Constructive and Simulated Annealing Heuristics for Hybrid Flow Shops with Unrelated Parallel Machines

Jitti Jungwattanakit, Manop Reodecha, Paveena Chaovalitwongse
Department of Industrial Engineering, Faculty of Engineering,
Chulalongkorn University, Bangkok 10330 Thailand. jitti.j@student.chula.ac.th

Frank Werner
Faculty of Mathematics, Otto-von-Guericke-University, P.O. Box 4120,
D-39016 Magdeburg, Germany. Frank.Werner@Mathematik.Uni-Magdeburg.DE

Abstract

Most scheduling problems are combinatorial optimization problems which are too difficult to be solved optimally, and hence heuristics are used to obtain good solutions in reasonable times. The specific goal of this paper is to investigate scheduling heuristics to seek the minimum of a positively weighted convex sum of makespan and the number of tardy jobs in a static hybrid flow shop environment, where at least one production stage is made up of unrelated parallel machines. In addition, sequence - and machine - dependent setup times are considered. Some simple dispatching rules and flow shop makespan heuristics are adapted for the sequencing problem under consideration. The improvement heuristic algorithm proposed is a reinsertion algorithm. A simulated annealing algorithm is presented in this paper. Three basic parameters (i.e., cooling schedules, neighborhood structures, and initial temperatures) of a simulated annealing algorithm 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.

Keywords: Hybrid flow shop scheduling; Constructive algorithms; Improvement heuristics; Simulated Annealing algorithms.

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 to the complex nature thereof.

This paper is primarily concerned with industrial scheduling problems, where one first has to assign limited resources to jobs 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, 1996), an automobile assembly plant (Agnetis *et al.*, 1997), a printed circuit board manufacture (Alisantoso, Khoo, and Jiang, 2003, and Hsieh, Chang, and Hsu, 2003), 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. The mixed character of a production system, which lies between flow shop and parallel machines, is known as a hybrid or flexible 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 hybrid flow shop model, a generalization of a classical flow shop model, is more realistic, and it assumes that at least one stage must have multiple machines. 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 jobs to machines at each stage and sequencing the jobs assigned to the same machine so that some optimality criteria are minimized.

Although the hybrid flow shop problem has been widely studied in the literature, most of the studies related to hybrid flow shop problems are concentrated on problems with identical processors, see for instance, Gupta, Krüger, Lauff, Werner and Sotskov (2002), Alisantoso, Khoo, and Jiang (2003), Lin and Liao (2003) and Wang and Hunsucker (2003). 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 hybrid 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 hybrid flow shop problem is given in Linn and Zhang (1999) and Wang (2005). Most of the earlier literature has considered the simple case of only two stages. Arthanari and Ramamurthy (1971) and Salvador (1973) are among the first who define the hybrid 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 hybrid flow shop problem are proposed by many authors, e.g. branch and bound algorithms are given in Brah and Hunsucker (1991) and Moursli and Pochet (2000).

When an exact algorithm is applied to large hybrid 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 (1988) proposes heuristic techniques for a simplified hybrid flow shop makespan problem with two stages and only one machine at stage two. The proposed heuristics are based on extensions of Johnson's algorithm. Sriskandarajah and Sethi (1989) develop simple heuristic algorithms for the two-stage hybrid 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 (1996) propose a heuristic for the makespan problem in a two-stage hybrid 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 (1994) consider the two-stage hybrid 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 one. Santos, Hunsucker, and Deal (1996) investigate scheduling procedures which seek to minimize the makespan in the 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. For example, Nowicki and Smutnicki (1998) propose a Tabu Search (TS) algorithm for the hybrid flow shop makespan problem. Gourgand, Grangeon, and Norre (1999) present several Simulated Annealing (SA)-based algorithms for the hybrid flow shop problem. A specific neighborhood is used and the authors apply the methods to a realistic industrial problem. Jin, Yang, and Ito (2006) consider a hybrid 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. We have found 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 (1994).

In this paper, a hybrid flow shop problem with unrelated parallel machines and setup times is studied. The goal of the problem is to seek a schedule which minimizes a positively weighted convex sum of makespan and the number of tardy jobs. Due to the complex of this problem, the constructive heuristics based on dispatching rules and pure flow shop makespan heuristics are adapted and simulated annealing (SA)-based algorithms as the iterative algorithms are proposed.

