the following evaluations, we consider the performance indices $PI(95)$ (which stands for an excellent performance) and $PI(75)$ (which stands for a good performance of the particular algorithm).

First, we give some general observations from our study. Then we discuss separately the results for the problems with $n < m$, $n = m$ and $n > m$ in more detail.

General Observations:

As a general observation we have found that the ratio of n and m influences the hardness of the problems. Among the problem data, the range of the processing times and the job weights have in particular an influence on the selection of an appropriate algorithm or the quality of the results, while release dates and due dates have only minor influence. Therefore, the recommendations in the following sections do not strongly depend on different due dates and release dates. Hence, at most four algorithms (for any combination of weights and processing times) are suggested for every stopping criterion S1, S2 and S3, respectively. On the other side, the range of due dates influences the range of the objective function values and their possible percentage improvements.

The use of the best constructive algorithm leads to better results with the simulated annealing algorithms than the use of only a randomly generated initial solution. The choice of an appropriate neighborhood turns out to be substantial for the quality of the results. An appropriate initial temperature is at least for certain problem types important. In particular, some problems with unit weights require a low initial temperature when using short runs while for most problems with w2, the results with the different initial temperatures do not differ very much. From an overall point of view, the number of cooling cycles per run has only a small influence on the quality of the results. In general, algorithms with variant CC2 turn out to be a bit superior to those with CC1.

If one looks for an overall variant that performs well, we can recommend the algorithms with a good initial solution, the use of the SHIFT neighborhood and a cooling scheme with a low initial temperature and one or five cooling cycles.

Problems with n < m:

In Table 1, we summarize some results for the problems with $n < m$. The rows refer to the 24 different problem types described by a 4-tuple (w,t,r,d). In column 2, we present the average objective function value F^{CON} (rounded to integers) of the best constructive algorithm taken over all instances of the six pairs (n, m) with $n < m$. In column 3, the average percentage improvement PERC of the best function value obtained by the 96 simulated annealing variants over the function value of the initial solution is given. In the remaining columns, we present first the average performance index (columns AVG) of the corresponding algorithm and then the values $PI(95)$ and $PI(75)$ (columns 95/75) for the recommended variants with stopping criterion S1 (Alg 1), criterion S2 (Alg 2) and criterion S3 (Alg 3), respectively. In particular, based on the experiments and the discussion below, we have chosen the following algorithms:

Alg 1: algorithm (I2,**S1**,N3,IT1,CC2) for problems with t1; algorithm (I2,**S1**,N1,IT2,CC2) for problems with t2;

Alg 2: algorithm (I2,**S2**,N3,IT1,CC1) for problems with w1; algorithm (I2,**S2**,N3,IT2,CC2) for problems with w2;

Alg 3: algorithm (I2,**S3**,N3,IT1,CC2) for problems with t1; algorithm (I2,**S3**,N1,IT1,CC1) for problems with w1 and t2; algorithm (I2,**S3**,N1,IT2,CC2) for problems with w2 and t2.

Table 1: Results for problems with $n < m$

(w,t,r,d)	F^{CON}	PERC	Alg 1		Alg 2		Alg 3	
			AVG	95/75	AVG	95/75	AVG	95/75
(w1,t1,r1,d1)	17,041	0.32	71	25/53	93	72/91	71	21/55
(w1,t1,r1,d2)	160	36.2	70	22/49	93	66/91	71	22/51
(w1,t1,r1,d3)	2	97.5	100	100/100	100	100/100	100	100/100
(w1,t1,r2,d1)	21,238	0.23	71	22/50	90	63/88	69	22/58
(w1,t1,r2,d2)	166	39.3	70	21/48	90	60/87	70	23/53
(w1,t1,r2,d3)	0.3	98.3	100	100/100	100	100/100	100	100/100
(w1,t2,r1,d1)	17,009	0.55	56	13/30	78	46/64	49	15/29
(w1,t2,r1,d2)	161	62.9	54	14/28	75	40/63	50	16/28
(w1,t2,r1,d3)	0	100	100	100/100	100	100/100	100	100/100
(w1,t2,r2,d1)	21,156	0.33	58	20/41	78	45/70	53	24/33
(w1,t2,r2,d2)	118	69.8	54	21/35	77	44/67	51	19/31
(w1,t2,r2,d3)	0	100	100	100/100	100	100/100	100	100/100
(w2,t1,r1,d1)	94,637	0.32	68	20/45	90	58/88	70	18/58
(w2,t1,r1,d2)	835	41.6	68	21/47	91	63/88	71	22/60
(w2,t1,r1,d3)	8	98.0	99	98/98	100	99/100	100	99/100
(w2,t1,r2,d1)	118,132	0.22	66	19/45	88	53/81	67	22/47
(w2,t1,r2,d2)	838	39.7	66	23/45	88	56/81	64	20/45
(w2,t1,r2,d3)	1	100	100	100/100	100	100/100	100	100/100
(w2,t2,r1,d1)	94,475	0.56	61	18/41	82	45/73	58	15/38
(w2,t2,r1,d2)	847	68.8	56	18/33	83	43/75	53	13/31
(w2,t2,r1,d3)	0	100	100	100/100	100	100/100	100	100/100
(w2,t2,r2,d1)	117,665	0.32	58	26/40	83	51/73	55	21/38
(w2,t2,r2,d2)	594	73.7	56	24/39	79	43/73	55	23/38
(w2,t2,r2,d3)	0	100	100	100/100	100	100/100	100	100/100
average			75.1	47/61	90.0	69/85	74.1	46/62

