Sequencing algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria

Jitti Jungwattanakit[a], Manop Reodecha[a], Paveena Chaovalitwongse[a], Frank Werner[b,*]

[a]*Department of Industrial Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand*
[b]*Faculty of Mathematics, Otto-von-Guericke-University, P.O. Box 4120, D-39016 Magdeburg, Germany*

Abstract

This paper considers a flexible flow shop scheduling problem, where at least one production stage is made up of unrelated parallel machines. Moreover, sequence- and machine- dependent setup times are given. The objective is to find a schedule that minimizes a positively weighted convex sum of makespan and the number of tardy jobs in a static flexible flow shop environment. For this problem, a 0-1 mixed integer program is formulated. The problem is however a combinatorial optimization problem which is too difficult to be solved optimally for large problem sizes, and hence heuristics are used to obtain good solutions in a reasonable time. The proposed constructive heuristics for sequencing the jobs start with the generation of the representatives of the operation time for each operation. Then some dispatching rules and flow shop makespan heuristics are developed. To improve the solutions obtained by the constructive algorithms, polynomial heuristic improvement algorithms based on shift moves and pairwise interchanges of jobs are applied. In addition, metaheuristics are suggested, namely simulated annealing, tabu search and genetic algorithms. The basic parameters of each metaheuristic are briefly discussed in this paper. The performance of the heuristics is compared relative to each other on a set of test problems with up to 50 jobs and 20 stages and with an optimal solution for small-size problems. We have found that among the constructive algorithms the insertion based approach is superior to the others, whereas the proposed simulated algorithms are better than tabu search and genetic algorithms among the iterative metaheuristic algorithms.

Keywords: Flexible flow shop; Constructive algorithms; Simulated Annealing; Tabu Search; Genetic algorithms

* Corresponding author. Tel.: +49-391-6712025; /fax.: +49-391-6711171.
E-mail address: frank.werner@mathematik.uni-magdeburg.de

1. Introduction

Production scheduling is a decision-making process in the operation level. It can be defined as the allocation of available production resources to carry out certain tasks in an efficient manner. A frequently occurring scheduling problem is difficult to solve due its complex nature.

This paper is primarily concerned with industrial scheduling problems, where one first has to assign jobs to limited resources and then to sequence the assigned jobs on each resource over time. It is mainly concerned with processing industries that are established as multi-stage production facilities with multiple production units per stage (i.e., parallel machines), e.g. a textile industry (Karacapilidis and Pappis [1]), an automobile assembly plant (Agnetis *et al.* [2]), a printed circuit board manufacture (Alisantoso, Khoo and Jiang [3], and Hsieh, Chang and Hsu [4]), and so on. In such industries, at some stages the facilities are duplicated in parallel to increase the overall capacities or to balance the capacities of the stages, or either to eliminate or to reduce the impact of bottleneck stages on the shop floor capacities. Due to the mixed character of such a production system, which lies between flow shop and parallel machines, it is known as a flexible or hybrid flow shop environment.

An ordinary flow shop model is a multi-stage production process, where the jobs have to visit all stages in the same order string, whereas a flexible flow shop model, a generalization of the classical flow shop model, is more realistic, and it assumes that at least one stage must have multiple machines. Moreover, a machine can process at most one job at a time and a job can be processed by at most one machine at a time. Preemption of processing is not allowed. The problem consists of assigning the jobs to machines at each stage and sequencing the jobs assigned to the same machine so that some optimality criteria are minimized.

Although the flexible flow shop problem has been widely studied in the literature, most of the studies related to flexible flow shop problems are concentrated on problems with identical processors, see for instance, Gupta, Krüger, Lauff, Werner and Sotskov [5], Alisantoso, Khoo and Jiang [3], Lin and Liao [6], and Wang and Hunsucker [7]. In a real world situation, it is common to find newer or more modern machines running side by side with older and less efficient machines. Even though the older machines are less efficient, they may be kept in the production lines because of their high replacement costs. The older machines may perform the same operations as the newer ones, but would generally require a longer operating time for the same operation. In this paper, the flexible flow shop problem with unrelated parallel machines is considered, i.e., there are different parallel machines at every stage and speeds of the machines are dependent on the jobs. Moreover, several industries encounter setup times which result in even more difficult scheduling problems. In this paper, both sequence- and machine-dependent setup time restrictions are taken into account as well.

A detailed survey for the flexible flow shop problem has been given by Linn and Zhang [8] and Wang [9]. Most of the earlier literature has considered the simple case of only two stages. Arthanari and Ramamurthy [10] and Salvador [11] are among the first who define the flexible flow shop problem. They propose a branch and bound method to tackle the problem. Such a method is an exact solution technique which guarantees optimal solutions. However, the exact algorithm presented can only be applied to very small instances. Other exact approaches for the multi-stage flexible flow shop problem are proposed by many authors, e.g. branch and bound algorithms are given in Brah and Hunsucker [12] and Moursli and Pochet [13].

When an exact algorithm is applied to large flexible flow shop problems, such an approach can take hours or days to derive a solution. On the other hand, a heuristic approach is much faster but does not guarantee an optimum solution. Gupta [14] proposes heuristic techniques for a simplified flexible flow shop makespan problem with two stages and only one machine at stage two. Sriskandarajah and Sethi [15] develop simple heuristic algorithms for the two-stage flexible flow shop problem. They discuss the worst and average case performance of algorithms for finding minimum makespan schedules. Their solutions are based on Johnson's rule. Guinet, Solomon, Kedia and Dussauchoy [16] also propose a heuristic for the makespan problem in a two-stage flexible flow shop based on Johnson's rule. They compare this heuristic with the

Shortest Processing Time (SPT) and the Longest Processing Time (LPT) dispatching rules. They conclude that the LPT rule gives good results for the two-stage makespan problem. Gupta and Tunc [17] consider the two-stage flow shop scheduling problem where there is one machine at stage one and the number of identical machines in parallel at stage two is less than the total number of jobs. The setup and removal times of each job at each stage are separated from the processing times. They propose heuristic algorithms that are empirically tested to determine the effectiveness in finding an optimal solution. Santos, Hunsucker and Deal [18] investigate scheduling procedures which seek to minimize the makespan in a static flow shop with multiple processors. Their method is to generate an initial permutation schedule based on the Palmer, CDS, Gupta and Dannenbring flow shop heuristics, and then it is followed by the application of the First in First out (FIFO) rule.

To obtain a near-optimal solution, metaheuristic algorithms have also been proposed. Gourgand, Grangeon and Norre [19] present several simulated annealing (SA)-based algorithms for the flexible flow shop problem. A specific neighborhood is used and the authors apply the methods to a realistic industrial problem. Jin, Yang and Ito [20] consider the flexible flow shop with identical parallel machines. They propose two approaches to generate the initial job sequence and use a simulated annealing algorithm to improve it. It can be seen that a simulated annealing algorithm has been successfully applied to various combinatorial optimization problems. For an extensive survey of the theory and applications of the SA algorithm, see Koulamas, Antony and Jaen [21]. Furthermore, Nowicki and Smutnicki [22] propose a tabu search (TS) algorithm for the flexible flow shop makespan problem. A genetic algorithm has been widely used in many previous works for the flow shop makespan problem, see e.g. Werner [23], Reeves [24]. Cheng, Gen and Tozawa [25] address the earliness/tardiness scheduling problem with identical parallel machines, and they apply a genetic algorithm to solve this problem. Ruiz, Maroto and Alcaraz [26] use a genetic algorithm to deal with the permutation flow shop scheduling problem with sequence-dependent setup times. However, little research has been done for flexible flow shop scheduling problems, especially for the general case with unrelated parallel machines and setup times (see for instance the recent review on scheduling with setup times by Allahverdi, Ng, Cheng and Kovalyov [27]).

In this paper, a flexible flow shop problem with unrelated parallel machines and setup times is studied. The goal is to seek a schedule which minimizes a positively weighted convex sum of makespan and the number of tardy jobs. We will investigate how to apply constructive and improvement algorithms as well as metaheuristics to solve the flexible flow shop problem with unrelated parallel machines approximately. In particular, constructive heuristics based on dispatching rules and flow shop makespan heuristics are adapted and fast polynomial heuristic improvement algorithms are used to improve the quality of the solution. Metaheuristics, namely simulated annealing, tabu search and genetic algorithms are proposed.

The rest of this paper is organized as follows: The problem considered is described in Section 2. The mathematical model for this problem is introduced in Section 3. The heuristic construction of a schedule and iterative algorithms are sketched in Section 4 and Section 5, respectively. Computational results with the heuristics are briefly discussed in Section 6 and conclusions are given in Section 7.

2. Problem description

The flexible flow shop system is defined by a set $O = \{1,\ldots, t,\ldots, k\}$ of k processing stages. At each stage t, $t \in O$, there is a set $M^t = \{1,\ldots, i,\ldots, m^t\}$ of m^t unrelated machines. The set $J = \{1,\ldots, j,\ldots, n\}$ of n independent jobs has to be processed on machines of the sets $M^1,\ldots, M^k$. Each job j, $j \in J$, has its release date $r_j \geq 0$ and a due date $d_j \geq 0$. It has its fixed standard processing time for every stage t, $t \in O$. Owing to the unrelated machines, the processing time p_{ij}^t of job j on machine i at stage t is equal to ps_j^t / v_{ij}^t, where ps_j^t is the standard processing time of job j at stage t, and v_{ij}^t is the relative speed of job j which is processed by the machine i at stage t.

There are processing restrictions of the jobs as follows: (1) jobs are processed without preemptions on any machine; (2) every machine can process only one operation at a time; (3) operations of a job have to be realized sequentially, without overlapping between the stages; (4) job splitting is not permitted.

Setup times considered in this problem are classified into two types, namely a machine-dependent setup time and a sequence-dependent setup time. A setup time of a job is machine-dependent if it depends on the machine to which the job is assigned. It is assumed to occur only when the job is the first job assigned to the machine. ch_{ij}^{t} denotes the machine-dependent setup time (or changeover time) of job j if job j is the first job assigned to machine i at stage t. A sequence-dependent setup time is considered between successive jobs. A setup time of a job on a machine is sequence-dependent if it depends on the job just completed on that machine. s_{lj}^{t} denotes the time needed to changeover from job l to job j at stage t, where job l is processed directly before job j on the same machine. All setup times are known and constant. Moreover, there is given a non-negative machine availability time for any machine of a particular stage.

The scheduling problem under consideration has dual objectives, namely minimizing the makespan and the number of tardy jobs. Therefore, the objective function to be minimized is

$$\lambda C_{max} + (1 - \lambda) \eta_T \tag{1}$$

where C_{max} is the makespan, which is equivalent to the completion time of the last job to leave the system, η_T is the total number of tardy jobs in the schedule, and λ is the weight (or relative importance) given to C_{max} and η_T ($0 \le \lambda \le 1$).

3. Mathematical model

In this section, we provide a 0-1 mixed integer linear programming formulation for the problem under consideration.