The rest of the paper is organized as follows: The problem considered in this paper is described in Section 2. Heuristic algorithms are sketched in Section 3. Section 4 and Section 5 present the variants of the simulated annealing algorithms. Computational results with the heuristics are briefly discussed in Section 6 and conclusions are in Section 7.

2. Problem Statement

The hybrid flow shop system is defined by the 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 a set $M = \{M^1,\ldots, M^k\}$. 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^t_{ij} of job j on machine i at stage t is equal to ps^t_j / v^t_{ij} , where ps^t_j is the standard processing time of job *j* at stage *t*, and v^t_{ij} is the relative speed of job *j* which is processed by the machine *i* at stage *t*.

There are processing restrictions of jobs as follows: (1) jobs are processed without preemptions on any machine; (2) a job cannot be processed before its completion of the previous operation; (3) every machine can process only one operation at a time; (4) operations have to be realized sequentially, without overlapping between stages; (5) job splitting is not permitted.

Setup times considered in this problem are classified into two types, namely machine-dependent setup time and 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 on the machine. ch^t_{ij} denotes the length of 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^t_{lj} 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.

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

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

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. Heuristic Algorithms

Heuristic algorithms have been developed to provide good and quick solutions. They obtain solutions to large problems with acceptable computational times. They are simple and have no mathematical proof, see Brah and Loo (1999), and Kurz and Askin (2003). They can be divided into either constructive or improvement algorithms. The former algorithms build a feasible solution from scratch. The latter algorithms try to improve a previously generated solution by normally using some form of specific problem knowledge. However, the time required for computation is usually larger compared to the constructive algorithms. The drawback of heuristic algorithms is that they do not generate optimality and it may be difficult to judge their effectiveness (Youssef, Sait, and Adiche, 2001).

3.1 Heuristic Construction of a Schedule

Since the hybrid 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 hybrid 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 (1996).

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 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.

3.2 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, while other are used mainly for comparison purposes.

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, 1974, and Pinedo and Chao, 1999).

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.

We remind that the dispatching rules related to the processing time calculations will generate the nine solutions from the nine combinations of the nine different relative speeds and setup times. The best solution is selected from them.

Palmer's heuristic (1965) 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 (1970) 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 (1971) provides an algorithm denoted by GUP, in a similar manner as algorithm PAL by using a different slope index and schedules the jobs according to the slope order.

Dannenbring (1977) denoted by DAN 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. Its purpose is to provide good and quick solutions.

Nawaz, Enscore and Ham (1983) develop the 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 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 algorithms to the hybrid 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.

3.3 Improvement Heuristics

Unlike constructive algorithms, improvement heuristics start with an already built schedule and attempt to improve it by some given procedure. Their use is necessary since the constructive algorithms (especially some algorithms that are adapted from pure makespan heuristics and some dispatching rules such as SPT, LPT rules) without due date considerations. In this section, we will improve the overall function value by concerning mainly the due date criterion.

In order to find a satisfactory solution of our due date problem, we use a polynomial heuristic by applying the shift neighborhood as an improvement mechanism based on the idea that we will consider the jobs that are tardy and move them left and right in all positions. The best schedule among the $(n-1)^2$ generated neighbors (if all jobs under consideration are late, i.e. at most $O(n^2)$ job sequences are examined by the improvement heuristics) is then taken as the result.

4. Simulated Annealing Heuristic

A simulated annealing (SA) heuristic has been introduced by Kirkpatrick, Gelatt, and Vecchi (1984). It constitutes a class of approximate (heuristic) algorithms. It is an enhanced version of local optimization or 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. Annealing refers to the process which occurs when a physical substance, such as metal, is heated until it melts and then gradually cooled (according to an annealing schedule) until the solid reaches the lowest energy or the ground state. Due to natural variability, however, there is some probability at each stage of the cooling process that a transition to a higher energy state will occur. As the energy state naturally declines, the probability of moving to a higher energy state decreases. However, if the initial temperature is not high enough or if the temperature is decreased rapidly, the solid at the ground state will have many defects or imperfections.

Before the development of metaheuristics such as simulated annealing, tabu search, and genetic algorithms, many local search techniques are used to solve large combinatorial optimization problems. These heuristics start with an initial solution and randomly generate a neighborhood solution. The cost of the generated neighborhood solution will be compared with the cost of the initial solution. If the cost of the new solution is better than the cost of the initial solution, this solution becomes the best solution and it is a starting solution in the next generation. Otherwise, the initial solution is still the starting solution in the next iteration. However, the procedure often converges to a poor local optimum. To overcome this drawback, the non-improving move technique is proposed to avoid being trapped in a poor local optimum. Such an idea is behind SA and other metaheuristics.