From Table 1 we see that there is a large range of percentage improvements over the constructive algorithm for the particular types of problems. For problems with d1 (i.e. minimization of mean flow time or its weighted version), the average percentage improvements are very small (always less than 1 %). This corresponds to the observation for problem type (w1,t1,r1,d1) in [2], where it has been found that the solutions obtained by constructive algorithms are already almost optimal and often even a lower bound for the corresponding preemptive problem has been reached. On the other hand, problems with d3 tend to be easy in the sense that the initial solution has already a function value close to zero. Note that for these problems, the performance indices of Alg 1 - Alg

3 are strongly influenced by the large number of instances with $F^{CON} = 0$, where the performance index is 100 per definition. For problems with d2, substantial average percentage improvements over the initial solution have been obtained. For these problems, it is remarkable that rather small objective function values have been obtained by the best simulated annealing algorithms although the tightness factor $TF = 1$ leads to tight due dates. As a consequence, there are only short waiting times of the jobs in the best solutions found. Among all particular combinations of a problem type (w,t,r,d) and a pair (n, m), we observe that the absolute improvements of the average function values obtained by the best simulated annealing variant over the average values of the initial solutions are up to 130 units for problems with w1 and up to 800 units for problems with w2. For problems with w1 and d1, they are typically around 50 units. However, from [2] it follows that often the heuristic solution is equal or close to a lower bound for the optimal value of a problem of type (w1,t1,r1,d1).

Moreover, the use of the SHIFT neighborhood is clearly superior for the problems with t1. In contrast, for most problem types with t2, both the API and 3-API neighborhoods are superior to the SHIFT neighborhood when using shorter runs with stopping criteria S1 and S3, and this tendency increases with the problem size. In addition, the API neighborhood is slightly superior to the 3-API neighborhood. We observed that the superiority of the SHIFT in comparison with the two API-based neighborhoods is larger for problems with t1 than the superiority of the API-based neighborhoods over the SHIFT neighborhood for those with t2 for short runs. However, the SHIFT neighborhood becomes the single best for the long runs with stopping criterion S2. The variants with stopping criterion S2 yield the best results, often followed by the algorithms with S1 and finally those with S3 (an explanation is given in the next paragraph). Variants with an initial temperature IT1 tend to be superior to those with the initial temperature IT2, in particular for problems with w1, t1, for which they are substantially better (for an arbitrary stopping criterion). Moreover, for most problems algorithms with five cooling cycles work slightly better than variants using only one cooling cycle. However, for the long runs with stopping criterion S2, the use of one cooling cycle is slightly better for the problems with w1.

Next, we discuss the number of iterations executed in the case of stopping criterion S3. First, taking the average number of generated solutions over all problems with $n < m$, this number is up to 25 % for algorithms with N3 and only up to 12 % for algorithms with N1, N2 and N4. In particular, for neighborhood N4 the algorithm stops very quickly. For problems with w1, even for neighborhood N3 the algorithm stops after 5 % when using the larger initial temperature IT2 and CC1. This means that for an initial temperature of 15, usually no improvements over the function value of the initial solution are obtained. The percentage of generated solutions is also larger for problems with t1 in comparison with the problems with t2. The largest percentage of generated solutions was obtained for problem type (w2,t1,r1,d1) as well as $n = 20$ and $m = 30$ using N3, IT1, CC2 and a random initial solution, where 48 % of the iterations were executed. Comparing stopping criteria S1 and S3, we observe that only for the SHIFT neighborhood usually more than 30,000 solutions were generated for S3 while for the other neighborhoods, typically only around 20,000 solutions have been generated.

Problems with n = m:

Some results for the problems with $n = m$ are given in Table 2. The meaning of the rows and columns is the same as in Table 1. Based on the experiments and the discussion below, the following algorithms for the stopping criteria S1, S2 and S3, respectively, are included in Table 2:

Alg 1: algorithm (I2,**S1**,N1,IT2,CC2) for problems with w1 and t2; algorithm (I2,**S1**,N3,IT1,CC2) for all other problems;

Alg 2: algorithm (I2,**S2**,N3,IT1,CC1) for problems with w1, algorithm (I2,**S2**,N3,IT1,CC2) for problems with w2 and t1; algorithm (I2,**S2**,N3,IT2,CC2) for problems with w2 and t2.

Alg 3: algorithm (I2,**S3**,N1,IT2,CC1) for problems with t2; algorithm (I2,**S3**,N3,IT1,CC2) for problems with w1 and t1; algorithm (I2,**S3**,N3,IT1,CC1) for problems with w2 and t1.