3.1. Notations

t	stage index, $t = 1, 2, 3, \ldots, k$
m^t	number of parallel machines at stage t
i	machine index, $i = 1, 2, 3, \ldots, m^t$
j, l	job index, $j, l = 1, 2, 3, \ldots, n$
r_j	release date of job j
d_j	due date of job j
s_{jl}^{t}	setup time between job j and job l at stage t
ch_{ij}^{t}	setup time of job j if job j is assigned to machine i at the first position at stage t
ps_{j}^{t}	standard processing time of job j at stage t
v_{ij}^{t}	relative speed of machine i at stage t for job j
a_{i}^{t}	time when machine i at stage t becomes available
X_{ijl}^{t}	1 if job j is scheduled immediately before job l on machine i at stage t, and 0 otherwise
O_{j}^{t}	operating time of job j at stage t
C_{j}^{t}	completion time of job j at stage t
C_{max}	makespan
U_j	a Boolean variable: 1 if job j is tardy, and 0 otherwise
T_j	tardiness of job j
η_T	total number of tardy jobs in the schedule

3.2. Mathematical formulation

The problem can be formulated as follows.

$$\textit{minimize} \quad \lambda C_{max} + (1-\lambda)\eta_T \tag{2}$$

Subject to:

$$\sum_{i=1}^{m^t}\sum_{j=0}^{n} X_{ijl}^t = 1 \qquad ,\forall t, l \tag{3}$$

$$\sum_{i=1}^{m^t}\sum_{l=1}^{n+1} X_{ijl}^t = 1 \qquad ,\forall t, j \tag{4}$$

$$\sum_{l=1}^{n+1} X_{i0l}^t = 1 \qquad ,\forall t, i \tag{5}$$

$$\sum_{j=0}^{n} X_{ij(n+1)}^t = 1 \qquad ,\forall t, i \tag{6}$$

$$X_{ijj}^t = 0 \qquad ,\forall t, i, j \tag{7}$$

$$\sum_{j=0}^{n} X_{ijl}^t = \sum_{j=1}^{n+1} X_{ilj}^t \qquad ,\forall t, i, l \tag{8}$$

$$X_{ijl}^t \in \{0,1\} \qquad ,\forall t, i, j, l;\ j=0;\ l=n+1 \tag{9}$$

$$O_j^t = \sum_{i=1}^{m^t}\sum_{l=1}^{n+1} \frac{ps_j^t}{v_{ij}^t} X_{ijl}^t \qquad ,\forall t, j \tag{10}$$

$$C_l^t - C_j^t \geq s_{jl}^t + O_l^t + [(\sum_{i=1}^{m^t} X_{ijl}^t) - 1]B \qquad ,\forall t, j, l; j \neq l \tag{11}$$

$$C_j^t \geq 0 \qquad ,\forall t, j \tag{12}$$

$$C_l^t - C_l^{t-1} \geq \sum_{i=1}^{m^t}\sum_{j=1}^{n} X_{ijl}^t s_{jl}^t + \sum_{i=1}^{m^t} ch_{il}^t X_{i0l}^t + O_l^t \qquad ,\forall t, l \tag{13}$$

$$C_j^0 = r_j \qquad ,\forall j \tag{14}$$

$$C_j^t \geq \sum_{i=1}^{m^t} a_i^t X_{i0j}^t + \sum_{i=1}^{m^t} ch_{il}^t X_{i0l}^t + O_j^t \qquad ,\forall t, j \tag{15}$$

$$C_{\max} \geq C_j^k \qquad ,\forall j \tag{16}$$

$$T_j \geq C_j^k - d_j \qquad ,\forall j \tag{17}$$

$$T_j \geq 0 \qquad ,\forall j \tag{18}$$

$$U_j \leq BT_j \qquad ,\forall j \tag{19}$$

$$BU_j \geq T_j \qquad ,\forall j \tag{20}$$

$$\eta_T = \sum_{j=1}^{n} U_j \qquad ,\forall j \tag{21}$$

$$U_j \in \{0,1\} \qquad ,\forall j \tag{22}$$

In the above formulation, the objective is to determine a schedule that minimizes the makespan and the number of tardy jobs, see Equation (2). We have $X_{i0l}^t = 1$ if job l is sequenced as the first job on machine i at stage t, and $X_{ij(n+1)}^t = 1$ if job j is sequenced as the last job on machine i at stage t. Constraints (3)–(9) ensure that the partial schedule on each machine at each stage is feasible. Constraint sets (3) and (4) ensure that only one job is assigned to each sequence position at each stage. Constraint sets (5) and (6) ensure that

only one job will be assigned to the first and last positions, respectively, on each machine at each stage. Constraint (7) assures that after the job has been finished at any stage, it cannot be reprocessed at the same stage. Constraint (8) forces to construct a consistent sequence at every stage. Constraint (9) specifies the decision variables X_{ijl}^{t} as binary variables. Constraint (10) determines the operating time of every job which is dependent on the machine. Constraints (11)–(15) find the completion time of every job. Constraint (11) is a set of disjunctive constraints. It states that, if jobs *j* and *l* are scheduled on the same machine at a particular stage with job *j* scheduled before job *l*, then job *j* must complete the processing before job *l* can begin. This constraint set forces job *l* to follow job *j* by at least the processing time of job *l* plus the setup time from *j* to *l* if job *l* is immediately scheduled after job *j*. The value of *B* is set to a very big constant, i.e., it is greater than the sum of all job processing times and setup times. Constraint (12) ensures that the completion time of every job at each stage is a non-negative value. Constraint (13) specifies the conjunctive precedence constraints for the jobs, which says that a job cannot start its processing at stage $t + 1$ before it finishes at stage *t*. Constraint (14) applies only to stage one, saying that a job cannot start its processing at stage one before its release date. Constraint (15) applies only to jobs that are assigned to the first sequence on each machine, that is, the job cannot start its processing before machine availability. Constraint set (16) links the makespan decision variable. Constraint sets (17) and (18) determine the correct value of the tardiness (T_j). Constraint set (17) determines the correct value of the lateness (L_j) and (18) specifies only the positive lateness as the tardiness ($T_j = \max\ \{0, C_j^k - d\}$). Constraint sets (19)–(22) link the decision variable of the number of tardy jobs, that is, if tardiness is greater than zero, the job is tardy; otherwise this job is not tardy.

It is noted that an optimal solution can be obtained by running a commercial mathematical programming software, CPLEX 8.0.0 and AMPL, with an Intel Pentium 4 2.00 GHz CPU. We have found that the mathematical model can be used for solving problems with up to seven jobs and four stages in acceptable time.

4. Constructive algorithms

Since the flexible flow shop scheduling problem is NP-hard, algorithms for finding an optimal solution in polynomial time are unlikely to exist. Thus, heuristic methods are studied to find approximate solutions. Most researchers develop existing heuristics for the classical flexible flow shop problem with identical machines by using a particular sequencing rule for the first stage. They follow the same scheme, see Santos, Hunsucker and Deal [18]. The algorithm is as follows:

Algorithm 1.
Step 1: Sequence the jobs by using a particular sequence rule (the first-stage sequence).
Step 2: Assign the jobs to the machines at every stage using the job sequence from either the First-In-First-Out (FIFO) rule or the Permutation rule.
Step 3: Return the best solution.

Firstly, a job sequence is determined according to a particular sequencing rule, and we will briefly discuss the modifications for the problem under consideration in the next section. Secondly, jobs are assigned as soon as possible to the machines at every stage using the job sequence determined for the first stage. There are basically two approaches for this subproblem. The first way is that for the other stages, i.e. from stage two to stage *k*, jobs are ordered according to their completion times at the previous stage. This means that the FIFO (First in First out) rule is used to find the job sequence for the next stage by means of the job sequence of the previous stage. The second way is to sequence the jobs for the other stages by using the same job sequence as for the first stage, called the permutation rule.

Assume now that a job sequence for the first stage has already been determined. Then we have to solve the problem of scheduling *n* jobs on unrelated parallel machines with sequence- and machine-dependent

setup times using this given job sequence for the first stage. We apply a greedy algorithm which constructs a schedule for the n jobs at a particular stage provided that a certain job sequence for this stage is known (remind that the job sequence for this particular stage is derived either from the FIFO rule or from the permutation rule), where the objective is to minimize the flow time and the idle time of the machines. The idea is to balance evenly the workload in a heuristic way as much as possible.

4.1. Constructive Heuristics

In order to determine the job sequence for the first stage by some heuristics, we remind that the processing and setup times for every job are dependent on the machine and the previous job, respectively. This means that they are not fixed, until an assignment of jobs to machines for the corresponding stage has been done. Thus, for applying an algorithm for fixing the job sequence for stage one, an algorithm for finding the representatives of the machine speeds and the setup times is necessary.

The representatives of machine speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$ for stage t, t=1,…k, use the minimum, maximum and average values of the data. Thus, the representative of the operating time of job j at stage t is the sum of the processing time $ps_j^t / v_{ij}^{/t}$ plus the representative of the setup time $s_{lj}^{/t}$. Nine combinations of relative speeds and setup times will be used in our algorithms. The job sequence for the first stage is then fixed as the job sequence with the best function value obtained by all combinations of the nine different relative speeds and setup times.

For determining the job sequence for the first stage, we adapt and develop several basic dispatching rules and constructive algorithms for the flow shop makespan scheduling problem. Some of the dispatching rules are related to tardiness-based criteria, whereas others are used mainly for comparison purposes. The steps of the algorithm are as follows:

Algorithm 2.

Step 1: Select the representatives of relative speeds and setup times for every job and every stage by using the combinations of the minimum, maximum, and average data values.

Step 2: Calculate the representatives of the operating time by using the term $ps_j^t / v_{ij}^{/t} + s_{lj}^{/t}$.

Step 3: Use a particular dispatching rule or constructive algorithm to find the first-stage sequence based on the specific representatives of the operating time.

Step 4: Apply the heuristic schedule construction (see Algorithm 1).

Step 5: Return the best solution.

The Shortest Processing Time (SPT) rule is a simple dispatching rule, in which the jobs are sequenced in non-decreasing order of the processing times, whereas the Longest Processing Time (LPT) rule orders the jobs in non-increasing order of their processing times. The Earliest Release Date first (ERD) rule is equivalent to the well-known first-in-first-out (FIFO) rule. The Earliest Due Date first (EDD) rule schedules the jobs according to non-decreasing due dates of the jobs. The Minimum Slack Time first (MST) rule is a variation of the EDD rule. This rule concerns the remaining slack of each job, defined as its due date minus the processing time required to process it. The Slack time per Processing time (S/P) is similar to the MST rule, but its slack time is divided by the processing time required as well (Baker [28], and Pinedo and Chao [29]).

The hybrid SPT and EDD (HSE) rule is developed to combine both SPT and EDD rules. Firstly, consider the processing times of each job and determine the relative processing time compared to the maximum processing time required. Secondly, determine the relative due date compared to the maximum due date. Next, calculate the priority value of each job by using the weight (or relative importance) given to C_{max} and η_T for the relative processing time and relative due date.

Palmer's heuristic [30] is a makespan heuristic denoted by PAL in an effort to use Johnson's rule by proposing a *slope order index* to sequence the jobs on the machines based on the processing times. The idea is to give priority to jobs that have a tendency of progressing from short times to long times as they move through the stages. Campbell, Dudek and Smith [31] develop one of the most significant heuristic methods for the makespan problem known as CDS algorithm. Its strength lies in two properties: (1) it uses Johnson's rule in a heuristic fashion, and (2) it generally creates several schedules from which a "best" schedule can be chosen. In so doing, $k - 1$ sub-problems are created and Johnson's rule is applied to each of the sub-problems. Thus, $k - 1$ sequences are generated. Since Johnson's algorithm is a two-stage algorithm, a k-stage problem must be collapsed into a two-stage problem.

Gupta [32] provides an algorithm denoted by GUP, in a similar manner as algorithm PAL by using a different slope index and scheduling the jobs according to the slope order. Dannenbring [33] develops a method, denoted by DAN, by using Johnson's algorithm as a foundation. Furthermore, the CDS and PAL algorithms are also exhibited. Dannenbring constructs only one two-stage problem, but the processing times for the constructed jobs reflect the behavior of PAL's slope index. Its purpose is to provide good and quick solutions.