A basic SA algorithm starts with generating an initial solution S_0 as a current solution S_{cur} and setting the SA parameters such as initial temperature, cooling schedule, acceptance probability, and stopping criteria. Then, at each iteration a neighbor solution $S' \in N(S_{cur})$ is generated. If $f(S') < f(S_{cur})$, the new solution S' is accepted as a current solution (i.e., set $S_{cur} = S'$), otherwise the acceptance probability is considered. The acceptance probability is the probability of accepting non-improving moves and is given by $\exp(-\Delta f/T)$, where Δf is the change in the cost function (i.e. the cost of neighbor solution minus the cost of the current solution, $\Delta f = f(S') - f(S_{cur})$), and T is the temperature control parameter. If RN is a randomly generated number between 0 and 1, and $RN < \exp(-\Delta f/T)$, then accept the non-improving solution S' as the current solution S_{cur} (i.e., set $S_{cur} = S'$). Otherwise, reject the non-improving solution, and keep the current solution. The acceptance probability is initially high, but as the search proceeds (and the temperature decreases), it will reduce as well. If $f(S') < f(S_{best})$, set $S_{best} = S_{cur}$. A neighborhood structure has to be defined from which a neighbor of the current solution is generated. Before reducing the temperature by using the cooling schedule, NT is the preset parameter which establishes the number of total allowed times for the annealing process as each temperature reduction, called the epoch length. The cooling schedule is another parameter for the SA heuristic, which is used to reduce the temperature during the execution of the algorithm.

4.1 The cooling schedule

The cooling schedule governs how likely the algorithm is to accept a bad transition as a function of the temperature T at each iteration. At the beginning of the search, the algorithm is eager to use randomness to explore the search space widely, so the probability of accepting a negative transition is high by using a higher temperature. As the search progresses, the temperature is decreased, thus the probability of accepting will gradually decrease, converging to a simple iterative improvement algorithm.

There are two most widely used cooling schedules: (1) the geometric reduction schedule using the function $T_{new} = \alpha \times T_{old}$; and (2) the schedule suggested by Lundy and Mees (1986) using the relation $T_{new} = T_{old}/(1+ \beta T_{old})$.

The scheme that follows a geometric law, which is one of the most often used, corresponds to an exponential decay of the temperature. The schedule suggested by Lundy and Mees (1986) provides a fast cooling in the early iterations and slower cooling at later iterations. Consequently, at the beginning the search will explore the search space, while at the end the search will exploit to the local minimum.

4.2 Neighborhoods

A key component of any local search algorithm is the move operator or neighborhood structure. This paper considers two alternative neighborhoods: (1) a pairwise interchange (PI) neighborhood, and (2) a shift move (SM) neighborhood.

The idea for a PI neighborhood is to exchange a pair of jobs, π_r and π_i, where $1 \le i \le n$ and $i \ne r$. Such an operation swaps the job 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 the selection of a neighbor, one of all possible $\frac{n}{2}(n-1)$ PI neighbors are checked is then compared to the starting one S_{cur}.

An SM neighborhood is to reposition some jobs. A 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 —$\pi' = (\pi_1,\ldots, \pi_{r-1}, \pi_{r+1}, \ldots, \pi_i, \pi_r,\ldots, \pi_n)$. If $1 \le i < r \le n$, it is called a left shift—$\pi' = (\pi_1,\ldots \pi_r, \pi_i,\ldots,\pi_{r-1}, \pi_{r+1}, \ldots, \pi_n)$. The SM neighborhood has $(n-1)^2$ neighbors.

5. Choice of an initial solution

A SA algorithm has been shown to be effective for many combinatorial optimization problems (see Koulamas, Antony, and Jaen, R., 1994), and it seems easy 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 solution by using particular constructive algorithms. We used as an 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 improvement heuristics, respectively.