Table 2: Results for problems with $n = m$

(w,t,r,d)	F^{CON}	PERC	Alg 1		Alg 2		Alg 3	
			AVG	95/75	AVG	95/75	AVG	95/75
(w1,t1,r1,d1)	22,362	0.60	39	0/1	68	14/38	41	1/10
(w1,t1,r1,d2)	1,690	7.1	46	4/15	76	30/54	46	3/15
(w1,t1,r1,d3)	241	27.2	46	5/10	73	24/59	43	1/15
(w1,t1,r2,d1)	26,962	0.55	46	4/11	77	30/55	52	8/19
(w1,t1,r2,d2)	1,314	8.3	45	3/9	77	23/61	47	1/16
(w1,t1,r2,d3)	171	41.3	50	4/18	76	30/59	52	4/25
(w1,t2,r1,d1)	21,928	0.80	40	4/14	60	21/35	42	8/18
(w1,t2,r1,d2)	1,360	9.4	40	3/9	66	25/45	44	14/20
(w1,t2,r1,d3)	53	79.3	45	24/34	69	35/53	40	20/26
(w1,t2,r2,d1)	26,682	0.73	35	1/6	61	19/35	38	4/11
(w1,t2,r2,d2)	1,106	13.4	33	1/9	59	20/33	31	3/10
(w1,t2,r2,d3)	51	93.7	57	25/31	73	36/54	38	11/23
(w2,t1,r1,d1)	124,053	0.77	45	1/13	78	28/61	50	5/20
(w2,t1,r1,d2)	8,974	9.0	44	0/9	76	19/55	52	4/24
(w2,t1,r1,d3)	1,122	36.0	55	4/26	78	28/61	51	8/33
(w2,t1,r2,d1)	150,066	0.53	46	1/11	76	23/59	56	3/25
(w2,t1,r2,d2)	7,140	8.9	43	0/9	75	19/58	51	0/20
(w2,t1,r2,d3)	790	40.9	57	4/29	84	40/78	57	5/35
(w2,t2,r1,d1)	121,967	1.33	45	1/16	80	26/63	39	5/14
(w2,t2,r1,d2)	7,404	17.1	48	4/15	79	41/68	42	3/19
(w2,t2,r1,d3)	258	78.0	54	24/34	74	38/58	49	26/38
(w2,t2,r2,d1)	148,458	1.09	45	4/18	80	33/68	34	4/15
(w2,t2,r2,d2)	5,965	20.8	46	5/18	82	38/63	32	0/9
(w2,t2,r2,d3)	237	94.7	50	11/30	72	34/64	44	15/25
average			45.7	6/16	73.6	28/56	44.6	6/20

For the problems with w1 and d1, the average percentage improvements are smaller than 1 %. These percentage improvements are larger for problems with w2 and t2. Here they are up to 2.53 % for problem type (w2,t2,r1,d1) and $n = m = 10$. For problem type (w1,t2,r2,d3), average percentage improvements of more than 90 % have been obtained and the final average objective function values for the instances of the particular pairs (n, m) are between 0 and 10 so that many problems have been solved to optimality. When comparing the average function values of the initial solutions with the average values by the best simulated annealing solutions, the absolute improvements are up to 200 units for problems with w1 and up to 1,500 units for the problems with w2.

Among the neighborhoods, the SHIFT neighborhood is clearly on the first place followed by the 3-API neighborhood (which is, however, substantially worse) when considering the results for all pairs (n, m). The crit-SHIFT neighborhood works extremely weak. The use of a small initial temperature is slightly superior. In particular, for the long runs with neighborhood N3 and stopping criterion S2, often the large initial temperature combined with one cooling cycle works weak for problems with w1. In general, the use of five cooling cycles is slightly superior in most cases. As for

the problems with $n < m$, for the long runs with stopping criterion S2, the use of one cooling cycle is better for the problems with w1 while the use of five cooling cycles is better for w2.

When looking at the instances of the particular pairs (n, m), we can note that there is a tendency that with an increasing number of jobs, the API neighborhood becomes more and more competitive to the SHIFT neighborhood. For the problems with $n = m = 20$, the API neighborhood becomes superior for the problems with w1 when using short runs. For the problems with $n = m = 30$, the API neighborhood is also superior for most problem types when using short runs and even for problems with w1 when using long runs. This corresponds to the observation in [2], where the API neighborhood became superior for the short runs with problem type (w1,t1,r1,d1) and $n \geq 20$.

Moreover, for most problems with w1 and t2, it turned out that in the case of short runs with stopping criterion S1, the recommended variant (I2,S1,N1,IT2,CC2) works not so good for small problems with $n = 10$ while the use of the API neighborhood is clearly superior for the larger problems with $n \geq 20$.

In addition, the variant (I2,S3,N1,IT2,CC1) was recommended for problems with w2 and t2 from an overall point of view when using S3. However, for these problems the performance depends also on the existence of release dates. In general, the API neighborhood is better for the problems with r1 while the SHIFT neighborhood is better for the problems with r2 when using S3 (the latter differs from the recommendation for Alg 3 made from an overall point of view for the corresponding group of problem types). Nevertheless, in contrast to the above comment, for the small problems with $n = m = 10$ and r1, the SHIFT neighborhood is superior while for the large problems with $n \geq 20$ and r2, the API neighborhood is clearly superior. This coincides with the general observation that the SHIFT neighborhood is often substantially better for small problems while the API neighborhood becomes better for the large problems.