Nawaz, Enscore and Ham [34] develop a probably best constructive heuristic method for the permutation flow shop makespan problem, called the NEH algorithm. It is based on the idea that a job with a high total operating time on the machines should be placed first at an appropriate relative order in the sequence. Thus, jobs are sorted in non-increasing order of their total operating time requirements. The final sequence is built in a constructive way, adding a new job at each step and finding the best partial solution. For example, the NEH algorithm inserts a third job into the previous partial solution of two jobs which gives the best objective function value under consideration (the relative position of the two previous jobs in the sequence remains fixed). The algorithm repeats the process for the remaining jobs according to the initial ordering of the total operating time requirements.

Again, to apply these algorithms to the flexible flow shop problem with unrelated parallel machines, the total operating times for calculating the job sequence for the first stage are calculated for the nine combinations of relative speeds of machines and setup times.

4.2. Improvement Heuristics

Unlike constructive algorithms, improvement heuristics start with an already built schedule and try to improve it by some given procedures. Their use is necessary since the constructive algorithms (especially some algorithms that are adapted from pure makespan heuristics and some dispatching rules such as the SPT and LPT rules) do not consider due dates (and therefore, they do not consider the minimization of the number of tardy jobs). In this section, some fast improvement heuristics will be investigated to improve the overall function value by concerning mainly the due date criterion. The iterative algorithms described in the following and in Section 5 are based on the shift move (SM) and the pairwise interchange (PI) neighborhoods.

The SM neighborhood repositions a chosen job. This means that an arbitrary job π_r at position r is shifted to position i, while leaving all other relative job orders unchanged. If $1 \le r < i \le n$, it is called a right shift and yields $\pi' = [\pi_1,\ldots, \pi_{r-1}, \pi_{r+1}, \ldots, \pi_i, \pi_r, \pi_{i+1},\ldots, \pi_n]$. If $1 < i < r < n$, it is called a left shift and yields $\pi' = [\pi_1,\ldots, \pi_{i-1}, \pi_r, \pi_i,\ldots, \pi_{r-1}, \pi_{r+1}, \ldots, \pi_n]$. For instance, assume that randomly one solution in the current generation is selected, say [4 9 8 7 3 1 6 2 5], and then randomly a couple of job positions for performing the shift is selected, e.g. positions 2 and 7 (in this case, it is a right shift). The new solution will be [4 8 7 3 1 6 ***9*** 2 5]. However, if positions 7 and 2 are randomly selected (i.e. it is a left shift), the new solution will be [4 ***6*** 9 8 7 3 1 2 5]. In the SM neighborhood, the current solution has $(n-1)^2$ neighbors.

The PI neighborhood exchanges a pair of arbitrary jobs π_r, and π_i, where $1 \le i, r \le n$ and $i \ne r$. Such an operation swaps the jobs at positions r and i, which yields $\pi' = [\pi_1,\ldots, \pi_{r-1}, \pi_i, \pi_{r+1}, \ldots, \pi_{i-1}, \pi_r, \pi_{i+1},\ldots, \pi_n]$. For example, assume that the current solution is [4 9 8 7 3 1 6 2 5], and then randomly the couple of job

positions to be exchanged is selected, e.g. positions 1 and 3. Thus, the new solution will be [***8*** 9 ***4*** 7 3 1 6 2 5]. In the PI neighborhood, the current solution has $n\times(n\text{-}1)/2$ neighbors.

In order to find a satisfactory solution of the due date problem, we apply fast polynomial heuristics by applying either the above shift move (SM) algorithm as an improvement mechanism based on the idea that we will consider the jobs that are tardy in a left-to-right scan and move each of them left and right, or the pairwise interchange (PI) algorithm, where a tardy job is selected and swapped to different job positions left and right, and either to two randomly determined positions (denoted by the number "2") or to all n–1 possible positions (denoted by the letter "A"). The best schedule among the generated neighbors is then taken as the result. The algorithm is as follows:

Algorithm 3.
Step 1: Select the first tardy job in the job sequence.
Step 2: Interchange (i.e. apply 2-PI or A-PI) or shift (i.e. apply 2-SM or A-SM) the chosen job and evaluate the objective function values.
Step 3: Update the current job sequence.
Step 4: Go to Step1 until all tardy jobs have been considered.
Step 5: Return the best sequence solution.

Since every tardy job in the job sequence is considered at most once, the complexity of the 2-PI and 2-SM procedures is $O(n)$ and the complexity of the A-PI and A-SM procedures is $O(n^2)$.

5. Iterative algorithms

5.1. Simulated Annealing

A simulated annealing (SA) algorithm is an iterative search method, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. It was developed by Kirkpatrick, Gelatt and Vecchi [35]. An SA algorithm simulates ideas and mechanisms found in the physical annealing of a solid. The concept comes from the field of materials science in which a solid is first melted after heated in a high temperature, and then it is slowly cooled so that it reaches a thermodynamic equilibrium.

In general, an SA algorithm is a stochastic optimization method for minimizing a function f over a discrete domain $\boldsymbol{S}$. Starting from an initial solution $s\in \boldsymbol{S}$, an SA algorithm generates a new solution $s'\in \boldsymbol{S}$ in the neighborhood of the initial solution s by using a suitable operator. Concerning the neighborhood, we considered both a shift move (SM) neighborhood (i.e. a job at an arbitrary position is selected and reinserted at some other position) and a pairwise interchange (PI) neighborhood (i.e. two arbitrary jobs are selected and interchanged). This new point's objective function value $f(s')$ is then compared to the initial point's value $f(s)$ (remind that the objective function value of the full schedule generated from the job sequence for the first stage is taken). The change in the objective function value, $\delta = f(s')–f(s)$, is calculated. If the objective function value decreases ($\delta < 0$), it is automatically accepted and it becomes the point from which the search will continue. If the objective function value increases ($\delta \geq 0$), then higher values of the objective function may also be accepted with a probability, usually determined by a function, $\exp(-\delta/T)$, where $T \in \Re$ is a control parameter of the SA algorithm called the *temperature*. The role of the temperature T is significant in the operation of an SA algorithm. This temperature, which is simply a positive number, is periodically reduced every NT iterations, where NT denotes the epoch length, so that it moves gradually from a relatively high value to near zero as the algorithm progresses according to a function referred to as the *cooling schedule*. In our tests, we investigated in particular the influence of the chosen neighborhood and the cooling scheme for controlling the temperature. We used a geometric (i.e. $T_{new} = \alpha\times T_{old}$) and a Lundy-Mees

reduction [36] (i.e. $T_{new} = T_{old}/(1+\beta \times T_{old})$) scheme and tested the parameters of these schemes (initial temperatures, temperature reductions and neighborhood structures).

5.2. Tabu Search

A tabu search (TS) algorithm, initially developed by Glover [37] is an iterative improvement approach designed to avoid terminating prematurely at a local optimum for solving combinatorial optimization problems. Similar to a simulating annealing algorithm, a TS algorithm is based on the idea of exploring the solution space of a problem by moving from one region of the search space to another in order to look for a better solution. The function transforming a solution into another solution is usually called *a move*. For any solution $s \in \boldsymbol{S}$, a subset of moves applicable to it is defined. This subset of moves generates the *neighborhood* $\aleph(s)$ of s. Starting from an initial solution s, the TS algorithm iteratively moves from the current solution s to the best solution $s^* \in \aleph(s)$ even though s^* is worse than the current solution s, until some stopping criterion is satisfied.

However, to escape from a local optimum, an SA algorithm accepts an inferior solution, which may lead to better solutions later by using an acceptance probability. In contrast, a TS algorithm allows the search to move to the best solution s^* among a set of candidate moves $\aleph(s)$ as defined by the neighborhood structure, although it can move to a neighbor with a worse objective function value. Nevertheless, subsequent iterations may cause the search to move repeatedly back to the same local optimum. In order to prevent cycling back to recently visited solutions, it should be *forbidden* or declared *tabu* for a certain number of iterations. This is accomplished by keeping the attributes of the forbidden moves in a list, called the *tabu list*. The size of the tabu list, called the *tabu tenure*, must be large enough to prevent cycling, but small enough not to forbid too many moves.

Additionally, an aspiration criterion is defined to deal with the case in which a move leading to a new best solution is tabu. If a current tabu move satisfies the *aspiration criterion*, its tabu status is canceled and it becomes an allowable move. The use of the aspiration criterion allows the TS algorithm to lift the restrictions and intensify the search into a particular solution region.

5.3. Genetic Algorithm

A genetic algorithm (GA) approach is an iterative heuristic based on Darwin's evolutionary theory about "survival of the fittest and natural selection". It belongs to the evolutionary class of artificial intelligent (AI) techniques. It was invented by Holland [38] and then it has been applied to a large number of complex search problems.

The GA approach is characterized by a parallel search of the state space in contrast to a point-by-point search by conventional techniques. The parallel search is achieved by keeping a set of possible solutions, called *a population*. An individual in the population is a string of symbols. The GA starts with the initial generation of artificial individuals which are often created randomly. Each symbol is called *a gene* and each string of genes is termed as *a chromosome*. The individuals in the population are evaluated by a measure called the *fitness* to describe quantitatively how well the individual masters its task. The *initial population* is then evolved into different populations over a number of *generations* through the use of two types of *genetic operators*: (1) unary operators such as mutation and inversion which change the genetic structure of a single chromosome, and (2) a higher-order operator, referred to as crossover which consists of obtaining new individual(s) by combining the genetic material from two selected parent chromosomes. When applying crossover, two individuals (parents) are selected from the population and new solution(s), called the *offspring*, is (are) created. Mutation creates a new solution by a random change on a selected individual. The genetic operators are applied to randomly selected parents to generate new offspring. Then the new population is selected out of the individuals of the current population and the new generated chromosomes.

The application of the GA approach requires the representation of a solution, the choice of genetic operators (crossover and mutation), an evaluation function, a selection mechanism and the determination of genetic parameters (population size as well as crossover and mutation rates). For the representation, consideration of a job permutation is straightforward and widely used in many previous works on the GA approach for the flow shop problem, see e.g. Werner (1984). Thus, in our GA approach, we apply a permutation-based code (or job code) using integers as the chromosome coding scheme. For instance, one chromosome of an example with nine jobs can be coded as the job sequence [9 3 6 5 8 7 2 4 1]. As crossover operator, we tested a partially mapped crossover (PMX) and a hybrid order and position-based crossover (OPX). The mutations are based again either on a pairwise interchange of two jobs or on a shift of one chosen job.

The PMX (partially mapped crossover) method may be the most popular crossover operator when operating with permutations. Firstly, choose two parents P1 and P2, e.g. P1 = [1 2 3 4 5 6 7 8 9] and P2 = [9 3 7 8 2 6 5 1 4], and two cutting sites along the string are randomly chosen, e.g. 3 and 7. The substrings defined by the two cutpoints are called the *mapping sections*. Secondly, exchange the two substrings between the parents to produce protochildren, and then they will be [1 2 3|8 2 6 5|8 9] and [9 3 7|4 5 6 7|1 4]. It is clear that protochildren will often lead to infeasible solutions. Then, one needs to determine the mapping relationship between the two mapping sections and finally, we legalize the offspring using this mapping relationship. In the first protochild, we can map the two infeasible genes 2 and 8 outside the mapping section, by using the mapping swaps, for instance, 2 in the first protochild's mapping section can be mapped to 5 in the second protochild's mapping section corresponding to the position. It does however not finish, because 5 is in the first protochild's mapping section as well. Again, 5 in the first protochild can be mapped to 7 in a similar way. At last, 2 in the first protochild can be swapped to 7. Similarly, 8 in the first protochild can be mapped to 4. Consequently, the first offspring is [1 ***7*** 3| 8 2 6 5| ***4*** 9]. Then, the second offspring is analogously created as [9 3 ***2***| 4 5 6 7|1 ***8***]**.**

The OPX (combined order and position-based crossover) method may be a good crossover choice, in which it creates feasible solutions like PMX and combines the characteristics of OX and PBX as well. We will create the first offspring based on OX, whereas the second offspring is characterized by PBX. Again two parents P1 and P2 are randomly selected, and consider the same example as for PMX above. Then, randomly select a substring from the first parent, e.g. [1 2 3|4 5 6 7|8 9]. Copy the substring into the first protochild corresponding to the first parent position, e.g. [_ _ _|4 5 6 7|_ _]. Then, delete all the symbols from the second parent which are already in the substring and place its symbols into the unfixed positions in the first protochild from left to right according to the second parent order, e.g. [***9 3 8***|4 5 6 7|***2 1***]. To create the second offspring, the second protochild is created by copying the symbols from the second parent, where the jobs are the same as the symbols in the substring in the corresponding position, e.g. [_ _ 7 _ _ 6 5 _ 4]. Then, place the symbols form the first parent into the unfixed positions in the second protochild from left to right according to the order of the first parent regarding the substring symbols to produce the second offspring, [***1 2*** 7 ***3 8*** 6 5 ***9*** 4].

