Jitti Jungwattanakit · Manop Reodecha · Paveena Chaovalitwongse · Frank Werner

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

In textile industries, production facilities are established as multi-stage production flow shop facilities where a production stage may be made up of parallel machines. It is known as flexible or hybrid flow shop environment. This paper considers the problem of scheduling n independent jobs in such an environment. In addition, we also consider the general case in which parallel machines in each stage may be unrelated. Each job is processed in ordered operations on a machine of each stage. Its release date and due date are given. Preemption of jobs is not permitted. We consider both sequence- and machine-dependent setup times. The problem is to determine a schedule that minimizes a convex combination of makespan and the number of tardy jobs. A 0-1 mixed integer program of the problem is formulated. Since this problem class is NP-hard in the strong sense, we develop heuristic algorithms to solve this problem approximately. Firstly, several basic dispatching rules and well-known constructive heuristics for flow shop makespan scheduling problems are generalized to the problem under consideration. We sketch how from a job sequence a complete schedule for the flexible flow shop problem with unrelated parallel machines can be constructed. Then genetic algorithms are suggested. We discuss the components of these algorithms and test their parameters. 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.

Keywords: Flexible flow shop scheduling; Mathematical programming; Constructive algorithms; Genetic algorithms;

1 Introduction

Production scheduling can be defined as the allocation of available production resources over time to perform a collection of tasks [1]. It is an important decision making process in the operation level. In a modern manufacturing environment, many scheduling problems occur. Most of the scheduling problems are very significant and hard to solve owing to the complex nature of the problems. A textile facility environment, an example of a scheduling problem, will be discussed in this paper.

In textile manufacturing, products that contain fibers, such as clothing, tires, and yarn are made. Its production unit can hardly fit in any classical scheduling model. Instead, such a production unit is characterized by multi-stage production flow shop facilities, where a production stage may be made up of parallel production lines, machines or any other production facility. At some stages, the facilities (machines, lines, etc.) are duplicated in parallel in order to increase the overall capacities of the shop floor, or in order to balance the capacities of the stages, or either to eliminate or reduce the impact of bottleneck stages on the overall shop floor capacities known as flexible flow shop, multiprocessor flow shop, or hybrid flow shop environment.

In addition, setup times and costs incur when machines often have to be reconfigured or cleaned between jobs. This process is known as a changeover or setup. If the length of the setup depends on the job just completed and on the one about to be started, then the setup times are sequence-dependent which often occurs in such an industry. For instance, in the weaving operation, the setup times depend on the types of clothes being processed in sequence. Another example arises in the dying operation. Every time a new color is used, the coloring devices must be cleaned. The cleanup time often depends on the color just used as well as the color about to be used.

A flexible flow shop environment is a generalization of the classical flow shop model. There are k stages and some stages may have only one machine, but at least one stage must have multiple machines. The jobs have to visit the stages in the same order string from stage one through stage k. 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 such a processing is not allowed. The problem consists of assigning 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 [2], Alisantoso, Khoo, and Jiang [3], Lin and Liao [4] and Wang and Hunsucker [5]. 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,

J. Jungwattanakit, M. Reodecha, and P. Chaovalitwongse
Department of Industrial Engineering, Faculty of Engineering,
Chulalongkorn University, Bangkok 10330 Thailand

F. Werner (✉)
Faculty of Mathematics, Otto-von-Guericke University Magdeburg,
P.O. Box 4120, 39016 Magdeburg, Germany
phone +49-391-6712025, fax: +49-391-6711171,
e-mail: frank.werner@mathematik.uni-magdeburg.de

the flexible flow shop problem with unrelated parallel machines is considered, that is, 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 is given by Linn and Zang [6] and Wang [7]. Most of the earlier literature has considered the simple case of only two stages. Arthanari and Ramamurthy [8] and Salvador [9] 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 multi-stage flexible flow shop problems are proposed by many authors. Brah and Hunsucker [10] present a branch and bound algorithm, and it is claimed that the method could also be used to optimize other criteria than makespan. Portmann, Vignier, Dardihac and Dezalay [11] improve the branch and bound method of Brah and Hunsucker by improving their lower bound and reducing the number of branches used in the search tree. Moursli and Pochet [12] propose a branch and bound algorithm for the hybrid flow shop scheduling problem, each stage being composed of identical parallel machines.

When an exact algorithm is applied to large flexible flow shop problems in particular, the optimum 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 [13] proposes heuristic techniques for a simplified flexible flow shop makespan problem with two stages and only one machine in the second stage. The proposed heuristics are based on extensions of Johnson's algorithm. Sriskandarajah and Sethi [14] develop simple heuristic algorithms for the two-stage flexible flow shop problem. They discuss the worst and average case performance of algorithms of finding minimum makespan schedules. Their solutions are based on Johnson's rule. Guinet, Solomon, Kedia, and Dussauchoy [15] propose a heuristic for the makespan minimization 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 makespan problem in a two-stage flexible flow shop. Gupta and Tunc [16] consider the two-stage flow shop scheduling problem when 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 one. Santos, Hunsucker and Deal [17] 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 such a heuristic is then followed by the application of the First in First out (FIFO) rule.

The question raised for the constructive algorithms is whether it is possible to improve their solution quality. Stagnation in a local optimum is one drawback of constructive algorithms, while most iterative algorithms (or artificial intelligent search techniques) always try to find a better solution or escape from a local optimum to reach globally better solutions. Jones, Mirrazavi, and Tamiz [18] state that 70% of the articles utilize genetic algorithms as the primary metaheuristic, 24% simulated annealing and only 6% tabu search. The genetic algorithm is so popular due to the flexibility of this technique. Thus, to determine near-optimal solutions, a genetic algorithm is proposed as an iterative algorithm in this paper as well.

Genetic algorithms are stochastic search methods for optimization problems based on the mechanism of natural selection and genetics by using the concept of the survival of the fittest tenet and offspring creation of Darwinian evolution. Recently, a genetic algorithm has been applied to harder combinatorial optimization problems [19, 20] because it has better characteristics, e.g. there is a smaller effect on the calculations when the system becomes more complex or larger. For example, Reeves [21] applies a genetic algorithm to the flow shop makespan problem. Cheng, Gen, and Tozawa [22] address the earliness/tardiness scheduling problem with identical parallel machines. A genetic algorithm is also applied to solve this problem. Ruiz, Maroto, and Alcaraz [23] 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 (see for instance the recent review on scheduling with setup times by Allahverdi, Ng, Cheng, and Kovalyov [24]). Thus, in this paper we will investigate how to apply a genetic algorithm to solve the flexible flow shop problem with unrelated parallel machines.

Although the genetic algorithm has gained many applications, it is reported that the traditional genetic algorithm often suffers from the trouble of premature convergence, the difficulty in constructing fitness functions and parameter dependence. So far, many improvements have been proposed to enhance the performance of a genetic algorithm, especially by the use of particular initial populations or a hybrid genetic algorithm.