For stopping criterion S3, the number of performed iterations slightly increases with the problem size. For problems with $n = m = 20$, up to 44 % of the iterations have been performed when using the SHIFT neighborhood. The largest number of iterations were performed for problems with t1. However, these numbers are substantially smaller for the other neighborhoods. In particular, for the small problems with $n = m = 10$, the number of performed iterations is roughly only the half of those for the large problems but in general, these percentages for the large problems are still rather low. For a substantial number of problems with d3, an objective function value of zero has been obtained for the long runs with the SHIFT neighborhood.

Problems with n > m:

Some results for the problems with $n > m$ are summarized in Table 3. The meaning of the rows and columns is the same as in Table 1. Based on the experiments and the discussion below, we have chosen the following algorithms for the stopping criteria S1, S2 and S3, respectively:

Alg 1: algorithm (I2,**S1**,N3,IT2,CC2) for problems with w2 and t1, algorithm (I2,**S1**,N3,IT1,CC1) for all other problems;

Alg 2: (I2,**S2**,N3,IT2,CC2) for problems with w2; algorithm (I2,**S2**,N3,IT1,CC2) for problems with w1 and t1, algorithm (I2,**S2**,N3,IT1,CC1) for problems with w1 and t2;

Alg 3: algorithm (I2,**S3**,N3,IT1,CC1) for problems with w1 and t2 as well as w2 and t1; algorithm (I2,**S3**,N3,IT1,CC2) for all other problems.

For the problems with d2 and d3, the average percentage improvements are much smaller than for the problems with $n \leq m$. In particular, for the problems with $n = 30$ and $m = 10$, these percentages are less than 1.4 % for the problems with w1. For the corresponding problems with w2, these average percentages are between 3.2 and 4.8 %. In terms of the objective function values, the

absolute improvements of the function values are larger than for the problems with $n \leq m$. More precisely, the absolute improvement of the average function value obtained by the best simulated annealing variant over the average value of the initial solution among all particular combinations of a problem type (w,t,r,d) and a pair (n, m) is up to 230 units for problems with w1 and up to 2,800 units for problems with w2. In particular, for problem type (w1,t2,r2,d3) and the instances with $n = 30$ and $m = 20$, the average function value of the initial solution is 233.5, but the average function value of the best simulated annealing solution is only 3.4.

In general, it can be observed that the performance indices of the algorithms using the SHIFT neighborhood and stopping criterion S2 are consistently rather large. One can also note that long runs with the API-based and crit-SHIFT neighborhoods do not reach the quality of short runs with the SHIFT neighborhood. As an exception, the crit-SHIFT neighborhood works (surprisingly) good for the problems with $n = 15$ and $m = 10$ as well as $n = 30$ and $m = 20$ for the problems with d3 (sometimes even better than the SHIFT neighborhood). Variants with a low initial temperature are mostly superior, and this superiority is stronger than for the problems with $n \leq m$. This becomes particularly obvious for the problems with w1. For the short runs with stopping criteria S1 and S3, often the use of one cooling cycle can be recommended.

Table 3: Results for problems with $n > m$

(w,t,r,d)	F^{CON}	PERC	Alg 1		Alg 2		Alg 3	
			AVG	95/75	AVG	95/75	AVG	95/75
(w1,t1,r1,d1)	26,932	0.69	49	0/9	86	41/76	59	3/22
(w1,t1,r1,d2)	5,142	2.9	48	3/13	86	45/78	53	8/24
(w1,t1,r1,d3)	2,935	13.6	54	16/25	86	51/75	62	21/35
(w1,t1,r2,d1)	29,207	0.63	52	0/10	87	37/80	65	8/28
(w1,t1,r2,d2)	3,953	5.2	50	1/13	85	43/77	57	5/24
(w1,t1,r2,d3)	2,070	28.4	63	27/39	91	63/83	68	32/47
(w1,t2,r1,d1)	26,417	0.75	44	1/13	78	33/68	37	0/11
(w1,t2,r1,d2)	5,558	4.4	46	5/19	72	32/56	35	5/12
(w1,t2,r1,d3)	3,180	32.9	44	18/23	71	36/54	38	18/22
(w1,t2,r2,d1)	28,692	0.66	44	5/11	73	23/58	38	3/7
(w1,t2,r2,d2)	4,069	12.2	38	3/10	73	27/53	28	1/9
(w1,t2,r2,d3)	2,304	51.9	60	31/38	87	64/75	54	30/38
(w2,t1,r1,d1)	133,174	1.03	56	0/16	88	35/85	73	8/46
(w2,t1,r1,d2)	21,466	6.9	50	0/10	82	32/73	65	11/45
(w2,t1,r1,d3)	11,848	16.1	52	13/21	85	43/72	64	18/38
(w2,t1,r2,d1)	151,172	0.98	52	0/8	87	32/84	67	5/40
(w2,t1,r2,d2)	17,471	8.4	49	0/9	80	23/63	60	7/28
(w2,t1,r2,d3)	9,025	31.5	59	26/36	85	50/74	68	31/43
(w2,t2,r1,d1)	126,784	2.03	61	4/28	87	42/83	68	8/44
(w2,t2,r1,d2)	17,057	10.9	52	3/18	83	38/73	53	4/26
(w2,t2,r1,d3)	9,829	38.7	52	17/24	79	41/62	52	18/26
(w2,t2,r2,d1)	145,917	1.64	56	4/15	86	44/81	62	9/30
(w2,t2,r2,d2)	13,994	17.6	50	3/13	81	35/71	48	3/18
(w2,t2,r2,d3)	7,686	55.5	59	26/35	88	59/78	62	28/38
average			51.6	8/19	82.8	40/72	55.6	12/29