In our tests, we investigated the influence of the choice of the initial generation, the choice of the population size, different crossover and mutation operators and the choice of probabilities for applying crossover and mutation.

5.4. Choice of an initial solution

To improve the quality of the solution finally obtained, we also investigated the influence of the choice of an appropriate initial solution for the SA and TS algorithms, and an initial population for the GA algorithm by using the heuristic constructive and improvement algorithms.

To this end, we use one or several constructive algorithm(s) SPT, LPT, ERD, EDD, MST, S/P, HSE, PAL, CDS, GUP, DAN and NEH as well as the other selected polynomial improvement heuristics as initial solution(s), respectively (for the GA algorithm, the remaining initial solutions are still randomly generated).

Table 1: Average overall performance of the constructive and polynomial improvement heuristics

λ	Problem size	CA	2-SM	A-SM	2-PI	A-PI
0	10×5	3.025[a]	1.525	**1.192**	1.650	1.200
	30×10	7.050	3.933	3.008	4.267	**2.050**
	50×20	9.567	5.717	4.575	5.550	**2.192**
	Sum	19.642	11.175	8.775	11.467	**5.442**
0.001	10×5	78.540[b]	28.530	**19.060**	32.590	21.360
	30×10	88.360	36.950	23.810	36.840	**19.040**
	50×20	35.280	12.600	12.180	9.710	**5.500**
	Sum	202.180	78.080	55.050	79.140	**45.900**
0.005	10×5	41.340	15.070	**9.490**	17.900	11.780
	30×10	40.100	16.200	10.620	17.490	**8.740**
	50×20	19.775	8.748	8.126	8.416	**4.536**
	Sum	101.215	40.018	28.236	43.806	**25.056**
0.01	10×5	29.640	10.860	**6.910**	13.530	8.430
	30×10	27.977	12.313	7.857	14.122	**6.802**
	50×20	15.136	8.373	7.512	8.679	**5.397**
	Sum	72.753	31.546	22.279	36.331	**20.629**
0.05	10×5	17.267	6.292	**4.703**	8.344	5.394
	30×10	16.803	8.225	6.185	9.980	**5.019**
	50×20	9.697	5.697	5.348	6.533	**5.035**
	Sum	43.767	20.214	16.236	24.857	**15.448**
0.1	10×5	15.783	5.520	**4.187**	8.520	4.847
	30×10	14.945	6.759	4.827	8.614	**3.761**
	50×20	9.162	5.255	5.128	5.776	**4.766**
	Sum	39.890	17.534	14.142	22.910	**13.374**
0.5	10×5	15.531	5.675	**4.043**	7.762	4.537
	30×10	14.780	7.244	5.583	8.332	**4.340**
	50×20	8.984	5.269	4.993	6.602	**4.676**
	Sum	39.295	18.188	14.619	22.696	**13.553**
1.0	10×5	15.832	5.213	**4.338**	7.894	4.617
	30×10	14.887	7.070	5.361	9.309	**4.314**
	50×20	8.879	5.340	4.862	6.051	**4.632**
	Sum	39.598	17.623	14.561	23.254	**13.563**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

6. Computational Results

Firstly, constructive algorithms according to Algorithm 2 and different fast polynomial improvement heuristics according to Algorithm 3 are studied. The constructive algorithms (denoted by the letter "CA") are simple dispatching rules such as the SPT, LPT, ERD, EDD, MST, S/P, and HSE rules, and some flow shop makespan heuristics adapted such as the algorithms PAL, CDS, GUP, DAN, and NEH. Then, we additionally applied the fast polynomial improvement heuristics based on the four cases stated above in Section 4.3 (denoted by 2-SM, A-SM, 2-PI, and A-PI, respectively). We used problems with 10 jobs × 5 stages, 30 jobs × 10 stages, and 50 jobs × 20 stages. For all problem sizes, we tested instances with $\lambda \in \{0, 0.001, 0.005, 0.01\ 0.05, 0.1, 0.5, 1\}$ in the objective function. Ten different instances for each problem size have been run.

For our experiments with problems with unrelated parallel machines, we generated the standard processing times, relative machine speeds, setup times, release dates and due dates as follows. The standard processing times are generated uniformly from the interval [10,100]. The relative speeds are distributed uniformly in the interval [0.7, 1.3]. The setup times, both sequence- and machine-dependent setup times, are generated uniformly from the interval [0, 50], whereas the release dates are generated uniformly from the interval between 0 and half of their total standard processing time mean. The due date of a job is set in a way that is similar to the approach presented by Rajendaran and Ziegler [39] and is as follows:

$$d_j = r_j + \sum_{t=1}^{k} ps_j^t + \text{total of mean setup time of a job on all stages} + (n-1) \times (\text{mean processing time of a job on one machine}) \times U(0,1) \quad (23)$$

The results for the constructive algorithms and fast polynomial improvement heuristics are given in Table 1. We give the average (absolute for $\lambda = 0$ resp. percentage for $\lambda > 0$) deviation of a particular algorithm from the best solution in these tests for the three problem sizes $n \times k$.

From these results it is obvious that the fast polynomial improvement heuristics can improve the quality of the constructive algorithms by about 60–70 percent. In addition, we have found that for the problem size 10 jobs × 5 stages the all-shift-move (A-SM) heuristic is slightly better than the others, whereas the all-pairwise-interchange-based (A-PI) improvement heuristic is the best algorithm otherwise. However, in general the all-pairwise-interchange algorithm should be selected as the improvement algorithm. Consequently, in this paper we use in the following only the all-pairwise-interchange-based improvement heuristics. However, when comparing between the 2-SM and 2-PI algorithms whose CPU time is smaller than the CPU time of both the A-SM and A-PI algorithms, we have found that the 2-SM algorithm certainly become better than the 2-PI algorithm.

Table 2: Average performance of the constructive algorithms of Group I & II

		Group I							**Group II**				
λ	**Problem size**	**SPT**	**LPT**	**ERD**	**EDD**	**MST**	**S/P**	**HSE**	**PAL**	**CDS**	**GUP**	**DAN**	**NEH**
0	10×5	3.000[a]	3.200	3.500	4.600	4.100	4.100	*2.800*	2.700	2.200	2.800	2.600	***0.700***
	30×10	*6.900*	7.900	7.700	7.900	8.400	7.900	7.200	7.700	6.100	7.300	7.800	*1.800*
	50×20	8.700	8.200	11.100	15.800	14.600	14.100	*7.900*	9.400	6.600	8.600	8.800	*1.000*
	Sum	18.600	19.300	22.300	28.300	27.100	26.100	*17.900*	19.800	14.900	18.700	19.200	***3.500***
0.001	10×5	90.920[b]	94.290	*87.880*	102.460	91.560	90.930	90.910	73.220	61.630	77.800	70.370	***10.470***
	30×10	89.090	104.510	94.730	90.250	100.510	91.170	*87.730*	101.410	77.990	94.510	98.410	*29.980*
	50×20	31.830	34.420	42.570	49.770	45.600	43.960	*30.970*	37.030	27.000	34.620	35.200	*10.320*
	Sum	211.840	233.220	225.180	242.480	237.670	226.060	*209.610*	211.660	166.620	206.930	203.980	*50.770*
0.005	10×5	45.290	*44.130*	44.710	58.240	52.180	52.520	45.250	38.870	31.010	41.540	36.150	***6.250***
	30×10	42.140	45.420	43.800	43.640	46.770	41.540	*41.360*	44.290	34.330	42.650	43.740	*11.500*
	50×20	18.812	*18.338*	23.140	28.685	26.281	25.233	18.798	20.046	15.791	18.867	18.902	*4.411*
	Sum	106.242	107.888	111.650	130.565	125.231	119.293	*105.408*	103.206	81.131	103.057	98.792	***22.161***
0.01	10×5	33.300	*30.430*	31.040	41.170	36.990	37.630	33.270	28.570	22.230	30.560	25.780	***4.710***
	30×10	30.633	30.954	30.780	31.752	33.282	*28.880*	29.870	29.870	23.506	29.744	29.561	*6.887*
	50×20	14.895	*14.199*	17.734	21.330	19.445	18.625	14.996	15.440	12.655	14.646	14.514	***3.160***
	Sum	78.828	*75.583*	79.554	94.252	89.717	85.135	78.136	73.880	58.391	74.950	69.855	***14.757***
0.05	10×5	22.154	*16.778*	17.176	21.889	20.413	19.662	21.591	18.069	12.421	19.019	15.633	***2.399***
	30×10	20.477	17.413	19.306	21.227	21.110	*16.986*	19.431	16.766	13.394	17.207	15.928	***2.386***
	50×20	10.406	*9.721*	11.872	12.748	11.476	10.838	10.425	10.355	8.400	9.898	9.457	***0.765***
	Sum	53.037	*43.912*	48.354	55.864	52.999	47.486	51.447	45.190	34.215	46.124	41.018	***5.550***
0.1	10×5	21.084	*15.177*	15.656	19.457	18.163	17.181	20.257	17.073	11.196	17.585	14.471	***2.097***
	30×10	18.691	15.309	17.482	19.453	19.007	*15.058*	17.658	14.637	11.722	15.071	13.784	***1.470***
	50×20	10.029	*9.384*	11.335	11.772	10.554	9.935	10.053	9.877	8.016	9.523	8.985	***0.479***
	Sum	49.804	*39.870*	44.473	50.682	47.724	42.174	47.968	41.587	30.934	42.179	37.240	***4.046***
0.5	10×5	21.203	*14.852*	15.456	18.446	17.310	16.114	20.073	17.373	11.176	17.221	14.448	***2.700***
	30×10	18.759	15.021	17.524	19.528	18.653	*14.916*	17.669	14.368	11.768	14.794	13.488	***0.869***
	50×20	9.985	*9.394*	11.181	11.244	10.068	9.446	10.011	9.754	7.933	9.489	8.866	***0.436***
	Sum	49.947	*39.267*	44.161	49.218	46.031	40.476	47.753	41.495	30.877	41.504	36.802	*4.005*
1.0	10×5	21.473	*15.061*	15.696	18.567	17.426	16.214	21.473	17.674	11.400	17.418	14.701	***2.885***
	30×10	18.793	15.018	17.551	19.567	18.630	*14.923*	18.793	14.367	11.785	14.780	13.477	***0.964***
	50×20	9.892	9.308	11.073	11.087	9.918	*9.296*	9.892	9.651	7.837	9.399	8.766	***0.428***
	Sum	50.158	*39.387*	44.320	49.221	45.974	40.433	50.158	41.692	31.022	41.597	36.944	***4.277***

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 3: Average performance of the polynomial improvement algorithms of Group III & IV