6. Computational results

Firstly, we studied the constructive algorithms that are separated into four main groups. The first heuristic group is the simple dispatching rules such as SPT, LPT, ERD, EDD, MST, S/P, and HSE. The second heuristic group is 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 heuristics in which they are improved by using an all-shift-move algorithm, and they are denoted by the letter "I" before the letters denoted by the first two heuristics. 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 $\lambda \in \{0, 0.05, 0.1, 0.5, \text{and } 1\}$ 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	S/P	HSE	PAL	CDS	GUP	DAN	NEH
0	10×5	2.3[a]	1.7	2.8	3.1	3.2	3.0	2.4	1.9	1.7	1.8	2.0	**0.5**
	30×10	8.0	8.9	8.3	12.4	12.3	12.2	7.9	8.0	6.4	7.8	7.7	**2.4**
	50×20	7.4	8.6	7.7	16.2	16.2	14.3	7.1	9.7	7.3	7.9	9.3	**2.3**
	Sum	17.7	19.2	18.8	31.7	31.7	29.5	17.4	19.6	15.4	17.5	19.0	**5.2**
0.05	10×5	18.82[b]	12.81	24.23	24.09	22.21	22.10	18.83	11.66	10.03	14.12	11.47	**2.52**
	30×10	17.78	14.72	19.61	20.91	18.21	17.61	16.51	16.80	12.35	14.71	14.77	**0.59**
	50×20	8.53	8.28	10.14	11.75	10.96	9.87	8.94	7.90	6.99	8.03	8.43	**0.30**
	Sum	45.14	35.81	53.98	56.75	51.37	49.58	44.28	36.36	29.36	36.86	34.67	**3.41**
0.1	10×5	17.90	11.82	22.91	22.67	20.21	20.52	17.84	10.71	8.79	13.05	10.32	2.86
	30×10	16.61	13.12	18.46	19.14	16.38	15.71	15.32	15.59	11.17	13.30	13.61	**0.40**
	50×20	8.13	7.75	9.72	10.47	9.65	8.77	8.49	7.27	6.45	7.57	7.79	**0.08**
	Sum	42.64	32.69	51.08	52.28	46.25	45.00	41.65	33.57	26.40	33.92	31.72	**3.33**

Table 1 Average performance of constructive algorithms

λ	Problem size	SPT	LPT	ERD	EDD	MST	S/P	HSE	PAL	CDS	GUP	DAN	NEH
0.5	10×5	17.48	11.21	22.17	21.94	18.81	19.21	17.42	10.33	7.87	12.44	9.74	2.98
	30×10	16.12	12.29	18.01	18.13	15.30	14.46	14.70	15.03	10.50	12.60	13.04	**0.26**
	50×20	8.11	7.59	9.68	9.71	8.84	8.12	8.38	7.04	6.28	7.50	7.56	**0.09**
	Sum	41.71	31.09	49.86	49.78	42.94	41.79	40.50	32.41	24.65	32.55	30.34	**3.33**
1.0	10×5	17.48	11.21	22.17	21.94	18.81	19.21	17.42	10.33	7.87	12.44	9.74	2.98
	30×10	16.34	12.46	18.24	18.29	15.43	14.57	16.34	15.24	10.68	12.77	13.24	**0.38**
	50×20	8.12	7.58	9.69	9.63	8.75	8.05	8.12	7.03	6.27	7.50	7.54	**0.08**
	Sum	41.94	31.24	50.10	49.85	42.98	41.82	41.88	32.59	24.82	32.72	30.52	**3.44**
0	10×5	1.0	0.7	1.4	0.7	1.0	0.9	0.7	0.6	**0.5**	0.9	1.0	**0.5**
	30×10	4.2	4.5	4.3	3.2	**2.4**	5.7	4.3	4.4	4.0	4.1	4.4	**2.4**
	50×20	3.6	4.3	3.5	3.5	3.9	7.8	4.1	4.6	4.5	4.8	5.3	**2.3**
	Sum	8.8	9.5	9.2	7.4	7.3	14.4	9.1	9.6	9.0	9.8	10.7	**5.2**
0.05	10×5	5.06	3.60	3.82	6.84	5.32	5.43	5.97	2.64	3.39	3.63	3.46	**2.52**
	30×10	6.03	6.66	7.75	8.00	11.15	8.29	8.96	7.07	4.78	6.84	6.24	**0.59**
	50×20	4.46	5.22	5.10	6.03	5.66	6.28	3.97	4.22	3.53	4.83	5.38	**0.30**
	Sum	15.55	15.49	16.67	20.88	22.14	19.99	18.90	13.93	11.70	15.30	15.08	**3.41**
0.1	10×5	4.63	3.95	4.60	6.40	5.78	4.48	4.50	**1.80**	2.01	3.20	1.94	2.86
	30×10	6.10	6.14	8.39	7.94	9.93	7.83	6.43	6.23	2.78	5.57	5.51	**0.40**
	50×20	4.27	5.22	4.78	5.56	5.66	4.61	3.73	3.78	3.15	4.88	4.73	**0.08**
	Sum	14.99	15.31	17.78	19.89	21.36	16.93	14.66	11.81	7.93	13.66	12.18	**3.33**
0.5	10×5	4.31	2.84	4.91	6.01	5.80	3.98	5.38	1.71	2.24	2.55	**0.80**	2.98
	30×10	6.13	6.10	8.94	7.85	9.18	7.20	5.92	5.19	1.65	4.88	5.95	**0.26**
	50×20	4.37	4.72	4.81	5.28	5.44	4.73	3.73	4.06	2.92	4.79	4.61	**0.09**
	Sum	14.81	13.66	18.66	19.14	20.42	15.91	15.03	10.96	6.82	12.22	11.36	**3.33**
1.0	10×5	4.31	2.84	4.91	6.01	5.80	3.98	5.38	1.71	2.24	2.55	**0.80**	2.98
	30×10	6.33	6.18	9.17	7.72	9.33	7.40	6.33	5.29	1.69	4.98	5.95	**0.38**
	50×20	4.39	5.01	4.91	5.17	5.45	4.09	4.39	4.05	2.93	4.77	4.59	**0.08**
	Sum	15.04	14.02	18.98	18.91	20.58	15.47	16.10	11.05	6.86	12.31	11.34	**3.44**