When looking at the instances of the particular pairs (n, m), we observe for the problems with $n = 30$ and $m = 20$, the API neighborhood and also the 3-API neighborhood become superior to the

SHIFT neighborhood for short runs. This tendency is stronger for the problems with w1. We note that this also corresponds to the observation in [2], where the API neighborhood became superior for problems of the type (w1,t1,r1,d1) with $n > m \geq 20$. One can conjecture that such a trend becomes even stronger for larger problems not considered in this study (see also [2]). For the long runs with S2, the SHIFT neighborhood is still superior to the API neighborhood for almost all problem types with $n = 30$ and $m = 20$. This observation is particularly obvious for the problems with w2. An exception are problems of the types (w1,t2,r1,d3) and (w2,t2,r1,d3), where both the API and the 3-API neighborhoods are clearly superior to the SHIFT neighborhood. For stopping criterion S3, sometimes the SHIFT and in other cases the API neighborhood works better. On the other side, for problems with $n = 30$ and $m = 15$ and short runs with S1, the API neighborhood is only superior for some problem types with w1 and t2.

When using stopping criterion S3, the largest number of performed iterations can be observed for algorithms with the SHIFT neighborhood and a randomly generated initial solution when $n = 30$ (the largest number of iterations have been executed for problems with w2 and t1). In this case, up to more than 90 % of the maximal number of generations have been generated. On the other side, in the case of a good initial solution the percentage of performed iterations is mostly less than 30 %, and for the API-based neighborhoods both with a weak and a good initial solution, these percentages are always less than 30 %, often even substantially less. Nevertheless, on average, only for these problems with $n > m$, the performance indices of the recommended algorithms with S3 are better than those of the recommended algorithms with S1.

From an overall point of view it turned out that problems with $n > m$ are the hardest ones, in particular those with a large ratio n/m.

4.5 Comparison with a Genetic Algorithm

Genetic algorithms belong to the class of artificial intelligence techniques and they are based on Darwin's theory about 'survival of the fittest and natural selection'. This type of algorithms has been developed by Holland [12], and one of the first genetic algorithms for scheduling problems has been given by Werner [26]. 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.

In our tests, we use the genetic algorithm tested in [2] on the mean flow time open shop scheduling problem. For a more detailed description of this algorithm, the reader is referred to [2]. Here, we use the recommended parameters, in particular we use a mutation probability of 0.8 and a crossover probability of 0.2. The initial population includes the best constructive solution of the algorithms described in Section 4.2 as one solution. We consider three variants of this genetic algorithm, denoted by $GA(popsize)$, differing only in the population size $popsize$. In particular, we apply the variants $GA(10), GA(50)$ and $GA(100)$.

We mainly compare the genetic algorithm with the short runs of simulated annealing (stopping criterion S1). In [2], both the simulated annealing and the genetic algorithms generated 30,000 solutions. However, the genetic algorithms needed substantially larger computational times. In the following, for the genetic algorithms we allow a time limit of two times the required average running times for the simulated annealing algorithms with 30,000 generated solutions (estimated in advance).

For evaluating the genetic algorithms, we also use the performance index PI introduced in Section 4.4. However, since we refer to the best value obtained by some simulated annealing variant, the

performance index can be greater than 100 for a particular instance, if a genetic algorithm generates a better solution than the best one obtained among all simulated annealing variants.

In Table 4, we present the average performance indices of the three genetic algorithms for the 24 problem types, where again all pairs (n, m) of the corresponding relation between n and m are considered. For $n < m$, it can be seen that in most cases a large population size of 100 is superior. Algorithm $GA(10)$ is better than the recommended variant Alg 1 (but we remind that the time limit for the genetic algorithms is roughly twice the time limit for Alg 1). However, on average, the performance of the long simulated annealing algorithms is not reached. Moreover, the performance indices of the genetic algorithms depend on the problem size. Sometimes the genetic algorithm reaches clearly a better performance (even than the large runs of simulated annealing with stopping criterion S2). The largest performance indices have been obtained as 143 for problem type (w2,t2,r2,d1) and as 137 for problem type (w2,t2,r1,d2) for the problems with $n = 10$ and $m = 15$ both with algorithm GA(100). On the other side, the performance index of algorithm $GA(100)$ for the problems with $n = 20$ and $m = 30$ and type (w1,t1,r1,d1) is only 14.