The purpose of this paper is to present and compare several constructive and genetic algorithms to solve the flexible flow shop scheduling problem with unrelated parallel machines and sequence- and machine-dependent setup times. In addition, while many papers have studied only the makespan criterion, we consider this scheduling problem with two optimization criteria. One reason for this consideration is the increasing pressure of high competition while customers expect ordered goods to be delivered on time.

The rest of the paper is organized as follows: The problem considered in this paper is described in Section 2. The mathematical model for this problem is introduced in Section 3. In Section 4, constructive heuristics for the flexible flow shop problem are sketched and improvement heuristics are proposed in Section 5 whereas Section 6 and Section 7 discuss the variants of the genetic algorithm. Computational results for the heuristics are briefly discussed in Section 8. Conclusions are given in Section 9.

2 Problem description

In a flexible flow shop problem, a set of n independent customer orders, $j \in \{1, 2, ..., n\}$, with due dates, $d_1 , ... , d_n$ has to be processed. Since customer orders may not arrive simultaneously in real-life problems, we assume that they are available at non-negative release times, $r_1 , ... , r_n$. Each job j consists of a chain of k ($k \geq 2$) different operations, O_j^1, ..., O_j^k, which have to be processed in this order. Each job j has its fixed standard processing time ps_j^t for every stage t, $t \in \{1, 2, ..., k\}$. Moreover, at each stage t, there is a set of m^t unrelated parallel machines, $i \in \{1, 2, ..., m^t\}$, where some of the production stages may have only one machine, but at least one production stage must have multiple machines. Hence, machine i at stage t can process job j at the relative speed v_{ij}^t. The processing time p_{ij}^t of job j on machine i at stage t is equal to ps_j^t / v_{ij}^t.

This problem assumes a static and deterministic scheduling environment. There are no precedence relationships between jobs. Preemption is not allowed; i.e., once an operation is started, it must be completed without interruption. The operations of a job have to be realized sequentially, without overlapping between stages. Job splitting is not permitted.

Setup times considered in this problem are classified into two types: (1) a machine-dependent setup time and (2) a sequence-dependent setup time. A machine-dependent setup time deals with the setup time that depends on the machine to which a job is assigned. It is assumed to occur only when a job is assigned to a machine at the first position at some stage. It means that the length of the setup time (or changeover time) ch_{ij}^t of job j if job j is assigned to machine i in the first position at any stage t depends on the machine performing it. Hence, a machine-dependent setup time is concerned with the job in the first sequence at any stage, whereas a sequence-dependence setup time is considered between the further jobs. The setup time s_{lj}^t between job l and job j at stage t, where job l is processed directly before job j on the same machine, might have different values depending on both the job just completed and the job to be processed (sequence-dependent). All setup times are known and constant.

The following two restrictions should also be fulfilled: (1) no machine can process more than one job at the same time; (2) no job can be processed by more than one machine at the same time. In addition, the release date a_i^t (or machine availability) of a machine for every stage is not necessarily equal to zero. However, the machines are continuously available from this availability time.

Let the completion time of job j be C_j, then

$$C_{max} = \max_{j \in \{1..n\}} \{ C_j \}.$$

Moreover, let $U_j = 1$ if due date for job j is smaller than the completion time C_j of job j, otherwise $U_j = 0$. The total number of tardy jobs (η_T) is defined as

$$\eta_T = \sum_{j=1}^{n} U_j$$

The objective is to seek a schedule that minimizes a convex combination of makespan and the number of tardy jobs. Thus, the objective function value is defined by

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

where $0 \leq \lambda \leq 1$. λ denotes the weight (or relative importance) given to C_{max} and η_T.

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, $t = 1, 2, 3, \ldots, k$
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
m^t	number of parallel machines at stage t
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}	the makespan
U_j	a Boolean variable; 1 if job j is tardy, and 0 otherwise
T_j	tardiness of job j
η_T	the total number of tardy jobs in the schedule

3.2 Mathematical formulation

The problem can be formulated as follows.

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

Subject to:

$$\sum_{i=1}^{m^t}\sum_{j=0}^{n} X_{ijl}^t = 1 \qquad ,\forall t, l \tag{2}$$

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

$$\sum_{l=1}^{n+1} X_{i0l}^t = 1 \qquad ,\forall t, i \tag{4}$$

$$\sum_{j=0}^{n} X_{ij(n+1)}^t = 1 \qquad ,\forall t, i \tag{5}$$

$$X_{ijj}^t = 0 \qquad ,\forall t, i, j \tag{6}$$

$$\sum_{j=0}^{n} X_{ijl}^t = \sum_{j=1}^{n+1} X_{ilj}^t \qquad ,\forall t, i, j \tag{7}$$

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

$$O_j^t = \sum_{i=1}^{m^t}\sum_{l=1}^{n+1} \frac{ps_j^t}{v_{ij}^t} X_{ijl}^t \qquad ,\forall t, j \tag{9}$$

$$C_l^t - C_j^t \geq s_{jl}^t + O_l^t + ((\sum_{i=1}^{m^t} X_{ijl}^t) - 1)B \qquad ,\forall t, j, l; j \neq l \tag{10}$$

$$C_j^t \geq 0 \qquad ,\forall t, j \tag{11}$$

$$C_l^t - C_l^{t-1} \geq \sum_{i=1}^{m^t}\sum_{j=1}^{n} X_{ijl}^t s_{jl}^t + \sum_{i=1}^{m^t} ch_{il}^t X_{i0l}^t + O_l^t \qquad ,\forall t, l \tag{12}$$

$$C_j^0 = r_j \qquad ,\forall j \tag{13}$$

$$C_j^t \geq \sum_{i=1}^{m^t} a_i^t X_{i0j}^t + \sum_{i=1}^{m^t} ch_{il}^t X_{i0l}^t + O_j^t \qquad ,\forall t, j \tag{14}$$

$$C_{\max} \geq C_j^k \qquad ,\forall j \tag{15}$$

$$T_j \geq C_j^k - d_j \qquad ,\forall j \tag{16}$$

$$T_j \geq 0 \qquad ,\forall j \tag{17}$$

$$U_j \leq BT_j \qquad ,\forall j \tag{18}$$

$$BU_j \geq T_j \qquad ,\forall j \tag{19}$$

$$\eta_T = \sum_{j=1}^{n} U_j \qquad ,\forall j \tag{20}$$

$$U_j \in \{0,1\} \qquad ,\forall j \tag{21}$$

Equation (1) describes the objective function. We have $X_{i0l}^t = 1$ if the 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 (2)–(8) ensure that the partial schedule on each machine at each stage is feasible. Constraint sets (2) and (3) ensure that only one job is assigned to each sequence position at each stage. Constraint sets (4) and (5) ensure that only one job will be assigned to the first and last positions, respectively, on each machine at each stage. Constraint (6) assures that after the job has been finished at any stage, it cannot be reprocessed at the same stage. Constraint (7) forces to construct a consistent sequence at every stage. Constraint (8) specifies the decision variables X_{ijl}^t as binary variables. Constraint (9) determines the operating time of every job which is dependent on the machine. Constraints (10)–(14) find the completion time of every job. Constraint (10) 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., greater than the sum of all job processing times and setup times. Constraint (11) ensures that the completion time of every job at each stage is a non-negative value. Constraint (12) 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 (13) applies only to stage one, saying that a job cannot start its processing at stage one before its release date. Constraint (14) 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 (15) links the makespan decision variable. Constraint sets (16) and (17) determine the correct value of the tardiness (T_j). Constraint set (16) determines the correct value of the lateness (L_j) and (17) specifies only the positive lateness as the tardiness ($T_j = \max\{0,\ C_j^k - d\}$). Constraint sets (18)–(21) link the decision variable of the number of tardy jobs, that is, if tardiness is larger 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 six 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 therefore 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 [17]). Firstly, a job sequence is determined according to a particular sequencing rule using modified flow shop algorithms as shown in detail in this 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. In other words, the permutation flexible flow shop sequencing problem determines the order of processing the jobs on all machines. We will make use of both procedures and briefly discuss the modifications for the problem under consideration.