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

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

From these results it is obvious that the constructive algorithms in the fourth heuristic group improved the pure 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), the HSE rule outperforms the other dispatching rules for $\lambda = 0$, and the LPT rule is better than the other rules for $\lambda > 0$. Among the adapted flow shop makespan heuristics in the heuristic Group II, the NEH algorithm is clearly the best algorithm among all studied constructive heuristics. The CDS algorithm is certainly the algorithm on the second rank whereas the remaining algorithms are slightly different from each other.

When we apply the insertion algorithm (denoted as the letter "I" first) to the dispatching rules and adapted makespan heuristics, we have found that the quality of solution can be improved by about 50 percent except for the NEH rule. It is noted that the NEH rule is not improved by using the improvement heuristics in algorithm INEH because the NEH algorithm is embedded by such an (re-)insertion algorithm itself. However, the improvement of the heuristics from the adapted pure makespan heuristics in heuristic Group IV is better than the improvement of the heuristics derived from the dispatching rules in the heuristic Group III.

Secondly, we studied the SA algorithm with a random initial solution. The purpose of this study is to determine the favorable SA parameters, i.e., initial temperatures, neighborhood structures, and cooling schedules**.** Given the above three different problem sizes, the following SA parameter values were used in this test.

Initial temperatures	:	100 through 1000, in steps of 100
Neighborhood structures	:	PI, SM

Cooling schedules	: CS1 – CS3 refer to geometric reduction schedule at α {0.85, 0.90, and 0.95}, and CS4 – CS6 are the schedules by Lundy and Mees at β {0.0005, 0.001, and 0.002}

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 $\lambda \in \{0, 0.05, 0.1, 0.5, \text{and } 1\}$. Table 2 through Table 4 present the effect of the initial temperatures, neighborhood structures and cooling schedules 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 using a 5% significant level. We have found that for neighborhood structures and cooling schedules, there are statistically significant differences, whereas there are not statistically significant differences in the initial temperatures. A low initial temperature is however slightly preferable (we recommend 100). It was clear that pairwise interchange moves were better than shift moves for $\lambda = 0$, whereas the shift moves were better than pairwise interchange moves for the other values. Consequently, the neighborhood structures should be based on pairwise interchanges for $\lambda = 0$ and on shifts of jobs otherwise. For cooling schedules, we observed that the geometric cooling scheme outperforms the other cooling schedule. In particular, we recommend the reduction scheme $T_{new}=0.85\times T_{old}$, where T_{new} and T_{old} denote the new and old temperatures.

Finally, we used the recommended SA parameters to test the choice of an initial solution. The letters before SA denote the heuristic rule as an initial solution for the SA algorithm. For example, SPTSA means that the SPT rule is used as an initial solution for the SA algorithm.