Table 4: Results of the genetic algorithms

(w,t,r,d)	$n < m$			$n = m$			$n > m$		
	10	50	100	10	50	100	10	50	100
(w1,t1,r1,d1)	67	65	63	83	104	111	37	25	12
(w1,t1,r1,d2)	67	67	64	64	80	89	34	24	13
(w1,t1,r1,d3)	100	100	100	60	66	62	42	34	24
(w1,t1,r2,d1)	68	73	76	77	103	100	38	27	16
(w1,t1,r2,d2)	67	74	76	82	103	105	38	27	17
(w1,t1,r2,d3)	100	100	100	74	78	88	54	43	36
(w1,t2,r1,d1)	67	77	84	94	145	149	43	31	19
(w1,t2,r1,d2)	60	75	74	66	111	122	42	33	21
(w1,t2,r1,d3)	100	100	100	74	93	101	49	47	38
(w1,t2,r2,d1)	76	98	99	107	150	166	47	39	21
(w1,t2,r2,d2)	73	98	102	96	146	166	43	38	26
(w1,t2,r2,d3)	100	100	100	78	92	101	62	59	55
(w2,t1,r1,d1)	67	67	65	60	72	76	46	25	12
(w2,t1,r1,d2)	62	66	63	45	52	60	49	30	14
(w2,t1,r1,d3)	100	100	100	55	57	60	49	35	22
(w2,t1,r2,d1)	70	74	75	77	83	86	42	25	13
(w2,t1,r2,d2)	68	81	78	71	79	88	44	28	14
(w2,t1,r2,d3)	100	100	100	72	79	81	55	43	34
(w2,t2,r1,d1)	61	75	70	81	92	90	55	35	18
(w2,t2,r1,d2)	70	76	83	76	87	99	55	39	20
(w2,t2,r1,d3)	100	100	100	84	104	97	57	46	41
(w2,t2,r2,d1)	78	97	100	108	139	227	59	37	20
(w2,t2,r2,d2)	80	94	102	93	577	363	56	43	25
(w2,t2,r2,d3)	100	100	100	86	96	97	66	62	57
average	79.2	85.6	86.5	77.5	116.1	115.9	48.3	36.5	24.5

For $n = m$, it can be observed that an average performance index of more than 100 has been obtained for 10 problem types both by algorithms $GA(50)$ and $GA(100)$. However, a large range of the performance indices can be observed. The smallest index of algorithm $GA(100)$, namely 33, has been obtained for problem type (w1,t1,r1,d3) for the instances with $n = m = 30$. Concerning

the large performance indices for problem type (w2,t2,r2,d2), we note that these two values for the algorithms $GA(50)$ and $GA(100)$ are strongly influenced by one outlier instance, where simulated annealing works bad and the function value of the initial solution is only improved by two units with the best simulated annealing algorithm while the genetic algorithm with a large population size can improve the function value by some hundreds of units. (On the other side, there are also instances for this type, where simulated annealing is better then the best genetic algorithm by several hundreds of units.) In a weaker form, this also holds for problem type (w2,t2,r2,d1). Excluding these outlier instances, the results of the genetic algorithms increase with the population size and particularly algorithm $GA(100)$ can be recommended for problems with $n = m$. However, from an overall point of view, all three genetic algorithms are superior to fast simulated annealing runs (see also [2] for mean flow time minimization).

A different behavior can be obtained for the problems with $n > m$. For these problems, the quality of the solutions of the genetic algorithm decreases with increasing population size in terms of the performance index. Moreover, the performance indices of the genetic algorithms are smaller than those obtained for fast simulated annealing algorithms (and they are substantially smaller than those for the best simulated annealing algorithms). For the best genetic algorithm $GA(10)$, the largest performance index for the problems with d1 and d2 is 84 for problem type (w2,t2,r1,d2) for the instances with $n = 15$ and $m = 10$ (note that some of the problems with d3 are easy so that larger indices have been obtained) while small performance indices have been obtained in particular for the problems with $n = 30$. More precisely, even for the best genetic algorithm $GA(10)$, for the problems with $n = 30$ and $m = 20$ a smallest performance index of 15 is obtained for problem type (w1,t2,r2,d2), for the problems with $n = 30$ and $m = 15$ the smallest index is 21 for problem type (w1,t1,r1,d3) and for the problems with $n = 30$ and $m = 10$ the smallest index is 29 for problem type (w1,t2,r1,d3). In general, among all 72 combinations of a problem type and one of the pairs (n, m) with $n = 30$, the indices of algorithm $GA(10)$ are smaller than 40 for 48 of the 72 cases, among them 32 cases with w1. Moreover, the smallest performance index of algorithm $GA(100)$ is even only 3 for the problems with $n = 30$ and $m = 15$ and type (w1,t1,r1,d2). The superiority of good simulated annealing variants becomes stronger for problems with an increasing number of jobs.

The results of the comparison of simulated annealing and genetic algorithms correspond to those obtained in [2] for problem type (w1,t1,r1,d1), where genetic algorithms are competitive for problems with $n \leq m$ while simulated annealing was clearly better for instances with $n > m$ and a large ratio of n/m.