λ	Problem size	Group III							Group IV				
		ISPT	**ILPT**	**IERD**	**IEDD**	**IMST**	**IS/P**	**IHSE**	**IPAL**	**ICDS**	**IGUP**	**IDAN**	**INEH**
0	10×5	1.400[a]	1.300	1.200	*1.000*	1.300	1.300	1.100	1.400	1.300	1.100	1.300	***0.700***
	30×10	2.100	2.500	2.400	1.500	***1.200***	1.800	2.100	2.500	2.300	2.100	2.300	*1.800*
	50×20	*1.300*	2.300	3.000	3.000	4.900	3.300	1.700	1.700	***0.700***	2.000	1.400	1.000
	Sum	*4.800*	6.100	6.600	5.500	7.400	6.400	4.900	5.600	4.300	5.200	5.000	***3.500***
0.001	10×5	16.220[b]	24.020	*12.550*	13.710	18.940	18.870	23.880	35.200	23.980	35.630	22.840	***10.470***
	30×10	20.310	18.910	18.970	***13.200***	15.710	13.910	19.090	*17.750*	18.110	21.450	21.140	29.980
	50×20	6.730	5.150	7.690	5.010	4.040	6.960	*2.990*	8.750	2.640	3.260	***2.460***	10.320
	Sum	43.260	48.080	39.210	***31.920***	38.690	39.740	45.960	61.700	*44.730*	60.340	46.440	50.770
0.005	10×5	10.000	13.410	*8.170*	8.650	11.990	11.740	14.840	13.980	14.060	15.260	12.970	***6.250***
	30×10	9.690	8.150	8.610	7.530	7.520	*6.610*	10.300	9.650	***6.050***	9.020	10.200	11.500
	50×20	3.924	5.228	5.514	6.051	*3.554*	5.631	3.922	5.453	***3.306***	3.734	3.703	4.411
	Sum	23.614	26.788	22.294	*22.231*	23.064	23.981	29.062	29.083	23.416	28.014	26.873	***22.161***
0.01	10×5	8.890	9.450	7.030	*6.640*	7.690	9.540	9.080	8.460	9.810	10.130	9.770	***4.710***
	30×10	6.373	9.709	5.676	6.753	4.762	***4.076***	9.096	7.585	*4.935*	8.988	6.782	6.887
	50×20	*4.699*	6.427	5.749	6.264	6.890	6.583	5.198	6.183	4.934	3.431	5.251	***3.160***
	Sum	19.962	25.586	*18.455*	19.657	19.342	20.199	23.374	22.228	19.679	22.549	21.803	***14.757***
0.05	10×5	5.476	5.281	*4.900*	6.629	5.643	5.620	7.194	6.322	4.229	5.675	5.365	***2.399***
	30×10	4.820	6.313	*2.768*	6.397	5.431	4.893	6.431	5.865	4.227	5.282	5.419	***2.386***
	50×20	*4.778*	5.247	5.438	7.221	6.010	6.711	5.028	5.486	3.139	5.538	5.064	***0.765***
	Sum	15.074	16.841	*13.106*	20.247	17.084	17.224	18.653	17.673	11.595	16.495	15.848	***5.550***
0.1	10×5	*4.546*	5.404	4.787	6.318	5.721	4.877	5.763	5.749	3.752	4.154	4.996	***2.097***
	30×10	3.255	4.743	*1.718*	5.523	4.957	5.033	4.619	4.193	2.241	3.848	3.537	***1.470***
	50×20	5.169	*4.241*	5.024	6.681	5.831	5.788	5.173	5.336	3.126	5.394	4.955	***0.479***
	Sum	12.970	14.388	*11.529*	18.522	16.509	15.698	15.555	15.278	9.119	13.396	13.488	***4.046***
0.5	10×5	4.969	4.932	5.195	5.707	6.287	*4.629*	5.414	4.327	2.790	4.147	3.346	***2.700***
	30×10	3.929	5.283	*2.404*	6.727	5.135	6.897	4.730	4.936	2.812	4.611	3.745	***0.869***
	50×20	5.453	*4.244*	5.090	6.215	5.900	4.725	5.406	5.163	3.450	5.125	4.906	***0.436***
	Sum	14.351	14.459	*12.689*	18.649	17.322	16.251	15.550	14.426	9.052	13.883	11.997	*4.005*
1.0	10×5	5.018	5.073	5.268	5.741	5.935	*4.840*	5.018	4.527	3.195	4.378	3.531	***2.885***
	30×10	4.155	4.838	*2.421*	6.843	5.932	6.940	4.155	4.910	2.405	4.666	3.543	***0.964***
	50×20	5.346	*4.147*	5.107	6.079	5.731	4.523	5.346	5.044	3.808	5.046	4.976	***0.428***
	Sum	14.519	14.058	*12.796*	18.663	17.598	16.303	14.519	14.481	9.408	14.090	12.050	***4.277***

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for λ

Next, we present the results for the constructive and improvement algorithms that are separated into four main groups. The first heuristic group includes the simple dispatching rules such as SPT, LPT, ERD, EDD, MST, S/P, and HSE. The second heuristic group includes the flow shop makespan heuristics adapted such as PAL, CDS, GUP, DAN, and NEH. The third and fourth heuristic groups are generated from the first two groups of heuristics where the solutions are improved by the selected polynomial improvement algorithm based on all-pairwise-interchange improvement heuristics, and they are denoted by the first letter "I" in front of the letters describing the heuristics of the first two groups.

The results for the constructive and fast polynomial improvement algorithms are given in Table 2 and Table 3, respectively (the best variant in each group is given in italic underlined while the overall best variant is given in bold face underlined). From these results, it is obvious that the algorithms in the fourth heuristic group improved the pure makespan heuristics form the second heuristic group (i.e., PAL, CDS, GUP, DAN, and NEH), and they are better than the dispatching rules in the first heuristic group (i.e., SPT, LPT, ERD, EDD, MST, S/P, and HSE) as well as the third heuristic group improved from them.

Among the simple dispatching rules (heuristic Group I), the SPT, LPT, ERD, and HSE rules are good dispatching rules. However, in general the HSE rule outperforms the other dispatching rules for $\lambda < 0.01$, and the LPT rule is better than the other rules otherwise. Among the adapted flow shop makespan heuristics

in heuristic Group II, the NEH algorithm is clearly the best algorithm among all studied constructive and improvement heuristics (but in fact, this algorithm takes the convex combination of both criteria into account when selecting partial sequences). The CDS algorithm is certainly the algorithm on the second rank (but it is substantially worse than the NEH algorithm even if the makespan portion in the objective function value is dominant, i.e. for large λ values).

When we apply a fast pairwise interchange algorithm (denoted by the letter "I" first) to the dispatching rules and adapted makespan heuristics, we have found that the quality of the solution can be improved by about 60 – 70 percent except for the NEH rule. It can be noted that the NEH rule is not improved by using the improvement heuristics of algorithm INEH because both algorithms use a very similar strategy. However, the improvement of the heuristics from the adapted pure makespan heuristics in the heuristic Group IV is better than the improvement of the heuristics derived from the dispatching rules in the heuristic Group III. These results are similar to the conclusions of Jungwattanakit, Reodecha, Chaovalitwongse and Werner [40] whose experiments compared the results for small problem sizes with the optimal solutions.

Table 4: Tested Parameters for the SA, TS, and GA algorithms

Algorithms	**Parameters**	**Levels**
SA	Initial temperature	2, 4, 6, 8, 10 through 100, in steps of 10
	Neighborhood structures	PI, SM
	Cooling schedules	1 – 4 (geometric reduction with $\alpha \in \{0.8, 0.85, 0.9, 0.95\}$)
		5 – 13 (LM reduction with β: 0.01 through 0.09, in steps of 0.01)
		14 – 23 (LM reduction with β: 0.1 through 1.0, in steps of 0.1)
TS	Number of neighbors	10 through 50, in steps of 10
	Neighborhood structures	PI, SM
	Sizes of tabu list	5, 10, 15, and 20
GA	Population sizes	10, 30, 50, 70
	Crossover types	PMX, OPX
	Mutation types	PI, SM
	Crossover rates	0.1 through 0.9, in steps of 0.1
	Mutation rates	0.1 through 0.9, in steps of 0.1

Thirdly, we have studied the SA, TS, and GA algorithms with a random initial solution (or population). The purpose of this study is to determine the favorable parameters of the heuristics. From the preliminary tests, we set the time limit equal to one second for the problems with ten jobs, ten seconds for the problems with 30 jobs, and 30 seconds for the problems with 50 jobs. Again, for all tests we considered instances with $\lambda \in \{0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, \text{and } 1\}$. Based on the preliminary tests, the tested parameters are shown in Table 4.

From the full factorial experiment, we analyzed our results by means of a multi-factor *Analysis of Variance (ANOVA)* technique using a 5% significance level. We give the average (absolute resp. percentage) deviation of a particular iterative algorithm from the best solution obtained by the iterative algorithms. For the SA algorithm, we have found that for the neighborhood structure and the cooling schedule, there are statistically significant differences, whereas there are slightly statistically significant differences in the initial temperature. For the initial temperature, we observed that a lower initial temperature is effective for the problem. However, not all the problem cases should use the same small initial temperature. For the problems with $\lambda < 0.5$, the use of an initial temperature of two and of ten for the other problems can be recommended. It is clear that shift moves are better than pairwise interchange moves for $\lambda > 0$, especially for the problems whose makespan objective is dominant in comparison with the tardiness objective, whereas the pairwise interchange moves are slightly better than or nearly as good as the shift moves for the other values. Consequently, the neighborhood structures should be based on pairwise interchanges for $\lambda = 0$ and on shifts of jobs otherwise, or it can also be recommended to use shift moves for

the whole range of λ. For the cooling schedule, we have found that the performance of a geometric reduction is as good as the performance of the Lundy and Mees reduction only for problems with $\lambda = 0$, whereas the Lundy and Mees reduction becomes better when the value λ increases. Nevertheless, the parameter of the Lundy and Mees reduction depends on the value λ. We have found that for the problems with $\lambda < 0.5$, the cooling rate of the Lundy and Mees reduction is suitable at a range from 0.5 through 1.0 (we recommend 1.0), whereas for the other problems it is suitable at a range from 0.05 through 0.2 (we recommend 0.1).

For the TS algorithm, 20 and 30 nontabu neighbors per iteration are good parameters, but 20 nontabu neighbors are still slightly better. It is clear that pairwise interchange moves are better than shift moves for $\lambda < 0.005$, whereas for $\lambda = 0.005$ and the problem sizes 10 jobs × 5 stages and 50 jobs × 20 stages, there are not statistically significant differences in both neighborhood structures, but they are statistically significant for the problem size 30 jobs × 10 stages. For the problem size 50 jobs × 20 stages and $\lambda \geq 0.1$, although the average main effect of pairwise interchange moves is better than that of shift moves, it was found that there is a statistically significant interaction between the neighborhood structure and the number of neighbors, that is, for 20 nontabu neighbors the shift moves become better than pairwise interchange moves. Hence, in general shift moves should be selected as the neighborhood structure for $\lambda \geq 0.005$. For the size of the tabu list, it can be seen that a size of 10 and 15 works best, but the size 10 of the tabu list is slightly superior.

For the GA algorithm, we have found that for the population size, crossover types and mutation types as well as crossover and mutation rates, there are statistically significant differences. The population sizes of 30 and 50 are quite not statistically different, but in general a population size of 30 is slightly better than a population size of 50. Considering the crossover types, the OPX crossover is obviously superior to the PMX crossover. For the mutation type, it is clear that shift moves are better than pairwise interchange moves for $\lambda \geq 0.01$, whereas the pairwise interchange moves are mostly better than shift moves for the other values. It means that the pairwise interchange moves are suitable for problems whose tardiness objective dominates the makespan objective. Consequently, the neighborhood structures should be based on pairwise interchanges for $\lambda < 0.01$ and on shifts of jobs otherwise. Given the above selected GA parameters, we analyzed the crossover and mutation rates again. There are no significant differences in these parameters. However, in general we recommend a crossover rate of 0.6 and a mutation rate of 0.3.