From these results in Table 5, we have found that there are no statistically significant differences in different initial solutions. We have however found that the IEDDSA rule is a good algorithm for problems with $\lambda= 0$, and the NEHSA, and INEHSA rules are slightly better than the others with $\lambda > 0$. Consequently, in general the NEHSA and INEHSA algorithms are good choices for the SA algorithm with using a biased initial solution.

Table 2 The effect of the various initial temperatures on the performance of the SA algorithm

λ	Problem size	100	200	300	400	500	600	700	800	900	1000
0	10×5	0.019[a]	0.022	0.019	**0.011**	0.025	0.019	0.019	0.017	0.022	0.025
	30×10	**2.747**	2.781	2.767	2.781	2.792	2.822	2.811	2.839	2.864	2.883
	50×20	**2.461**	2.531	2.561	2.561	2.539	2.603	2.628	2.608	2.653	2.681
	Sum	**5.227**	5.334	5.347	5.353	5.356	5.444	5.458	5.464	5.539	5.589
0.05	10×5	**1.954**[b]	2.140	2.171	2.079	2.010	2.195	2.195	2.192	2.261	2.251
	30×10	**7.662**	7.727	7.979	7.816	7.880	7.809	7.925	7.877	7.770	7.838
	50×20	**3.901**	4.010	4.100	4.161	4.117	4.145	4.151	4.232	4.197	4.277
	Sum	**13.517**	13.877	14.250	14.056	14.007	14.149	14.271	14.301	14.228	14.366
0.1	10×5	1.707	**1.647**	1.840	1.917	1.922	1.864	1.839	1.969	1.895	1.893
	30×10	**6.126**	6.137	6.218	6.237	6.218	6.254	6.304	6.291	6.386	6.361
	50×20	**3.440**	3.446	3.535	3.596	3.608	3.652	3.626	3.658	3.750	3.675
	Sum	11.273	**11.230**	11.593	11.750	11.748	11.770	11.769	11.918	12.031	11.929
0.5	10×5	**0.850**	0.884	0.873	0.947	0.962	0.959	0.968	1.048	1.025	1.030
	30×10	**3.723**	3.781	3.814	3.898	3.931	3.909	3.947	3.926	3.915	3.974
	50×20	**2.125**	2.240	2.285	2.312	2.360	2.404	2.381	2.377	2.414	2.338
	Sum	**6.698**	6.905	6.972	7.157	7.253	7.272	7.296	7.351	7.354	7.342
1.0	10×5	**0.513**	0.641	0.633	0.653	0.690	0.726	0.705	0.756	0.721	0.684
	30×10	**3.337**	3.392	3.470	3.452	3.504	3.497	3.520	3.554	3.546	3.534
	50×20	**1.761**	1.837	1.847	1.915	1.924	1.987	1.942	1.963	2.007	1.953
	Sum	**5.611**	5.870	5.950	6.020	6.118	6.210	6.167	6.273	6.274	6.171

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

Table 3 The effect of the various neighborhood structures on the performance of the SA algorithm

λ	Problem size	PI	SM
0	10×5	**0.016**[a]	0.024
	30×10	**2.794**	2.823
	50×20	**2.522**	2.643
	Sum	**5.332**	5.490
0.05	10×5	2.270[b]	**2.020**
	30×10	8.098	**7.522**
	50×20	4.192	**4.067**
	Sum	14.560	**13.609**
0.1	10×5	1.973	**1.725**
	30×10	6.522	**5.985**
	50×20	3.646	**3.551**
	Sum	12.141	**11.261**
0.5	10×5	1.136	**0.773**
	30×10	4.249	**3.515**
	50×20	2.425	**2.222**
	Sum	7.810	**6.510**
1.0	10×5	0.865	**0.479**
	30×10	3.897	**3.065**
	50×20	2.049	**1.778**
	Sum	6.811	**5.322**

Table 4 The effect of the various cooling schedules on the performance of the SA algorithm