5 Concluding Remarks

Often in the literature, a particular type of a problem is considered (e.g. processing times are uniformly distributed in the interval $[1, 100]$) and then the parameters of a simulated annealing algorithm are tuned for this concrete situation. The use of such an algorithm is then recommended for arbitrary instances of the problem under consideration. However, in general it is not a priori clear that this particular tuning is also recommendable for other types of instances of the problem when, for instance, processing times have a substantially different range, or due dates are set in another way, job weights are very different, etc. One of the major goals of this study was to find out which parameters of open shop problems with the minimization of total weighted tardiness have a strong influence and which have a smaller influence on the selection of an appropriate simulated annealing algorithm.

From our computational study for problems with up to 30 jobs and 30 machines, we can give the following conclusions and recommendations:

- The concrete data of the problems have a substantial influence on the design of an appropriate

simulated annealing algorithm. While for makespan minimization in an open shop only square problems with $n = m$ have been considered in the literature (because they are the hardest problems), the ratio of n and m has an influence on the performance of particular simulated annealing and genetic algorithms for problems with sum criteria. As in [2], we have evaluated the results separately for the cases $n < m$, $n = m$ and $n > m$.

- For problems with $n \leq m$ including positive due dates, only instances with a tightness factor up to approximately 1.1 are of interest. Even for the problems with a tightness factor between 1.0 and 1.1, the final function values are rather small and therefore, the corresponding solutions are close to the optimal ones. For larger tightness factors, problems are very easy in the sense that already simple dispatching rules construct solutions with a function value equal or close to zero. For problems with $n > m$, instances with larger tightness factors are of interest. If $n = 30$ and $m = 10$ and due dates according to d3 are considered, the objective function values of the best solution are still around 10,000 for the problems with w1 and around 30,000 for the problems with w2.

- In terms of the objective function value, the absolute improvements of the average function values of the final solution over the average values of the initial solutions are usually larger for the problems with $n > m$ (where these improvements are even up to 2,800 units) while for problems with $n < m$, these improvements are smaller (always less than 130 units for all problem types and pairs (n, m)). It appears that problems with $n < m$ are easier to solve while problems with $n > m$ are the hardest ones. This coincides with the observations made in [2], where it has been found for the problems of type (w1,t1,r1,d1) that the objective function values of the heuristic solutions are close to a lower bound for problems with $n < m$. If we consider percentage improvements of the objective function values, they are higher for the problems with d2 and d3 (where the function values of the initial solutions are considerably smaller).

- In general, the choice of a good initial solution strongly influences the quality of the iterative solution finally obtained. One possibility is to generate nondelay schedules by priority dispatching rules. Among the six rules used in our study, the LPT rule can be recommended for problems with $n < m$, the WSPT, EDD and FCFS rules are good for particular types of problems with $n > m$, and for problems with $n = m$, all the six rules considered in our study contribute good initial solutions. This confirms and generalizes the results from [5]. Since these algorithms are very fast, the application of several rules and the selection of the best solution can be recommended to generate appropriate initial solutions.

- The choice of an appropriate neighborhood has probably the largest influence on the quality of a simulated annealing algorithm. For most problem types considered in this study, the use of the SHIFT neighborhood is strongly recommended and superior to API-based neighborhoods. An exception are the following situations when using short runs with stopping criterion S1 or S3. For problems with $n < m$ and t2, the results both with the API- and the 3-API neighborhoods are better than with the SHIFT neighborhood. In addition, for large problems with $n \geq m \geq 20$, the API neighborhood and also the 3-API neighborhood become superior for more and more problem types. On the other side, for large runs with stopping criterion S2, the API neighborhood becomes superior to the SHIFT neighborhood only for the large square problems with $n = m = 30$, in particular for the problems with w1 and also for most problems with w2 and d3. Moreover, the algorithms using the crit-SHIFT neighborhood are not competitive for almost all problems.

- The selection of an appropriate simulated annealing algorithm does not essentially depend on

the concrete pair (n, m) within each of the three groups $n < m$, $n = m$ and $n > m$ with the exceptions discussed in the previous item. However, the observed trends for the problems with $n < m$ are more stronger if n/m is small, and the trends for the problems with $n > m$ are stronger if n/m is large. On the other side, if n/m is close to one, the observations are more similar to the case $n = m$. This corresponds to the results in [2] for problems with minimizing mean flow time.

- For some problems it is essential to start with an extremely small temperature. This is particularly true for problems with w1, especially for short runs. On the other side, the choice of an appropriate initial temperature is not so important for the problems with w2. In particular, for the long runs with stopping criterion S2, the use of a small initial temperature is advantageous for problems with w1 while for problems with w2, variants with a larger initial temperature become more competitive. Moreover, the use of a low initial temperature is superior for most problems with $n > m$ as well as for the problems with w1, t1, arbitrary values of n and m and arbitrary stopping criterion.

- The number of cooling cycles does not have a substantial influence on the quality of the simulated annealing algorithms. Among the recommended algorithms, there are variants with one and five cooling cycles. From an overall point of view, the use of five cooling cycles leads to slightly better results, in particular for the problems with $n \leq m$.