Finally, we used the recommended SA, TS, and GA parameters to test the choice of an appropriate initial solution (or a part of the initial population for the GA approach). The letters before the letters SA (or TS, or GA) denote the heuristic rule as one initial solution for SA (or TS, or GA). For example, LPTSA means that the LPT rule is used as an initial solution for the SA algorithm, RNDTS means that an initial solution in TS is randomly generated, or NEHGA means that one initial solution of the initial population in the GA approach is generated by the NEH rule but the other initial solutions are still randomly generated.

In Table 5, we compared the LPT, ILPT, NEH, and INEH rules as well as the iterative algorithms with random initial solution and population, respectively (denoted by RNDSA, RNDTS, and RNDGA). We give the average (absolute resp. percentage) deviation of a particular algorithm from the best solution in these tests. We have found that iterative metaheuristic algorithms can improve the quality of the LPT rule by about 90 percent or even more. Among the iterative algorithms, the results show that for the RNDSA and RNDTS algorithms there are not statistically significant differences, but in general, the RNDSA algorithm is better than the RNDTS algorithm.

In Table 6, we selected the SPT, LPT, S/P, HSE, ISPT, ILPT, IS/P, IHSE, PAL, CDS, NEH, IPAL, ICDS, and INEH rules as the initial solution under consideration. The letter CA denotes the whole group of constructive algorithms considered. The letter C before the letters SA, TS, and GA denote the SA, TS, and GA algorithms using the best of the constructive algorithms as an initial solution. In addition, for the GA approach, we used some selected algorithms in parallel as a part of the initial population. Based on each heuristic group, we used all solutions in each heuristic group stated above as a part of the initial population (the other initial solutions are still randomly generated). Consequently, we have four new choices of initial

populations tested (denoted by MIX1GA, MIX2GA, MIX3GA, and MIX4GA, respectively). We have found that the SA-based algorithms are still good algorithms. In addition, the GA-based algorithms can be improved by using a group of biased initial solutions as a part of the initial population instead of all randomly generated or only one biased initial solution.

Concerning the choice of an initial solution, we only show the results for the recommended SA, TS, and GA parameters to test an appropriate selection of an initial solution in Table 7. We have found that for the SA algorithm, there are no statistically significant differences when using different initial solutions. We have however found that the ILPTSA, NEHSA and INEHSA rules are slightly better than the others in general. Consequently, the ILPTSA, NEHSA and INEHSA algorithms are good choices for the SA algorithm with using a biased initial solution. However, our experiments have shown that there are slightly statistically significant differences for different initial solutions. The NEHTS, INEHTS, NEHGA and INEHGA rules are however still good solutions when compared within each group too.

Finally, we present the results of some algorithms for small-size problems with a number of jobs ranging from three to seven. We give the average deviation from the optimal solution obtained by means of the 0-1 mixed linear integer programming formulation given in Section 3 using a commercial mathematical programming software, CPLEX 8.0.0 and AMPL, with an Intel Pentium 4 2.00GHz CPU. However, the CPU time is limited to at most 2 hours.

Table 5: Average performance of the constructive and iterative algorithms

λ	Problem Size	LPT	ILPT	NEH	INEH	RNDSA	RNDTS	RNDGA
0	10×5	3.200[a]	1.300	0.700	0.700	**0.000**	0.040	0.100
	30×10	8.400	3.000	2.300	2.300	0.300	**0.260**	0.680
	50×20	8.400	2.500	1.200	1.200	**0.040**	**0.040**	0.400
	Sum	20.000	6.800	4.200	4.200	**0.340**	**0.340**	1.180
0.001	10×5	99.380[b]	26.900	13.320	13.320	**0.030**	0.060	3.430
	30×10	115.570	24.880	36.810	36.810	6.320	**5.880**	9.460
	50×20	35.302	5.650	10.921	10.921	**0.748**	0.917	1.694
	Sum	250.252	57.430	61.051	61.051	7.098	**6.857**	14.584
0.005	10×5	46.353	15.036	7.862	7.862	**0.031**	0.946	1.067
	30×10	56.309	16.126	19.711	19.711	**3.899**	4.167	7.785
	50×20	19.539	6.237	5.427	5.427	**0.946**	1.617	2.512
	Sum	122.201	37.399	33.000	33.000	**4.876**	6.730	11.364
0.01	10×5	31.949	10.704	5.944	5.944	**0.160**	0.565	1.359
	30×10	40.655	17.738	14.730	14.730	**3.497**	4.214	6.731
	50×20	15.380	7.504	4.199	4.199	**0.923**	1.669	2.079
	Sum	87.984	35.946	24.873	24.873	**4.580**	6.448	10.169
0.05	10×5	17.358	5.831	2.936	2.936	**0.015**	0.039	0.701
	30×10	24.011	12.349	8.167	8.167	**3.054**	4.059	6.164
	50×20	11.311	6.766	2.218	2.218	**0.806**	2.187	2.650
	Sum	52.680	24.946	13.321	13.321	**3.875**	6.285	9.515
0.1	10×5	15.885	6.054	2.720	2.720	0.082	**0.077**	0.747
	30×10	21.987	10.890	7.440	7.440	4.098	**3.705**	6.333
	50×20	10.779	5.565	1.755	1.755	**1.099**	1.880	2.266
	Sum	48.651	22.509	11.915	11.915	**5.279**	5.662	9.346
0.5	10×5	15.493	5.523	3.284	3.284	**0.013**	0.052	0.630
	30×10	21.179	10.949	6.291	6.291	**3.089**	3.682	5.769
	50×20	10.547	5.337	1.492	1.492	**0.888**	2.076	2.432
	Sum	47.219	21.809	11.067	11.067	**3.990**	5.810	8.831
1.0	10×5	15.486	5.474	3.279	3.279	**0.041**	0.075	0.607
	30×10	21.079	10.419	6.324	6.324	**3.965**	3.365	5.692
	50×20	10.591	5.365	1.605	1.605	**1.021**	2.012	2.666
	Sum	47.156	21.258	11.208	11.208	**5.027**	5.452	8.965

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for λ

Table 6: Average performance of the iterative algorithms with biased initial solutions

λ	Problem size	CA	C-SA	C-TS	C-GA	MIX1GA	MIX3GA	MIX2GA	MIX4GA
0	10×5	1.943[a]	**0.001**	0.006	0.089	0.120	0.040	**0.100**	0.060
	30×10	4.829	0.274	**0.237**	0.794	0.520	0.780	0.360	0.720
	50×20	5.050	**0.014**	0.056	0.474	0.460	0.160	0.200	0.080
	Sum	11.821	**0.290**	0.299	1.357	1.100	0.980	0.660	0.860
0.001	10×5	51.554[b]	**0.096**	0.421	2.487	3.600	2.630	2.560	3.010
	30×10	59.438	5.719	4.909	9.640	8.270	10.540	**3.080**	8.710
	50×20	19.131	0.854	0.923	2.410	2.481	1.286	0.488	**0.344**
	Sum	130.123	6.668	6.253	14.537	14.351	14.456	**6.128**	12.064
0.005	10×5	26.798	**0.206**	0.791	1.433	1.778	1.303	1.271	1.431
	30×10	32.128	4.198	**3.834**	7.380	6.433	6.990	4.945	6.122
	50×20	12.033	0.894	1.463	3.070	3.400	1.557	**0.564**	0.973
	Sum	70.958	**5.298**	6.088	11.883	11.611	9.850	6.780	8.526
0.01	10×5	19.264	**0.205**	0.502	1.044	1.445	1.233	0.904	1.595
	30×10	24.923	**3.781**	4.030	7.693	6.412	8.697	5.147	6.042
	50×20	10.475	**0.924**	1.532	3.362	2.431	1.515	1.382	1.013
	Sum	54.662	**4.910**	6.064	12.099	10.288	11.445	7.433	8.650
0.05	10×5	11.247	0.051	**0.047**	0.844	0.756	0.547	0.277	0.696
	30×10	16.360	**3.745**	4.100	6.521	6.499	4.872	5.042	4.525
	50×20	8.117	**0.891**	2.064	3.207	3.037	1.484	2.686	1.131
	Sum	35.725	**4.687**	6.210	10.572	10.292	6.903	8.005	6.352
0.1	10×5	10.408	0.049	**0.025**	0.608	0.855	0.476	0.659	0.294
	30×10	14.940	4.107	**3.868**	5.762	6.160	4.527	4.905	4.647
	50×20	7.573	1.107	1.963	3.019	2.827	1.105	2.983	**1.103**
	Sum	32.921	**5.263**	5.856	9.389	9.842	6.108	8.547	6.044
0.5	10×5	10.134	**0.020**	0.034	0.640	0.646	0.418	0.688	0.449
	30×10	14.625	**3.316**	3.636	5.808	5.941	3.647	4.942	3.724
	50×20	7.249	**0.757**	1.820	2.764	2.331	0.961	2.724	0.811
	Sum	32.008	**4.093**	5.490	9.212	8.918	5.026	8.354	4.984
1.0	10×5	10.181	**0.036**	0.051	0.582	0.691	0.492	0.566	0.408
	30×10	14.565	3.764	3.554	5.696	5.292	3.732	4.530	**3.255**
	50×20	7.312	0.993	1.809	2.987	2.366	0.950	2.861	**0.875**
	Sum	32.058	4.792	5.414	9.264	8.349	5.174	7.957	**4.538**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

In the tests, we used again problems with the same generation of the data and $\lambda \in \{0, 0.001, 0.005, 0.01$ 0.05, 0.1, 0.5, 1} in the objective function. We considered the constructive algorithms LPT and NEH, the fast improvement algorithms ILPT and INEH, and the iterative metaheuristic algorithms RNDSA, NEHSA, RNDTS, NEHTS, RNDGA and NEHGA.

From the results in Table 8, it can be observed that the ILPT improvement algorithm can improve the constructive LPT rule by about 50 – 70 percent in terms of the deviation from the optimal value. The performance of the NEH and INEH rules as well as the ILPT rule are not significantly different. As stated above for the large-size problems, the NEH rule has already embedded the improvement routine as used by the ILPT algorithm. However, in contrast to large-size problems it can be observed that the ILPT algorithm is slightly better than NEH and INEH for problems with $0.001 \leq \lambda \leq 0.05$. For the iterative algorithms such as RNDSA, RNDTS, and RNDGA, we have found that they can improve the quality of the solution of the constructive and fast improvement algorithms such as ILPT, NEH and INEH by about 50 – 70 percent. Similar to the large-size problems, the SA-based algorithms certainly outperform the other algorithms. In addition, we have found that a biased initial solution for the iterative algorithm is slightly better than a random initial solution.