λ	Problem size	CS1	CS2	CS3	CS4	CS5	CS6
0	10×5	**0.000**	0.002	0.028	0.033	0.022	0.035
	30×10	**0.653**	0.915	1.663	4.625	4.562	4.433
	50×20	**0.320**	0.638	1.880	4.237	4.283	4.137
	Sum	**0.973**	1.555	3.571	8.895	8.867	8.605
0.05	10×5	**0.936**	1.052	1.930	3.142	3.043	2.767
	30×10	**3.388**	3.443	4.614	12.314	12.034	11.068
	50×20	**1.059**	4.411	3.101	6.559	6.472	6.171
	Sum	**5.383**	8.906	9.645	22.015	21.549	20.006
0.1	10×5	**0.855**	0.959	1.561	2.770	2.696	2.255
	30×10	**2.741**	2.839	3.725	10.487	9.717	8.011
	50×20	**1.000**	1.295	2.620	6.008	5.758	4.911
	Sum	**4.596**	5.093	7.906	19.265	18.171	15.177
0.5	10×5	0.658	0.636	0.920	1.755	1.149	**0.610**
	30×10	**2.450**	2.560	2.940	6.833	4.816	3.693
	50×20	**0.959**	1.168	1.969	4.429	3.103	2.313
	Sum	**4.067**	4.364	5.829	13.017	9.068	6.616
1.0	10×5	0.590	0.546	0.782	1.122	0.628	**0.364**
	30×10	2.754	**2.700**	3.102	5.051	3.885	3.393
	50×20	**0.963**	1.157	1.827	3.167	2.400	1.968
	Sum	**4.306**	4.403	5.711	9.340	6.913	5.726

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

7. Conclusions

In this paper, we have investigated both constructive and iterative (SA-based) approaches for minimizing a convex combination of makespan and the number of tardy jobs for the hybrid flow shop problem with unrelated parallel machines and setup times, which is often occurring 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 and CDS algorithms outperform the others, respectively. In particular, the NEH algorithm is most superior to the other constructive algorithms regardless improvement heuristics. After we apply the improvement heuristics, the INEH algorithm based on the NEH rule is still better than others.

In addition, we use SA-based algorithms as improving algorithms. Before we studied the influence of the initial solution on the performance of the SA algorithm, 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 100). The neighborhood structures should be based on pairwise interchanges for $\lambda = 0$ and on shifts of jobs otherwise. The geometric cooling scheme $T_{new}=0.85\times T_{old}$ is recommended. For the recommended SA parameters, we investigated the selection of a starting solution by using several constructive algorithms. The variants NEHSA and INEHSA can both be recommended in general.

Further research can be done to use other improving algorithms such as tabu search, genetic algorithm, 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 simulated annealing as a local search algorithm within a Genetic Algorithm or the other algorithms.

Table 5 Comparisons of the SA algorithm with different initial solutions

λ	Problem size	SPTSA	LPTSA	ERDSA	EDDSA	MSTSA	S/PSA	HSESA	PALSA	CDSSA	GUPSA	DANSA	NEHSA
0	10×5	**0**[a]	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	30×10	0.84	0.76	0.86	0.82	0.76	0.76	**0.66**	0.76	0.78	0.84	0.78	0.82
	50×20	0.34	0.3	0.44	0.32	0.30	0.34	0.40	0.36	0.28	0.38	0.44	0.38
	Sum	1.18	1.06	1.3	1.14	1.06	1.10	1.06	1.12	1.06	1.22	1.22	1.20
0.05	10×5	0.70[b]	0.38	0.72	0.45	0.59	0.60	0.58	0.56	0.63	0.66	0.52	0.52
	30×10	2.43	2.69	2.83	2.53	2.70	2.63	2.56	2.79	2.63	2.39	2.57	2.33
	50×20	1.09	1.07	1.12	1.21	1.06	1.25	1.07	1.15	1.16	1.06	**1.01**	1.11
	Sum	4.22	4.14	4.67	4.19	4.35	4.48	4.21	4.49	4.42	4.11	4.10	3.96
0.1	10×5	0.64	0.51	0.39	0.43	0.65	0.55	0.55	0.50	0.44	0.59	0.55	0.48
	30×10	2.10	2.37	2.41	2.22	2.20	2.07	2.30	2.36	**1.96**	2.33	2.10	2.00
	50×20	0.86	0.90	0.92	0.96	1.03	0.96	0.97	1.08	0.90	1.10	1.13	**0.80**
	Sum	3.59	3.78	3.72	3.60	3.88	3.58	3.83	3.94	3.31	4.02	3.78	3.28
0.5	10×5	0.48	0.39	0.30	0.39	0.33	0.34	0.34	0.27	0.36	0.34	0.43	0.43
	30×10	1.99	2.05	1.82	1.98	1.88	2.06	1.97	2.10	1.79	1.94	2.10	1.70
	50×20	0.97	0.94	0.83	0.79	0.75	0.90	0.75	0.88	0.80	0.86	0.77	**0.63**
	Sum	3.44	3.37	2.95	3.16	2.96	3.30	3.06	3.25	2.96	3.14	3.29	2.76
1.0	10×5	0.40	0.33	0.34	0.24	0.23	0.38	0.36	0.35	0.30	0.27	0.28	0.36
	30×10	2.38	2.02	2.29	2.08	2.34	2.10	2.48	1.94	2.08	2.11	2.31	2.05
	50×20	0.83	0.92	0.80	0.81	0.90	0.85	0.88	1.02	0.86	0.96	0.73	**0.66**
	Sum	3.61	3.27	3.43	3.13	3.48	3.33	3.72	3.30	3.24	3.34	3.33	3.07