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. A greedy algorithm 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 or from the permutation rule). The objective of the greedy scheduling algorithm 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.

In order to determine the job sequence for the first stage by some heuristic, 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. This algorithm is as follows.

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 $s_{ij}^{/t}$. Nine combinations of relative speeds and setup times will be used in our algorithm. 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, while other are used mainly for comparision purposes and to have a broad spectrum of solutions in the initial population. As the basic dispatching rules, we use the rules given by Shortest Processing Time (SPT), Longest Processing Time (LPT), Earliest Release Date first (ERD), Earliest Due Date first (EDD), Minimum Slack Time first (MST), Slack time per Processing time (S/P), and Hybrid SPT and EDD (HSE) rules, whereas as flow shop makespan heuristics, we use the algorithms given by Palmer [25], Campbell, Dudek, and Smith [26], Gupta [27], and Dannenbring [28] and the insertion heuristic by Nawaz, Enscore, and Ham [29]. Notice that the first four algorithms try to minimize makespan while the insertion heuristcs can be used for any regular optimization criterion and for the multi-criteria problem under consideration too.

4.1 Dispatching rules

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. In the single machine model, the SPT rule is proved that the sum of completion times and mean lateness are minimized by using it [1]. For parallel machines, the LPT rule tends to balance the workload over the machines. The reasoning is to keep jobs with short processing times to be assigned and sequenced without affecting the workload balance. However, due to the flexible flow shop problems with unrelated parallel machine, we have to adapt the SPT and LPT rules by using the representative of the operating time as stated above. Then, the best solution is selected among the nine combinations of relative machine speeds and setup 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 ERD rule in a sense minimizes the variation in the waiting times of the jobs at a machine, whereas the EDD rule tends to minimize the maximum lateness [30]. 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. Again, the jobs are subsequently ranked in a non-decreasing order of each rule, and nine combinations of representatives of relative speeds and setup times are considered (expect for the EDD rule).

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.

4.2 Palmer

A heuristic developed by Palmer [25], in an effort to use Johnson's rule, proposes 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. It means that the first stage sequence can be generated based upon a non-increasing order of the slope indices.

Now, the modified Palmer's method (in the following denoted by PAL) for the flexible flow shop problem with unrelated parallel machines and sequence-dependent setup times is developed as follows.

Let ps_l^t be the standard processing time of job j at stage t, $v_{ij}^{/t}$ be the representative of the relative speed on machine i at stage t for job j, and $s_{lj}^{/t}$ be the representative of the setup time between job l and job j at stage t. Then $S(j, v_{ij}^{/t}, s_{lj}^{/t})$ denotes the slope index for job j at the relative speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$. PAL's slope index for the flexible flow shop problem with unrelated parallel machines and setup times is calculated as follows:

$$S(j, v_{ij}^{/t}, s_{lj}^{/t}) = -\sum_{t=1}^{k}\left\{[k-(2t-1)](\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})\right\}$$

4.3 Campbell, Dudek, and Smith

Campbell, Dudek, and Smith [26] 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. Let g be a counter for the $k-1$ sub-problems. Again, due to the unrelated parallel machines, the constructed processing time for the "first" stage is denoted as $a(j, g, v_{ij}^{/t}, s_{lj}^{/t})$ where j denotes the job, g denotes the g-th sub-problem and $v_{ij}^{/t}$ and $s_{lj}^{/t}$ are the representatives of the relative speed and setup time, respectively. Similarly, $b(j, g, v_{ij}^{/t}, s_{lj}^{/t})$ denotes the "second" stage processing time. Given these definitions, the constructed processing times are calculated according to the following two equations:

$$a(j, g, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{g}(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})$$

and

$$b(j, g, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=k-g+1}^{k}(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})$$

4.4 Gupta

Gupta [27] provides an algorithm denoted by GUP, in a similar manner as algorithm PAL by using a slope index. Denote by $G(j, v_{ij}^{/t}, s_{lj}^{/t})$ the slope index of algorithm GUP for job j at relative speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$ which is calculated from

$$G(j, v_{ij}^{/t}, s_{lj}^{/t}) = \frac{e_j}{\min\limits_{1 \le g \le k-1}\{(\frac{ps_j^g}{v_{ij}^{/g}} + s_{lj}^{/g}) + (\frac{ps_j^{g+1}}{v_{ij}^{/g+1}} + s_{lj}^{/g+1})\}}$$

and

$$e_j = \begin{cases} 1 & if \quad (\frac{ps_j^1}{v_{ij}^{/1}} + s_{lj}^{/1}) > (\frac{ps_j^k}{v_{ij}^{/k}} + s_{lj}^{/k}) \\ -1 & if \quad (\frac{ps_j^1}{v_{ij}^{/1}} + s_{lj}^{/1}) \le (\frac{ps_j^k}{v_{ij}^{/k}} + s_{lj}^{/k}) \end{cases}$$

After calculating $G(j, v_{ij}^{/t}, s_{lj}^{/t})$ for all jobs, the jobs are subsequently ranked in a non-decreasing order of the slope indices.

4.5 Dannenbring

Like PAL's rule, Dannenbring [28] develops a method 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. In the following, this method is denoted by DAN. Moreover, let $a(j, v_{ij}^{/t}, s_{lj}^{/t})$ and $b(j, v_{ij}^{/t}, s_{lj}^{/t})$ be the processing times for the constructed two-stage problem. They are calculated using the following equations:

$$a(j, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{k}\left\{(k-t+1)(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})\right\}$$

and

$$b(j, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{k}\left\{t \times (\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t})\right\}$$

4.6 Nawaz, Enscore and Ham

Nawaz, Enscore and Ham [29] develop the probably best constructive heuristic method for the permutation flow shop makespan problem, called 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 that gives the best objective function value under consideration. However, the relative position of the two previous job 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 the NEH algorithm 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. Contrary to the algorithms presented before, the NEH algorithm constructs job sequences by considering the minimization of the convex combination of makespan and tardy number of jobs.

5 Improvement Heuristics

After several constructive algorithms have been adapted in the previous section, we now describe a fast polynomial improvement heuristic applied to the solutions found by the constructive algorithms.