Table 7: Average performance of iterative algorithms with biased initial solutions

		SA-Based Algorithms						TS-Based Algorithms						GA-Based Algorithms (1)						GA-Based Algorithms (2)			
λ	Problem size	LPTSA	ILPTSA	CDSSA	NEHSA	ICDSSA	INEHSA	LPTTS	ILPTTS	CDSTS	NEHTS	ICDSTS	INEHTS	LPTGA	ILPTGA	CDSGA	NEHGA	ICDSGA	INEHGA	MIX1GA	MIX2GA	MIX3GA	MIX4GA
0	10×5	***0.000***[a]	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	0.020	***0.000***	0.100	0.120	0.120	*0.060*	0.120	*0.060*	0.120	*0.040*	0.100	0.060
	30×10	*0.260*	0.280	*0.260*	0.300	0.300	0.300	0.220	0.220	0.340	0.180	0.300	***0.160***	*0.760*	0.900	0.780	0.880	0.880	0.880	0.520	0.780	*0.360*	0.720
	50×20	***0.000***	***0.000***	***0.000***	***0.000***	0.040	**0.000**	0.060	0.120	0.020	***0.000***	0.040	***0.000***	0.580	0.960	0.580	*0.240*	0.340	*0.240*	0.460	0.160	0.200	*0.080*
	Sum	*0.260*	0.280	*0.260*	0.300	0.340	0.300	0.280	0.340	0.360	0.180	0.360	***0.160***	1.440	1.980	1.480	*1.180*	1.340	*1.180*	1.100	0.980	*0.660*	0.860
0.001	10×5	0.520[b]	***0.010***	0.030	0.020	0.510	0.030	*0.090*	0.570	0.100	0.170	1.060	0.530	2.160	2.990	2.480	*2.070*	3.660	*2.070*	3.600	2.630	*2.560*	3.010
	30×10	5.380	6.050	5.600	5.870	*5.220*	5.880	5.470	4.640	4.520	*4.220*	5.760	*4.220*	10.680	9.310	*8.230*	10.340	8.430	10.340	8.270	10.540	***3.080***	8.710
	50×20	0.954	*0.778*	0.831	0.844	0.802	0.940	1.211	1.287	0.817	*0.615*	0.652	0.622	1.839	3.610	1.933	*1.323*	2.114	1.259	2.481	1.286	0.488	***0.344***
	Sum	6.854	6.838	*6.461*	6.734	6.532	6.850	6.771	6.497	5.437	***5.005***	7.472	5.372	14.679	15.910	*12.643*	13.733	14.204	13.669	14.351	14.456	*6.128*	12.064
0.005	10×5	0.115	0.236	0.531	***0.041***	0.060	0.259	*0.984*	1.279	1.179	1.076	1.103	0.721	*1.134*	1.275	1.437	1.374	1.899	1.374	1.778	1.303	*1.271*	1.431
	30×10	*3.999*	4.203	4.249	4.264	4.727	4.320	3.897	3.785	***3.687***	4.075	4.051	4.183	8.038	7.726	7.481	7.055	*6.482*	7.055	6.433	6.990	*4.945*	6.122
	50×20	0.969	0.866	*0.799*	0.888	0.923	0.917	1.524	2.046	1.175	0.838	0.975	*0.842*	2.464	4.560	2.157	1.426	3.183	*1.426*	3.400	1.557	***0.564***	0.973
	Sum	***5.083***	5.305	5.579	5.193	5.710	5.496	6.405	7.110	6.041	5.989	6.129	*5.746*	11.636	13.561	11.075	9.855	11.564	*9.855*	11.611	9.850	*6.780*	8.526
0.01	10×5	***0.084***	0.163	0.212	0.382	0.086	0.324	0.551	0.481	0.779	0.664	*0.387*	0.706	1.283	1.111	*0.859*	0.873	1.273	0.873	1.445	1.233	*0.904*	1.595
	30×10	3.701	3.957	***3.450***	4.103	4.064	4.104	4.389	4.172	*3.677*	3.892	3.934	3.819	*7.098*	8.423	7.383	8.586	7.142	8.586	6.412	8.697	*5.147*	6.042
	50×20	0.965	0.969	***0.895***	0.936	0.933	0.957	1.245	2.171	1.767	*0.931*	1.517	0.931	2.706	5.873	2.239	*1.498*	4.247	*1.498*	2.431	1.515	1.382	*1.013*
	Sum	4.750	5.089	***4.557***	5.421	5.083	5.385	6.185	6.824	6.223	5.487	5.838	*5.456*	11.087	15.407	*10.481*	10.957	12.662	10.957	10.288	11.445	*7.433*	8.650
0.05	10×5	0.038	*0.032*	0.054	0.049	0.040	0.049	0.033	0.052	***0.029***	0.042	0.054	0.029	*0.567*	0.793	0.926	0.625	0.859	0.625	0.756	0.547	*0.277*	0.696
	30×10	4.176	3.369	3.903	4.016	*3.301*	4.005	4.654	3.823	4.165	***3.250***	4.378	3.356	6.440	8.122	6.803	*4.730*	7.754	*4.730*	6.499	4.872	5.042	*4.525*
	50×20	0.896	0.852	0.918	0.748	0.881	***0.740***	1.963	2.152	2.430	*1.491*	1.894	*1.491*	3.133	5.216	3.199	*1.382*	3.472	*1.382*	3.037	1.484	2.686	*1.131*
	Sum	5.110	4.253	4.875	4.813	***4.222***	4.794	6.650	6.027	6.624	*4.783*	6.326	4.876	10.140	14.131	10.928	*6.737*	12.085	*6.737*	10.292	6.903	8.005	*6.352*
0.1	10×5	*0.033*	0.064	0.038	0.070	0.047	0.070	0.057	0.035	0.025	0.030	0.021	***0.012***	0.688	0.523	0.669	*0.504*	0.540	*0.504*	0.855	0.476	0.659	*0.294*
	30×10	4.794	*3.305*	4.647	3.581	3.705	3.581	4.067	4.061	3.724	3.260	3.786	***3.254***	5.945	6.834	5.444	*4.219*	6.190	*4.219*	6.160	*4.527*	4.905	4.647
	50×20	1.067	1.105	1.168	***0.813***	0.861	0.837	2.142	2.264	2.141	*1.369*	1.994	1.370	2.484	4.394	2.892	1.091	3.441	*1.086*	2.827	1.105	2.983	*1.103*
	Sum	5.894	4.474	5.853	***4.464***	4.613	4.488	6.266	6.360	5.890	4.659	5.801	*4.636*	9.117	11.751	9.005	5.814	10.171	*5.809*	9.842	6.108	8.547	*6.044*
0.5	10×5	0.016	0.014	***0.009***	0.013	0.014	0.013	*0.015*	0.035	0.039	0.039	0.041	0.039	0.787	0.887	0.720	*0.537*	0.671	*0.537*	0.646	*0.418*	0.688	0.449
	30×10	3.228	3.500	3.571	3.140	4.027	*3.153*	3.864	3.721	3.894	***2.854***	3.753	2.904	5.853	7.253	5.490	*3.775*	5.813	*3.775*	5.941	3.647	4.942	*3.724*
	50×20	0.776	0.794	0.826	0.715	0.689	***0.687***	1.895	2.099	2.157	1.170	1.723	*1.157*	2.150	4.194	2.401	*0.982*	3.585	*0.982*	2.331	0.961	2.724	*0.811*
	Sum	4.020	4.308	4.406	3.868	4.730	***3.853***	5.774	5.855	6.090	4.063	5.517	*4.100*	8.790	12.334	8.611	*5.294*	10.069	*5.294*	8.918	5.026	8.354	*4.984*
1.0	10×5	0.020	***0.006***	0.059	0.038	0.035	0.038	0.059	0.065	0.069	0.047	0.024	*0.035*	0.626	0.591	0.604	0.458	*0.387*	0.458	0.691	0.492	0.566	*0.408*
	30×10	4.467	***2.588***	4.071	3.183	3.709	3.148	3.813	3.726	3.763	*3.024*	3.471	3.030	5.461	7.428	5.580	*3.533*	5.579	3.547	5.292	3.732	4.530	*3.255*
	50×20	0.858	1.066	1.184	***0.588***	1.014	0.588	1.821	2.068	2.199	*1.198*	1.874	*1.198*	2.367	4.351	2.523	*1.040*	4.303	*1.040*	2.366	0.950	2.861	*0.875*
	Sum	5.345	***3.660***	5.314	3.809	4.758	3.774	5.693	5.859	6.031	4.269	5.369	*4.263*	8.454	12.370	8.707	*5.031*	10.269	5.045	8.349	5.174	7.957	*4.538*

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 8: Average performance of the constructive, fast improvement and metaheuristic algorithms for small-size problems

λ	Problem size	LPT	ILPT	NEH	INEH	RNDSA	NEHSA	RNDTS	NEHTS	RNDGA	NEHGA
0	3 jobs	0.300	**0.050**	0.100	0.100	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**
	4 jobs	0.750	0.250	0.250	0.250	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**
	5 jobs	0.700	0.250	**0.150**	**0.150**	**0.150**	**0.150**	0.190	0.200	**0.150**	**0.150**
	6 jobs	1.200	0.400	0.450	0.450	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**
	7 jobs	1.150	0.500	0.450	0.450	**0.200**	**0.200**	**0.200**	**0.200**	**0.200**	**0.200**
	Sum	4.100	1.450	1.400	1.400	**0.700**	**0.700**	0.740	0.750	**0.700**	**0.700**
0.001	3 jobs	45.270	5.180	8.610	8.610	**4.950**	**4.950**	**4.950**	**4.950**	**4.950**	**4.950**
	4 jobs	88.390	32.670	37.460	37.460	**7.350**	**7.350**	7.500	7.480	**7.350**	**7.350**
	5 jobs	23.570	2.320	5.700	5.700	**1.870**	**1.870**	2.270	2.320	**1.870**	**1.870**
	6 jobs	97.960	33.860	27.320	27.320	**13.370**	**13.370**	**13.370**	**13.370**	**13.370**	**13.370**
	7 jobs	23.960	7.140	7.200	7.200	**0.970**	**0.970**	**0.970**	**0.970**	**0.970**	**0.970**
	Sum	279.150	81.170	86.290	86.290	**28.510**	**28.510**	29.060	29.090	**28.510**	**28.510**
0.005	3 jobs	14.218	3.797	5.367	5.367	**3.569**	**3.569**	3.771	4.111	**3.569**	**3.569**
	4 jobs	29.860	9.370	12.480	12.480	**4.400**	**4.400**	4.410	4.420	**4.400**	**4.400**
	5 jobs	15.182	2.383	3.756	3.756	**1.627**	**1.627**	**1.627**	**1.627**	**1.627**	**1.627**
	6 jobs	31.400	8.990	9.070	9.070	**3.440**	**3.440**	3.480	3.480	**3.440**	**3.440**
	7 jobs	14.180	4.960	4.970	4.970	**2.240**	**2.240**	**2.240**	**2.240**	**2.240**	**2.240**
	Sum	104.840	29.500	35.643	35.643	**15.276**	**15.276**	15.528	15.878	**15.276**	**15.276**
0.01	3 jobs	9.820	3.371	4.379	4.379	**3.143**	**3.143**	3.550	3.550	**3.143**	**3.143**
	4 jobs	20.475	5.812	8.182	8.182	**3.350**	**3.350**	3.423	3.370	**3.350**	**3.350**
	5 jobs	11.884	2.334	2.899	2.899	**1.450**	**1.450**	**1.450**	**1.450**	**1.450**	**1.450**
	6 jobs	20.018	5.542	5.677	5.677	**2.154**	**2.154**	2.165	**2.154**	2.164	2.155
	7 jobs	10.528	2.917	2.903	2.903	**1.096**	**1.096**	**1.096**	**1.096**	1.102	**1.096**
	Sum	72.725	19.976	24.040	24.040	**11.193**	**11.193**	11.684	11.620	11.209	11.194
0.05	3 jobs	7.048	3.388	3.267	3.267	**2.887**	**2.887**	3.046	2.934	**2.887**	**2.887**
	4 jobs	12.651	3.366	4.521	4.521	**2.084**	**2.084**	**2.084**	**2.084**	**2.084**	**2.084**
	5 jobs	8.846	2.576	1.778	1.778	**0.899**	**0.899**	**0.899**	**0.899**	**0.899**	**0.899**
	6 jobs	9.344	2.516	3.256	3.256	**1.042**	**1.042**	**1.042**	**1.042**	1.047	1.047
	7 jobs	6.291	0.960	0.650	0.650	**0.192**	**0.192**	**0.192**	**0.192**	0.229	**0.192**
	Sum	44.180	12.806	13.472	13.472	**7.104**	**7.104**	7.263	7.151	7.146	7.109
0.1	3 jobs	6.828	3.343	3.149	3.149	**2.929**	**2.929**	3.011	3.011	**2.929**	**2.929**
	4 jobs	12.301	3.529	4.315	4.315	**2.059**	**2.059**	**2.059**	**2.059**	**2.059**	**2.059**
	5 jobs	8.705	2.417	1.574	1.574	**0.889**	**0.889**	**0.889**	**0.889**	**0.889**	**0.889**
	6 jobs	9.172	2.656	2.186	2.186	**0.949**	**0.949**	**0.949**	**0.949**	**0.949**	**0.949**
	7 jobs	5.906	0.615	0.335	0.335	**0.146**	**0.146**	**0.146**	**0.146**	0.184	0.155
	Sum	42.912	12.560	11.559	11.559	**6.972**	**6.972**	7.054	7.054	7.010	6.981
0.5	3 jobs	6.760	3.475	3.237	3.237	**3.123**	**3.123**	3.295	3.360	**3.123**	**3.123**
	4 jobs	12.063	2.776	4.137	4.137	**1.921**	**1.921**	1.945	**1.921**	**1.921**	**1.921**
	5 jobs	8.531	2.215	1.548	1.548	**0.745**	**0.745**	**0.745**	**0.745**	**0.745**	**0.745**
	6 jobs	8.607	2.620	2.514	2.514	**1.238**	**1.238**	**1.238**	**1.238**	1.269	**1.238**
	7 jobs	6.840	1.581	0.391	0.391	**0.113**	**0.113**	**0.113**	**0.113**	0.149	**0.113**
	Sum	42.801	12.667	11.827	11.827	**7.140**	**7.140**	7.336	7.377	7.207	**7.140**
1.0	3 jobs	6.750	3.492	3.247	3.247	**3.142**	**3.142**	3.182	3.191	**3.142**	**3.142**
	4 jobs	12.038	2.745	4.107	4.107	**1.877**	**1.877**	**1.877**	**1.877**	**1.877**	**1.877**
	5 jobs	9.458	3.192	2.441	2.441	**1.712**	**1.712**	**1.712**	**1.712**	**1.712**	**1.712**
	6 jobs	9.368	2.811	2.747	2.747	**1.518**	**1.518**	**1.518**	**1.518**	1.545	1.531
	7 jobs	7.834	2.139	0.843	0.843	**0.297**	**0.297**	0.332	**0.297**	0.315	0.306
	Sum	45.448	14.379	13.385	13.385	**8.546**	**8.546**	8.621	8.595	8.591	8.568