- As one can expect, the long runs with stopping criterion S2 obtain the best results. However, when using long runs with the API and 3-API neighborhoods, for most problem types the results are nevertheless worse than in the case of short runs with the SHIFT neighborhood. This is partially opposite for problems with $n < m$ and t2. Variants with the flexible stopping criterion S3 are not superior to short runs with stopping criterion S1 for the majority of problem types. An exception are most types of the hard problems with $n > m$, in particular problems with w2.

- From an overall point of view, a variant using a good initial solution and the SHIFT neighborhood with a small initial temperature of two and one or five cooling cycles can be recommended among the simulated annealing algorithms for problems with up to 30 jobs and 30 machines. However, as mentioned above, for the problems with $n \geq m \geq 20$, the API neighborhood becomes better. It can be conjectured that this trend becomes even stronger for problems with $n \geq m$ as the number of machines increases further.

- When comparing fast simulated annealing and the genetic algorithms used in our study, we have to distinguish the cases $n \leq m$ and $n > m$. While for problems with $n \leq m$ the genetic algorithm with a large population size often gets a better solution than short and sometimes even the best simulated annealing algorithm, this is not true for the problems with $n > m$. Here a good fast simulated annealing algorithm is usually superior to the best genetic algorithm (and the genetic algorithms perform extremely poor in comparison to the long simulated annealing algorithms).

The algorithms presented in this paper are included into the program package LiSA - A Library of Scheduling Algorithms, version 3.0 (see http://lisa.math.uni-magdeburg.de). For a free use of the algorithms discussed in this paper and the whole package, the interested reader can contact the LiSA team under the above website. A table with the seeds for generating the open shop instances used in this paper can also be obtained.

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] Andresen, M.; Bräsel, H.; Mörig, M.; Tusch, J.; Werner, F.; Willenius, P.: Simulated Annealing and Genetic Algorithms for Minimizing Mean Flow Time in an Open Shop, Math. Comp. Modelling (to appear, doi 10.1016/j.mcm.2008.01.002).

[3] Blazewicz, J.; Pesch, E.; Sterna, M.; Werner, F.: Open Shop Scheduling with Late Work Criteria, Discrete Appl. Math., Vol. 134, 2004, 1 - 24.

[4] Bräsel, H.: Matrices in Shop Scheduling Problems, in: Perspectives on Operations Research - Essays in Honor of Klaus Neumann (ed. by M. Morlock, C. Schwindt, N. Trautmann and J. Zimmermann), Deutscher Universitäts-Verlag, Wiesbaden, 2006, 17 - 43.

[5] Bräsel, H.; Herms, A.; Mörig, M.; Tautenhahn, T.; Tusch, T.; Werner, F.: Heuristic Constructive Algorithms for Open Shop Scheduling to Minimize Mean Flow Time, European J. Oper. Res., Vol. 189, 2008, 856 - 870.

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

[7] Brightwell, G.; Winkler, P.: Counting Linear Extensions, Order, Vol. 8, 1991, 225 - 242.

[8] 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.

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

[10] 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.

[11] Gueret, C.; Prins, C.: A New Lower Bound for the Open-Shop Problem, Annals Oper. Res., Vol. 92, 1999, 165 - 183.

[12] Holland, J.A.: Adaptation in Natural and Artificial Systems, Ann Arbor, University of Michigan, 1975.

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

[14] Laborie, P.: Complete MCS-Based Search, Application to Resource Constrained Project Scheduling, Proceedings of International Joint Conference on Artificial Intelligence, Vol. 19, 2005, 181 - 186.

[15] Liaw, C.-F.: Applying Simulated Annealing to the Open Shop Scheduling Problem, IEE Transactions, Vol. 31, 1999, 457 - 465.

[16] Liaw, C.-F.: Scheduling Two-Machine Preemptive Open Shop Shops to Minimize Total Completion Time, Comput. Oper. Res., Vol. 31, 2004, 1349 - 1363.

[17] Liaw, C.-F.: Scheduling Preemptive Open Shops to Minimize Total Tardiness, European J. Oper. Res., Vol. 162, 2005, 175 - 183.

[18] 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.

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

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

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

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

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

[24] Taillard, E.: Benchmarks for Basic Scheduling Problems, European J. Oper. Res., Vol. 64, 1993, 278 - 285.

[25] Tamura, N.; Taga, A.; Kitagawa, S.; Banbara, M.: Compiling Finite Linear CSP into SAT, Proceeding of the 12th International Conference on Principles and Practice of Constraint Programming (CP'06), Lecture Notes in Computer Science, Vol. 4204, Springer, 2006, 590 - 603.

[26] Werner, F.: On the Solution of Special Sequencing Problems, Ph.D. Thesis, TU Magdeburg, 1984 (in German).

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

[28] Yang, Q; Sun, J.; Zhang, J.; Wang C. : A Hybrid Discrete Particle Swarm Algorithm for Open-Shop Problems, Lecture Notes in Computer Science, Vol. 4247, 2006, 158 - 165.