Since the constructive algorithms (especially some algorithms that are adapted from the flow shop makespan heuristics and some dispatching rules such as SPT, LPT rules without due date considerations) do not explicitly minimization of number of tardy jobs, in this section we will improve the solution by concerning the due date criterion.

In order to find a satisfactory solution to the multicriteria problem considered in this paper, we apply the following heuristic by using the shift neighborhood as a polynomial improvement mechanism based on the idea that we will consider the jobs that are tardy and move them randomly left and right in order to improve the overall criterion value.

Step 1: Find a first stage sequence solution given for each combination of relative machine speed and setup times using a heuristic from Section 4 and set it as the current sequence solution and sequence list ($\mathcal{L}$).

Step 2: **While** Sequence list ($\mathcal{L}$) is not empty **do**

- Choose the first job j in the sequence list.
- If due date $d_j \leq$ completion time C_j^k, we randomly move the selected job left and right, and find the best sequence solution among the two neighbors and update the best sequence solution as the current sequence solution.
- Delete job j from the sequence list.

end while

Reminding that there are nine combination solutions generated from a heuristic in each iteration and find the best solution for the multi-criteria problem under consideration among them. In step 2, there are generated at most $2\times(n\text{-}1)$ neighbors, if all jobs under consideration are late, i.e. at most $O(n)$ job sequences are investigated by the improvement algorithm.

6 A genetic algorithm

A genetic algorithm (GA) developed by Holland [31] 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.

A GA is characterized by a parallel search of the state space in contrast to a point-by-point search by conventional optimization techniques. The parallel search is achieved by keeping a set of possible solutions under consideration, called a *population*. An individual in the population is a string of symbols, and it is an abstract representation of a solution. The algorithm starts with an 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 some *fitness* measure to describe quantitatively how well the individual masters its task. The population of chromosomes evolves from some generation to the next 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 *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.

6.1 Representation

The application of a GA requires the representation of a solution. It is the primary and key issue to encode the problem into a search solution for the GA. Consideration of a job permutation is straightforward and widely used in many previous works on GA for the flow shop problem (see e.g. Werner [32]). Thus, in our GA, 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].

6.2 Initialization

Generally, the initial population is generated in a random way from the solution space. Later we also use an initial population, when one or several particular solutions obtained by constructive algorithms are included.

6.3 Evaluation

During each generation, chromosomes are evaluated using some measure of fitness. In most optimization applications, the fitness function is constructed based on the original objective function. The fitness value of each chromosome is a key measure to guide the direction of the search in GA. Due to the minimization problem, the fitness value must be in inverse proportion to the objective function value so that a fitter chromosome has a larger fitness value.

$$fitness(v_z) = \frac{1}{f(v_z)} \quad , z = 1,..., population\_size;$$

where $fitness(v_z)$ is the fitness value and $f(v_z)$ is the objective function value of the z-th chromosome for the complete schedule generated from the corresponding job sequence for the first stage using a greedy algorithm (see Section 4).

In this paper, the objective is to minimize a convex combination of makespan and the number of tardy jobs. Thus, the fitness value of a chromosome, $fitness(v_z)$, is given by

$$fitness(v_z) = \frac{1}{\lambda C_{\max}(v_z) + (1-\lambda)\sum_{j=1}^{n} U_j(v_z) + 1} \quad , z = 1,..., population\_size;$$

where $C_{max}(v_z)$ is the makespan of the z-th chromosome (resp. of the resulting complete schedule), $U_j(v_z)$ is a Boolean variable for job j of the z-th chromosome which is equal to 1 if job j is tardy, and 0 otherwise, and λ denotes the weight given to makespan and number of tardy jobs. The largest value of the fitness function is the lowest value of the convex combination of makespan and the number of tardy jobs. In the denominator, value one is added in order to prevent a division by zero when the weight λ and the number of tardy jobs are equal to zero.

6.4 Selection

An elitist policy and enlarged sampling space technique are used. Both parents and the offspring have the same chance of completing for survival. Then Holland's *proportionate selection* or *roulette wheel selection* is employed to reproduce the next generation based on the current enlarged population. The idea is to determine a selection probability (also called survival probability) for each chromosome proportional to its fitness value. For the chromosome v_z with fitness $fitness(v_z)$, its selection probability $prob(v_z)$ is calculated as follows:

$$prob(v_z) = \frac{fitness(v_z)}{\sum_{z=1}^{population\_size + offspring\_size} fitness(v_z)}$$

6.5 Crossover

Crossover and mutation are the most important parts of a GA. The crossover is used in GA to exchange information among chromosomes to introduce offspring. Based on a job permutation encoding, the partially mapped crossover (PMX) is mostly used, because it always creates a feasible offspring. In this paper, we will propose a new crossover by combining the order crossover (OX) and the position-based crossover (PBX), and we denote it as OPX. Both OX and PBX create only one offspring, whereas PMX create a couple of offspring. Thus, the proposed crossover can create a couple of offspring like PMX as well, that is both can be compared in a fair way.

6.5.1 PMX

PMX (partially mapped crossover) 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 *mapping sections*. Secondly, exchange two substrings between parents to produce protochilderen, 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 not however 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***]**.**

6.5.2 OPX

OPX (combined order and position-based crossover) 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 from the last section. Then, randomly select a substring from the first parent, e.g. [1 2 3|4 5 6 7|8 9]. Copy the substring into 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 that jobs are the same as the symbols in the substring in 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].

6.6 Mutation

Mutation serves to prevent all solutions in the population from falling into a local optimum. In this paper, two mutation operations named pairwise interchange move, and shift move are tested.

6.6.1 Pairwise interchange move

A pairwise interchange move (PI) exchanges a pair of genes π_r, and π_i, where $1 \le i, r \le n$ and $i \ne r$. Such an operation swaps the gene at position r and one at position i — $\pi' = [\pi_1, \ldots, \pi_{r-1}, \pi_i, \pi_{r+1}, \ldots, \pi_{i-1}, \pi_r, \pi_{i+1}, \ldots, \pi_n]$. For example, assume that randomly one parent, [4 9 8 7 3 1 6 2 5] is selected, and then randomly the couple of gene positions to be exchanged is selected, e.g. positions 1 and 3. Thus, the offspring will be [***8*** 9 ***4*** 7 3 1 6 2 5]. The pairwise interchange neighborhood has $n(n-1)/2$ neighbors.

6.6.2 Shift move

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

6.7 Termination criteria

In fact, after many generations of evolution throughout the repeated applications of selection, crossover, and mutation, the individuals in the population will often begin to look alike. At this point, the GA typically terminates because additional evolution will produce little improvement in fitness. Several termination criteria may be used, where the most simple one is to stop just after some predetermined number of generations. However, in this paper we will use a certain CPU time limit as the termination criterion.

7 Choice of an initial population

A GA has been proven to be effective for many combinatorial optimization problems [19, 20], and it seems natural to apply such an approach to scheduling problems. To improve the quality of the solution finally obtained, we also investigated the influence of the choice of an appropriate initial population by using the constructive algorithms.