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

7. Conclusions

In this paper, we have investigated both constructive and iterative (SA, TS, and GA-based algorithms) approaches for minimizing a convex combination of makespan and the number of tardy jobs for the flexible flow shop problem with unrelated parallel machines and setup times, which is often occurring e.g. in the textile industry. All algorithms are based on the list scheduling principle by developing job sequences for the first stage and assigning and sequencing the remaining stages by both the permutation and FIFO approaches. The constructive algorithms are compared to each other. It is shown that the NEH algorithm is an excellent constructive algorithm for minimizing the objective function considered. In particular, the NEH algorithm is most superior to the other constructive algorithms. When applying a fast polynomial improvement algorithm, we have found that the all-pairwise-interchange algorithm is a good improvement algorithm.

Moreover, we used SA-based algorithms as iterative algorithms. We tested the SA parameters, i.e., initial temperatures, neighborhood structures, and cooling schedules. We have found that a low initial temperature is slightly preferable (we recommend two for $\lambda < 0.5$ and ten otherwise). The neighborhood structures should be based on pairwise interchanges for $\lambda = 0$ and on shifts of jobs otherwise. The Lundy and Mees cooling scheme $T_{new} = T_{old}/(1+\beta \times T_{old})$ is recommended. In the TS algorithm, we studied the TS parameters, i.e., neighborhood structures, size of the tabu list, and the number of generated neighbors per iteration. Similar to the SA algorithm, the pairwise interchange neighborhood is better for $\lambda < 0.005$, whereas the shift neighborhood becomes better otherwise. We can recommend a tabu list size of 10 and the generation of a constant number of 20 nontabu neighbors in each iteration. For the GA algorithm, we have found that the OPX crossover is clearly superior to the PMX crossover, whereas we recommend that the SM neighborhood should be selected as the mutation operator for problems with $\lambda \geq 0.01$, and the PI neighborhood otherwise. We have fixed the crossover and mutation rates at 0.6 and 0.3, respectively.

For the recommended SA, TS, and GA parameters, we investigated the performance of these algorithms RNDSA, RNDTA, and RNDGA. We have found that the RNDSA algorithm outperforms the other algorithms. Then, we studied the influence of the initial solution on these algorithms. The results have shown that the SA-based algorithms are still good algorithms. However, among the SA-, TS- and GA-based algorithms, the NEH and INEH rules are a good choice of an initial solution in general.

Further research can be done to use other iterative algorithms such as ant colony algorithms. The choice of good parameters for them should be tested. In addition, the influence of the starting solution should be investigated. Moreover, hybrid algorithms should be developed by using a local search algorithm within a GA. This means that, after generating an offspring, this solution should be improved by applying for instance tabu search or simulated annealing before applying the selection criterion of GA.

Acknowledgements. This work was supported in part by INTAS (project 03-51-5501).

References

[1] Karacapilidis NI, Pappis CP. Production planning and control in textile industry: A case study. Computers in Industry 1996;30(2): 127–144.

[2] Agnetis A, Pacifici A, Rossi, F, Lucertini M, Nicoletti S, Nicolò F, Oriolo G, Pacciarelli D, Pesaro E. Scheduling of flexible flow lines in an automobile assembly plant. European Journal of Operational Research 1997:97(2): 348–362.

[3] Alisantoso D, Khoo LP, and Jiang PY. An immune algorithm approach to the scheduling of a PCB flexible flow shop. The International Journal of Advanced Manufacturing Technology 2003;22(11–12): 819–827.

[4] Hsieh JC, Chang PC, Hsu LC. Scheduling of drilling operations in printed circuit board factory. Computers & Industrial Engineering 2003;44(3): 461–473.

[5] Gupta JND, Krüger K, Lauff V, Werner F, Sotskov YN. Heuristics for hybrid flow shops with controllable processing times. Computers and Operations Research 2002;29(10): 1417–1439.

[6] Lin HT, Liao CJ. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. International Journal of Production Economics 2003;86(2): 133–143.
[7] Wang W, Hunsucker JL. An evaluation of the CDS heuristic in flow shops with multiple processors. Journal of the Chinese Institute of Industrial Engineers 2003;20(3): 295–304.
[8] Linn R, Zhang W. Hybrid flow shop scheduling: A survey. Computers & Industrial Engineering 1999;37(1–2): 57–61.
[9] Wang W. Flexible flow shop scheduling: Optimum, heuristics, and artificial intelligence solutions. Expert Systems 2005;22(2): 78–85.
[10] Arthanari TS, Ramamurthy KG. An extension of two machines sequencing problem. Opsearch 1971;8(1): 10–22.
[11] Salvador MS. A solution to a special case of flow shop scheduling problems. In: Elmaghraby SE (Ed.). Symposium of the Theory of Scheduling and Applications. New York: Springer, 1973, p. 83–91.
[12] Brah SA, Hunsucker JL. Branch and bound algorithm for the flow shop with multiple processors. European Journal of Operational Research 1991;51(1): 88–99.
[13] Moursli O, Pochet Y. Branch and bound algorithm for the hybrid flowshop. International Journal of Production Economics 2000;64(1–3): 113–125.
[14] Gupta JND. Two-stage, hybrid flow shop scheduling problem. Journal of Operational Research Society 1988;39(4): 359–364.
[15] Sriskandarajah C, Sethi SP. Scheduling algorithms for flexible flowshops: worst case and average case performance. European Journal of Operational Research 1989;43(2): 143–160.
[16] Guinet A, Solomon MM, Kedia PK, Dussauchoy A. A computational study of heuristics for two-stage flexible flowshops. International Journal of Production Research 1996;34(5): 1399–1415.
[17] Gupta JND, Tunc EA. Scheduling a two-stage hybrid flowshop with separable setup and removal times. European Journal of Operational Research 1994;77(3): 415–428.
[18] Santos DL, Hunsucker JL, Deal DE. An evaluation of sequencing heuristics in flow shops with multiple processors. Computers & Industrial Engineering 1996;30(4): 681-691.
[19] Gourgand M, Grangeon N, Norre S. Metaheuristics for the deterministic hybrid flow shop problem. Proceeding of the International Conference on Industrial Engineering and Production Management (IEPM'99), Glasgow, United Kingdom, July 12-15 1999. p. 136–145.
[20] Jin Z, Yang Z, Ito T. (2006). Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. International Journal of Production Economics 2006;100(2): 322–334.
[21] Koulamas C, Antony SR, Jaen R. A survey of simulated annealing applications to operations research problems. Omega International Journal of Management Science 1994;22(1): 41–56.
[22] Nowicki E, Smutnicki C. The flow shop with parallel machines: A tabu search approach. European Journal of Operational Research 1998;106(2–3): 226–253.
[23] Werner F. On the solution of special sequencing problems. Dissertation, TU Magdeburg, Germany, 1984.
[24] Reeves CR. A genetic algorithm for flowshop sequencing. Computers and Operations Research 1995;22(1): 5-13.
[25] Cheng R, Gen M, Tozawa M. Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. Computers and Industrial Engineering 1995;29(1-4): 513-517.
[26] Ruiz R, Maroto C, Alcaraz, J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. European Journal of Operational Research 2005;165(1): 34–54.
[27] Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY. A survey of scheduling problems with setup times or cost. European Journal of Operational Research 2006. (to appear).
[28] Baker KR. Introduction to Sequencing and Scheduling. New York: John Wiley & Sons; 1974.
[29] Pinedo M, Chao X. Operations scheduling with applications in manufacturing and services. New York: Irwin/McGraw-Hill; 1999.
[30] Palmer DS. Sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum. Operations Research Quarterly 1965;16(1): 101–107.
[31] Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the n-job m-machine sequencing problem. Management Science 1970;16(10): 630–637.
[32] Gupta JND. A functional heuristic algorithm for the flow-shop scheduling problem. Operations Research Quarterly 1971;22(1): 39-47.

[33] Dannenbring DG. An evaluation of flow shop sequencing heuristics. Management Science 1977;23(11): 1174-1182.
[34] Nawaz M, Enscore Jr E, Ham I. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. Omega International Journal of Management Science 1983;11(1): 91–95.
[35] Kirkpatrick S, Gelatt CD Jr, Vecchi MP. Optimization by simulated annealing. Science 1984;220(4598): 671–68.
[36] Lundy M, Mees A. Convergence of an annealing algorithm. Mathematical Programming 1986;34(1): 111–124.
[37] Glover F. Future paths for integer programming and links to artificial intelligence. Computers and Operations Research 1986;13(5); 533–549.
[38] Holland JA. Adaptation in natural and artificial systems. Ann Arbor: University of Michigan; 1975.
[39] Rajendran C, Ziegler H. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. European Journal of Operational Research 2003;149(3): 513–522.
[40] Jungwattanakit J, Reodecha M, Chaovalitwongse P, Werner F. (2006). Sequencing heuristics for flexible flow shop scheduling problems with unrelated parallel machines and setup times. Proceedings of the 2006 IE Network National Conference, Bangkok, Thailand, December 18-19 2006. (to appear)