To this end, we used as one initial solution that obtained from the constructive algorithms SPT, LPT, ERD, EDD, MST, S/P, HSE, PAL, CDS, GUP, DAN and NEH, as well as the other polynomial improvement heuristics respectively (the other initial solutions are still randomly generated). In addition, we used all selected constructive algorithms in parallel as a part of the initial population.

8 Computational results

Firstly, we studied the constructive algorithms that are separated into four main groups. The first heuristic group are the simple dispatching rules such as SPT, LPT, ERD, EDD, MST, S/P, and HSE. The second heuristic group are 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 polynomial improvement algorithm (based on shift moves), and they are denoted by the first letter "I" in front of the letters describing the heuristics of the first two groups. We used problems with 10 jobs × 5 stages, 30 jobs × 10 stages, and 20 jobs × 10 stages. For all problem sizes, we tested instances with $\lambda \in \{0, 0.05, 0.1, 0.5,$ and $1\}$ in the objective function. Ten different instances for each problem size have been run.

Table 1 Average performance of constructive algorithms

λ	Problem size	SPT	LPT	ERD	EDD	MST	SPP	HSE
0	10×5	2.3	1.7	2.8	3.1	3.2	3.0	2.4
	30×10	6.1	7.0	6.4	10.5	10.4	10.3	6.0
	50×20	5.9	7.1	7.2	14.7	14.7	12.8	5.6
	Sum	14.3	15.8	16.4	28.3	28.3	26.1	14
0.05	10×5	18.019	12.04	23.393	23.259	21.379	21.269	18.029
	30×10	17.115	14.069	18.933	20.220	17.544	16.943	15.844
	50×20	8.204	7.951	9.812	11.411	10.627	9.544	8.616
	Sum	43.338	34.06	52.138	54.890	49.550	47.756	42.489
0.1	10×5	17.068	11.024	22.058	21.810	19.369	19.668	17.005
	30×10	16.154	12.677	17.994	18.669	15.928	15.254	14.871
	50×20	8.076	7.694	9.663	10.411	9.596	8.711	8.439
	Sum	41.298	31.395	49.715	50.890	44.893	43.633	40.315
0.5	10×5	17.004	10.74	21.682	21.457	18.345	18.736	16.948
	30×10	15.902	12.075	17.790	17.908	15.078	14.242	14.483
	50×20	8.095	7.576	9.663	9.692	8.823	8.108	8.364
	Sum	41.001	30.391	49.135	49.057	42.246	41.086	39.795
1.0	10×5	16.888	10.647	21.551	21.32	18.106	18.508	16.888
	30×10	15.998	12.122	17.869	17.938	15.090	14.23	15.998
	50×20	8.09	7.549	9.656	9.591	8.714	8.017	8.090
	Sum	40.976	30.318	49.076	48.849	41.910	40.755	40.976

λ	Problem size	PAL	CDS	GUP	DAN	NEH
0	10×5	1.9	1.2	1.8	2.0	**0.5**
	30×10	6.1	4.3	5.9	5.8	**0.5**
	50×20	8.2	5.1	6.4	7.8	**0.8**
	Sum	16.2	10.6	14.1	15.6	**1.8**
0.05	10×5	10.892	9.095	13.322	10.717	**1.815**
	30×10	16.143	11.527	14.063	14.121	**0.021**
	50×20	7.58	6.555	7.709	8.107	**0**
	Sum	34.615	27.177	35.094	32.945	**1.836**
0.1	10×5	9.906	7.814	12.225	9.545	2.136
	30×10	15.146	10.513	12.856	13.163	**0**
	50×20	7.216	6.292	7.517	7.739	**0.026**
	Sum	32.268	24.619	32.598	30.447	**2.162**
0.5	10×5	9.884	7.222	12.001	9.308	2.563
	30×10	14.82	10.089	12.387	12.821	**0.067**
	50×20	7.028	6.166	7.486	7.544	**0.076**
	Sum	31.732	23.477	31.874	29.673	**2.706**
1.0	10×5	9.813	7.047	11.871	9.175	2.46
	30×10	14.902	10.156	12.438	12.902	**0.087**
	50×20	6.996	6.139	7.472	7.510	**0.048**
	Sum	31.711	23.342	31.781	29.587	**2.595**

Table 1 Average performance of constructive algorithms

λ	Problem size	ISPT	ILPT	IERD	IEDD	IMST	ISPP	IHSE
0	10×5	0.9	0.8	1.4	1.3	1.6	1.4	1.0
	30×10	2.3	3.5	3.0	4.3	4.8	4.6	2.9
	50×20	3.1	4.5	4.1	6.3	5.0	8.1	4.1
	Sum	6.3	8.8	8.5	11.9	11.4	14.1	8
0.05	10×5	5.801	4.889	8.02	7.244	6.675	5.588	6.855
	30×10	9.299	7.602	10.022	10.591	11.228	7.282	7.448
	50×20	4.508	5.195	4.474	5.952	6.539	6.143	5.092
	Sum	19.608	17.686	22.516	23.787	24.442	19.013	19.395
0.1	10×5	5.502	2.909	6.436	7.004	6.140	6.76	6.071
	30×10	8.720	6.869	9.834	9.022	10.883	6.908	9.005
	50×20	4.829	4.785	5.106	6.252	5.627	5.578	4.783
	Sum	19.051	14.563	21.376	22.278	22.65	19.246	19.859
0.5	10×5	7.048	3.068	4.135	7.819	6.247	5.559	4.842
	30×10	8.690	6.435	9.506	10.257	8.482	7.513	9.936
	50×20	5.211	4.681	4.818	5.296	6.286	5.219	4.928
	Sum	20.949	14.184	18.459	23.372	21.015	18.291	19.706
1.0	10×5	6.776	2.548	4.393	8.101	6.932	5.699	5.869
	30×10	8.882	6.641	9.645	11.081	8.711	6.882	8.759
	50×20	5.015	4.668	5.027	5.622	5.830	5.377	4.948
	Sum	20.673	13.857	19.065	24.804	21.473	17.958	19.576

λ	Problem size	IPAL	ICDS	IGUP	IDAN	INEH
0	10×5	0.9[a]	**0.5**	0.8	1.1	**0.5**
	30×10	3.3	2.7	3.6	3.3	**0.5**
	50×20	4.3	2.5	3.5	4.7	**0.8**
	Sum	8.5	5.7	7.9	9.1	**1.8**
0.05	10×5	3.701[b]	1.949	4.449	2.251	**1.815**
	30×10	7.336	7.070	8.984	7.950	**0.021**
	50×20	3.717	4.174	4.266	5.703	**0**
	Sum	14.754	13.193	17.699	15.904	**1.836**
0.1	10×5	3.402	**1.939**	3.555	3.501	2.136
	30×10	8.350	6.838	6.836	7.680	**0**
	50×20	3.485	4.986	4.958	4.865	**0.026**
	Sum	15.237	13.763	15.349	16.046	**2.162**
0.5	10×5	2.937	1.901	2.792	**1.446**	2.563
	30×10	7.221	5.487	7.946	6.639	**0.067**
	50×20	3.898	4.755	4.684	4.291	**0.076**
	Sum	14.056	12.143	15.422	12.376	**2.706**
1.0	10×5	3.144	**1.661**	3.073	1.952	2.46
	30×10	6.671	5.962	6.425	5.811	**0.087**
	50×20	4.546	4.14	4.717	4.126	**0.048**
	Sum	14.361	11.763	14.215	11.889	**2.595**

[a] average absolute deviation for $\lambda = 0$

[b] average percentage deviation for $\lambda > 0$

An experiment was conducted to test with data such as the standard processing times, relative machine speeds, setup times, release dates and due dates. The standard processing times are generated uniformly from the interval [10,100]. Due to the unrelated machine problem, 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 [0,20]. The due date of a job is set in a way that it is similar to the approach presented by Rajendaran and Ziegler [33] and is as follows:

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

where the mean processing time of a job on one machine is determined by summing the mean setup time and standard processing time of all jobs on all stages and dividing by the number of machines.

The results for the constructive algorithms are given in Table 1. We give the average (absolute for λ = 0 resp. percentage for λ > 0) deviation of a particular constructive algorithm from the best constructive solution for three problem sizes n× k.

From these results it is obvious that the constructive algorithms in the fourth heuristic group improved from flow shop makespan heuristics from the second heuristic group (i.e., PAL, CDS, GUP, DAN, and NEH) are better than the dispatching rules in the first heuristic group (i.e., SPT, LPT, EDD, MST, S/P, and HSE) as well as the third heuristic group improved from them.

Among the simple dispatching rules (heuristic Group I), HSE rule outperforms the other dispatching rules for λ = 0, and the LPT rule is better than the other rules for λ > 0. Among the adapted flow shop makespan heuristics in heuristic Group II, NEH algorithm is clearly the best algorithm among all studied constructive 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 NEH even if the makespan portion in the objective function value is dominant, i.e. for large λ values).

When we apply the polynomial improvement (‘reinsertion‘) algorithm (denoted as the letter “I” first) to the solutions obtained by the dispatching rules and adapted flow shop makespan heuristics, we have found that the quality of the solution in terms of the deviation from the best solution can be improved by about 50 percentage except for NEH rule (it is even independent of the concrete value of λ). It is noted that the NEH rule is not improved by using the improvement heuristics in INEH because the NEH rule is embedded by the insertion algorithm itself (it confirms the excellent solution quality by algorithm NEH). However, the improvement of heuristics from the adapted flow shop makespan heuristics in the heuristic Group IV is better than the improvement of heuristics from the dispatching rules in the heuristic Group III (since for most of the problems, there is an substantial portion in the objective function value resulting from the makespan).

Secondly, we studied the GA with random initial population. The purpose of this study is to determine the favorable GA parameters, i.e., population size, crossover types, mutation types, as well as crossover and mutation rates.

Given the above three different problem sizes, the following GA parameter values were used in this test.
Population size : 30, 50, 70
Crossover type : PMX, OPX
Mutation type : PI, SM
Crossover rate : 0.1 through 0.9, in steps of 0.1
Mutation rate : 0.1 through 0.9, in steps of 0.1

Table 2 The effect of various population sizes on the performance of the genetic algorithm

λ	Problem size	30	50	70
0	10×5	**0**[a]	**0**	**0**
	30×10	2.0162	**1.9159**	2.2693
	50×20	0.4051	**0.3194**	0.4144
	Sum	2.4213	**2.2353**	2.6837
0.05	10×5	1.2516[b]	**1.5950**	1.1617
	30×10	**3.4810**	3.9820	4.7860
	50×20	**1.5130**	1.7580	1.9880
	Sum	**6.2456**	7.3350	7.9357
0.1	10×5	1.3760	1.3430	**1.2540**
	30×10	**2.8070**	3.1730	3.8990
	50×20	**1.4780**	1.7710	2.0200
	Sum	**5.6610**	6.2870	7.1730
0.5	10×5	1.5832	1.4729	**1.4307**
	30×10	**2.9740**	3.2940	3.9870
	50×20	**1.2990**	1.6440	1.7990
	Sum	**5.8562**	6.4109	7.2167
1.0	10×5	1.6149	1.4609	**1.4021**
	30×10	**2.9260**	3.2730	3.9370
	50×20	**1.1870**	1.5270	1.7190
	Sum	**5.7279**	6.2609	7.0581

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

Table 3 The effect of the various crossover types on the performance of the genetic algorithm

λ	Problem size	PMX	OPX
0	10×5	**0**[a]	**0**
	30×10	2.2109	**1.9234**
	50×20	**0.3776**	0.3817
	Sum	2.5885	**2.3051**
0.05	10×5	1.3214[b]	**1.0605**
	30×10	4.1920	**3.9740**
	50×20	1.7740	**1.7310**
	Sum	7.2874	**6.7655**
0.1	10×5	1.4890	**1.1600**
	30×10	3.3900	**3.1950**
	50×20	1.7580	**1.7540**
	Sum	6.6370	**6.1090**
0.5	10×5	1.6726	**1.3186**
	30×10	3.5200	**3.3170**
	50×20	1.5960	**1.5650**
	Sum	6.7886	**6.2006**
1.0	10×5	1.6434	**1.3418**
	30×10	3.4740	**3.2830**
	50×20	1.4790	**1.4760**
	Sum	6.5964	**6.1008**

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

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, 0.05, 0.1, 0.5, and 1}. Table 2 through 4 present the effect of the population size, crossover types and mutation types by using the average (absolute resp. relative) deviation from the best value as the performance measure.

From the full factorial experiment, we analyzed our results by means of a multi-factor *Analysis of Variance* (*ANOVA*) technique. We have found that for population sizes, crossover types, and mutation types there were statistically significant differences. In general, a small population size (30) is superior. The OPX crossover is clearly superior to the PMX crossover. Since there were some interactions between crossover types and mutation types for the problem size 30 jobs × 10 stages, if we select OPX as the crossover type, pairwise interchange moves are better than shift moves for $\lambda = 0$. Consequently, the mutation operator should be based on pairwise interchanges for $\lambda = 0$ and on shifts of jobs otherwise. For ANOVA and Tukey's test at a significance level $\alpha = 0.05$ in the crossover and mutation rates, we have found that no particular mutation rate (we fixed 0.5) and crossover rate (we fixed 0.8) are superior to the others.

Table 4 The effect of the various mutation types on the performance of the genetic algorithm

λ	Problem size	PI	SM
0	10×5	**0**[a]	**0**
	30×10	2.2016	**1.9326**
	50×20	**0.3632**	0.3961
	Sum	2.5648	**2.3287**
0.05	10×5	1.3514[b]	**1.0305**
	30×10	4.4330	**3.7330**
	50×20	**1.7510**	1.7550
	Sum	7.5354	**6.5185**
0.1	10×5	1.5430	**1.1060**
	30×10	3.5940	**2.9910**
	50×20	1.7700	**1.7430**
	Sum	6.9070	**5.840**
0.5	10×5	1.7074	**1.2838**
	30×10	3.6590	**3.1780**
	50×20	**1.5650**	1.5960
	Sum	6.9314	**6.0578**
1.0	10×5	1.6841	**1.3011**
	30×10	3.4740	**3.2830**
	50×20	1.4920	**1.4630**
	Sum	6.6501	**6.0471**

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

Finally, we used the recommended GA parameters to test the choice of an appropriate initial population. The letters before letters GA denote the heuristic rule as one initial solution for GA. For example, SPTGA means that the SPT rule is used as one initial solution for GA, or RNDGA means that the initial population in GA is completely randomly generated. In addition, we use some selected algorithms in parallel as a part of the initial population. Based on each heuristic group, we use all solutions in each heuristic group stated above as a part of the initial population. Consequently, we have four new choices of initial populations tested (denoted as MIX1GA, MIX2GA, MIX3GA, and MIX4GA, respectively).

Table 5 Comparisons of the genetic algorithm with different initial populations

λ	Problem size	RNDGA	SPTGA	LPTGA	ERDGA	EDDGA	MSTGA	S/PGA	HSEGA	ISPTGA	ILPTGA	IERDGA	IEDDGA	IMSTGA	IS/PGA	IHSEGA
0	10×5	**0**	0.04	0.02	**0**	**0**	0.04	**0**	0.02	0.04	**0**	0.04	0.02	**0**	**0**	0.08
	30×10	1.26	1.14	1.16	1.26	1.04	1.10	1.26	1.24	1.18	1.38	1.34	1.10	**1.02**	1.14	1.18
	50×20	0.84	0.92	0.86	0.92	0.92	0.76	0.86	1.04	0.86	0.98	1.04	0.64	0.82	0.88	0.96
	Sum	2.10	2.10	2.04	2.18	1.96	1.90	2.12	2.30	2.08	2.36	2.42	**1.76**	1.84	2.02	2.22
0.05	10×5	1.080	1.122	1.091	0.915	1.007	1.007	1.049	1.196	1.328	0.877	1.018	1.220	1.568	1.018	1.162
	30×10	3.518	3.780	4.005	4.100	3.453	3.139	3.656	3.396	4.179	4.262	4.746	3.485	3.725	3.456	4.408
	50×20	1.400	2.211	1.735	1.453	1.971	1.709	1.549	1.575	2.226	1.594	1.808	1.293	1.896	1.703	2.523
	Sum	5.998	7.113	6.831	6.468	6.431	5.855	6.254	6.167	7.733	6.733	7.572	5.998	7.189	6.177	8.093
0.1	10×5	0.935	1.129	1.153	1.399	1.407	1.237	1.156	1.035	1.101	0.995	1.163	1.114	0.916	1.120	0.866
	30×10	2.666	2.624	2.956	3.107	2.748	2.655	2.862	2.645	3.255	3.826	3.166	2.915	3.437	2.917	3.796
	50×20	1.675	1.952	1.824	1.570	1.793	1.800	1.593	2.042	2.139	2.879	1.889	2.044	1.842	1.632	2.251
	Sum	5.276	5.705	5.933	6.076	5.948	5.692	5.611	5.722	6.495	7.700	6.218	6.073	6.195	5.669	6.913
0.5	10×5	1.070	1.088	1.066	0.956	0.987	0.980	1.040	1.137	0.892	0.896	**0.591**	0.977	0.847	1.307	1.287
	30×10	2.352	2.512	2.723	2.646	2.715	2.896	2.505	2.836	3.112	2.532	3.420	2.336	3.000	2.224	3.315
	50×20	1.651	1.786	1.880	1.719	1.841	1.781	1.540	1.521	2.130	1.674	1.864	1.944	1.756	2.019	2.171
	Sum	5.073	5.386	5.669	5.321	5.543	5.657	5.085	5.494	6.134	5.102	5.875	5.257	5.603	5.550	6.773
1.0	10×5	0.936	1.298	0.764	1.007	0.786	0.929	0.781	1.298	0.840	0.773	0.741	1.198	0.929	1.290	1.519
	30×10	2.253	2.241	2.118	2.495	2.040	2.618	2.420	2.241	3.045	2.786	3.203	2.306	3.493	2.202	3.770
	50×20	1.472	1.635	1.672	1.481	1.650	1.450	1.664	1.635	1.912	2.000	1.806	1.526	1.930	2.032	2.130
	Sum	4.660	5.174	4.554	4.983	4.476	4.997	4.865	5.174	5.797	5.558	5.750	5.029	6.352	5.524	7.419

λ	Problem size	PALGA	CDSGA	GUPGA	DANGA	NEHGA	IPALGA	ICDSGA	IGUPGA	IDANGA	INEHGA	MIX1GA	MIX2A	MIX3A	MIX4GA
0	10×5	0.02[a]	0.02	0.02	0.02	**0**	0.02	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	30×10	1.44	1.20	1.34	1.24	1.56	1.12	1.42	1.30	1.32	1.56	1.10	1.38	1.28	1.50
	50×20	0.72	0.92	0.88	0.76	1.04	1.16	1.10	1.06	0.86	1.04	0.78	1.02	**0.50**	1.04
	Sum	2.18	2.14	2.24	2.02	2.60	2.30	2.52	2.36	2.18	2.60	1.88	2.40	1.78	2.54
0.05	10×5	1.361[b]	1.150	0.846	1.311	1.172	1.108	1.359	1.183	1.518	1.172	1.202	1.124	1.178	**0.791**
	30×10	3.404	3.786	3.611	3.853	2.496	3.901	3.693	3.954	4.972	2.496	4.031	2.559	3.068	**2.394**
	50×20	1.898	1.729	1.546	1.590	**0.403**	2.229	2.625	1.403	2.501	**0.403**	1.471	0.410	1.063	0.441
	Sum	6.663	6.665	6.003	6.754	4.071	7.238	7.677	6.540	8.991	4.071	6.704	4.093	5.309	**3.626**
0.1	10×5	1.196	1.143	1.017	1.385	1.211	1.206	1.040	1.137	1.210	1.211	1.226	1.132	**0.792**	0.920
	30×10	2.880	2.503	2.938	2.719	1.499	3.419	3.507	3.110	3.199	1.499	2.793	1.479	2.480	**1.466**
	50×20	1.901	1.654	1.810	1.765	**0.370**	1.829	3.271	1.976	2.394	**0.370**	1.559	0.444	1.516	0.422
	Sum	5.977	5.300	5.765	5.869	3.080	6.454	7.818	6.223	6.803	3.080	5.578	3.055	4.788	**2.808**
0.5	10×5	1.208	0.929	0.871	1.278	1.044	0.895	0.718	0.903	0.938	1.044	1.054	0.972	0.852	0.832
	30×10	2.101	2.540	2.364	2.393	1.542	2.209	2.323	3.064	2.645	1.542	2.130	**1.367**	3.533	1.405
	50×20	1.749	1.661	1.557	1.643	0.351	1.967	2.977	2.332	2.069	0.351	1.606	0.358	1.221	**0.330**
	Sum	5.058	5.130	4.792	5.314	2.937	5.071	6.018	6.299	5.652	2.937	4.790	2.697	5.606	**2.567**
1.0	10×5	0.856	0.788	0.645	1.033	1.009	1.016	0.845	**0.607**	0.753	1.009	0.751	0.920	0.712	0.781
	30×10	2.329	2.247	2.641	2.218	1.340	3.286	2.889	2.059	2.829	1.340	2.788	**1.314**	2.508	1.437
	50×20	1.700	1.447	1.653	1.505	0.351	2.433	2.483	2.279	2.023	0.351	1.890	0.339	1.435	**0.330**
	Sum	4.885	4.481	4.939	4.756	2.700	6.735	6.217	4.944	5.604	2.700	5.429	2.573	4.655	**2.547**

[a] average absolute deviation for $\lambda = 0$

[b] average percentage deviation for $\lambda > 0$

From the results in Table 5, we have found that IEDDGA IMSTGA, MIX1GA, and MIX3GA rules are good algorithms for problems with $\lambda = 0$ (notice that they are more oriented to the minimization of tardiness-based criteria), and they are slightly statistically significantly different from NEHGA (or INEHGA, which yields the same results), MIX2GA and MIX4GA. However, for NEHGA, INEHGA, MIX2GA and MIX4GA there are no statistically significant differences for all problems, but they were largely statistically significantly different for the large problem sizes with $\lambda > 0$. Consequently NEHGA, INEHGA, MIX2GA and MIX4GA are good overall choices for GA with using biased initial solutions instead of random initial solutions.

9 Conclusions

In this paper, we have investigated both constructive and iterative (GA-based) 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 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 NEH is an excellent constructive algorithm for minimizing the

objective function considered. In particular, the NEH algorithm is most superior to the other constructive algorithms regardless polynomial improvement heuristics.

In addition, we use GA-based algorithms as improving algorithms. Before we studied the influence of the initial population on the performance of GA, we tested the GA parameters. We have found that OPX crossover is certainly superior to PMX, whereas we recommend that the PI move should be selected as the mutation operator for problems with $\lambda = 0$, and the shift move for the others with $\lambda > 0$. We have fixed the crossover and mutation rates at 0.8 and 0.5, respectively. For the recommended GA parameters, we investigated the selection of a starting population by using the constructive algorithms. The variants NEHGA, INEHGA, MIX2GA and MIX4GA can all be recommended in general.

Further research can be done to use other improving algorithms such as tabu search, simulated annealing, or 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. Baker KR (1974) Introduction to Sequencing and Scheduling. John Wiley & Sons, New York
2. Gupta JND, Krüger K, Lauff V, Werner F, Sotskov YN (2002) Heuristics for hybrid flow shops with controllable processing times. Comput Oper Res 29(10): 1417–1439
3. Alisantoso D, Khoo LP, Jiang PY (2003) An immune algorithm approach to the scheduling of a flexible PCB flow shop. Int J Adv Manuf Tech 22(11–12): 819–827
4. Lin HT, Liao CJ (2003) A case study in a two-stage hybrid flow shop with setup time and dedicated machines. Int J Prod Econ 86(2): 133–143
5. Wang W, Hunsucker JL (2003) An evaluation of the CDS heuristic in flow shops with multiple processor. J Chinese Inst Ind Eng 20(3): 295–304
6. Linn R, Zhang W (1999) Hybrid flow shop scheduling: A survey Comput Ind Eng 37(1–2): 57–61
7. Wang W (2005) Flexible flow shop scheduling: Optimum, heuristics, and artificial intelligence solutions," Expert Systems 22(2): 78–85
8. Arthanari TS, Ramamurthy KG (1971) An extension of two machines sequencing problem. Opsearch 8(1): 10–22
9. Salvador MS (1973) A solution to a special case of flow shop scheduling problems. in: Elmaghraby SE (ed.), Symposium of the Theory of Scheduling and Applications. Springer, New York: 83–91.
10. Brah SA, Hunsucker JL (1991) Branch and bound algorithm for the flow shop with multiple processors. Eur J Oper Res 51(1): 88–99.
11. Portmann MC, Vignier A, Dardilhac D, Dezalay D (1998) Branch and bound crossed with GA to solve hybrid flowshops. Eur J Oper Res 107(2):389–400
12. Moursli O, Pochet Y (2000) Branch and bound algorithm for the hybrid flowshop. Int J Prod Econ 64(1–3): 113–125
13. Gupta JND (1988) Two-stage, hybrid flow shop scheduling problem. J Oper Res Soc 39(4):. 359–364
14. Sriskandarajah C, Sethi SP (1989) Scheduling algorithms for flexible flowshops: worst case and average case performance. Eur J Oper Res 43(2): 143–160
15. Guinet A, Solomon MM, Kedia PK, Dussauchoy A (1996) A computational study of heuristics for two-stage flexible flowshops. Int J Prod Res 34(5): 1399–1415
16. Gupta JND, Tunc EA (1994) Scheduling a two-stage hybrid flowshop with separable setup and removal times. Eur J Oper Res 77(3): 415–428
17. Santos DL, Hunsucker JL, Deal DE (1996) An evaluation of sequencing heuristics in flow shops with multiple processors. Comput Ind Eng 30(4): 681-691
18. Jones DF, Mirrazavi SK, Tamiz M (2002) Multi-objective meta-heuristics: An overview of the current state-of-art. Eur J Oper Res 137(1): 1-9
19. Gen M, Cheng R (1997) Genetic algorithms and engineering design. Willy, New York
20. Gen M, Cheng R (2000) Genetic algorithm & engineering optimization. Willy, New York
21. Reeves CR (1995) A genetic algorithm for flowshop sequencing. Comput Oper Res 22(1): 5-13
22. Cheng R, Gen M, Tozawa T (1995) Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. Comput Ind Eng 29(1–4): 513–517
23. Ruiz R, Maroto C, Alcaraz J (2005) Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. Eur J Oper Res 165(1): 34–54
24. Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY (2006) A survey of scheduling problems with setup times or cost. Eur J Oper Res (to appear).
25. Palmer DS (1965) sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum.Operat Res Quart 16(1): 101–107
26. Campbell HG, Dudek RA, and Smith ML (1970) A Heuristic algorithm for the n-Job m-Machine sequencing problem. Manag Sci 16(10): 630–637
27. Gupta JND (1971) A functional heuristic algorithm for the flow-shop scheduling problem. Operat Res Quart 22(1): 39-47
28. Dannenbring DG (1977) An evaluation of flow shop sequencing heuristics. Manag Sci 23(11): 1174-1182
29. Nawaz M, Enscore Jr. E, Ham I (1983) A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. OMEGA Int J Manag Sci 11(1): 91–95
30. Pinedo M, Chao X (1999) Operations scheduling with applications in manufacturing and services. Irwin/McGraw-Hill, New York
31. Holland JA (1975) Adaptation in natural and artificial systems. Annn Arbor. University of Michigan
32. Werner F (1984) On the solution of special sequencing problems. PhD Thesis, TU Magdeburg.
33. Rajendran C, Ziegler H (2003) Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. Eur J Oper Res 149(3): 513–522