λ	Problem size	ISPTSA	ILPTSA	IERDSA	IEDDSA	IIMSTSA	IS/PSA	IHSESA	IPALSA	ICDSSA	IGUPSA	IDANSA	INEHSA
0	10×5	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	30×10	0.82	0.78	0.88	0.68	0.78	0.86	0.72	0.86	0.72	0.74	0.72	0.74
	50×20	0.36	0.32	0.34	**0.12**	0.42	0.46	0.38	0.50	0.34	0.32	0.42	0.40
	Sum	1.18	1.10	1.22	**0.80**	1.20	1.32	1.10	1.36	1.06	1.06	1.14	1.14
0.05	10×5	0.51	0.54	0.59	0.49	**0.38**	0.61	0.52	0.54	0.45	0.62	0.46	0.50
	30×10	2.57	2.60	2.64	2.81	2.36	2.38	2.53	2.54	2.82	2.39	2.76	**2.32**
	50×20	1.17	1.12	1.30	1.15	1.17	1.16	1.10	1.13	1.11	1.11	1.21	1.09
	Sum	4.24	4.26	4.52	4.45	3.91	4.14	4.15	4.22	4.38	4.11	4.43	**3.91**
0.1	10×5	0.59	0.65	0.55	0.42	0.64	**0.36**	0.46	0.70	0.48	0.44	0.66	0.37
	30×10	2.14	2.21	2.12	2.45	2.16	2.41	2.22	2.04	2.22	2.03	2.37	2.04
	50×20	0.94	0.91	1.19	1.02	1.02	1.05	1.10	0.99	1.06	1.04	1.04	0.82
	Sum	3.67	3.77	3.86	3.90	3.82	3.82	3.78	3.73	3.76	3.51	4.07	**3.23**
0.5	10×5	0.28	0.38	0.33	0.32	0.32	**0.25**	0.42	0.40	0.42	0.37	0.28	0.37
	30×10	1.85	1.70	2.06	2.24	2.07	2.09	1.91	1.93	1.93	1.87	1.90	**1.51**
	50×20	0.83	0.83	0.78	0.81	0.83	0.80	0.79	0.90	0.96	0.89	0.89	0.63
	Sum	2.96	2.91	3.17	3.38	3.22	3.15	3.11	3.23	3.31	3.13	3.07	**2.52**
1.0	10×5	0.30	0.28	**0.16**	0.24	0.27	0.26	0.38	0.40	0.32	0.21	0.22	0.37
	30×10	2.15	2.09	2.25	2.05	2.15	2.30	2.16	**1.88**	2.07	2.15	1.94	2.03
	50×20	0.93	0.81	0.83	0.90	0.82	0.89	0.91	0.84	0.90	0.88	0.83	0.67
	Sum	3.38	3.19	3.25	3.18	3.24	3.45	3.45	3.11	3.29	3.24	**2.98**	3.07

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

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

References

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

Alisantoso, D., Khoo, L.P., and Jiang, P.Y. (2003) "An immune algorithm approach to the scheduling of a PCB flexible flow shop", *The International Journal of Advanced Manufacturing Technology*, 22(11–12): 819–827.

Arthanari, T.S., and Ramamurthy, K.G. (1971) “An extension of two machines sequencing problem”, *Opsearch*, 8(1): 10–22.

Baker, K.R. (1974) “Introduction to Sequencing and Scheduling”, John Wiley & Sons, New York.

Brah, S.A., and Hunsucker, J.L. (1991) “Branch and bound algorithm for the flow shop with multiple processors”, *European Journal of Operational Research*, 51(1): 88–99.

Brah, S.A., and Loo, L.L. (1999) “Heuristics for scheduling in a flow shop with multiple processors”, *European Journal of Operational Research*, 113(1): 113–122.

Campbell, H.G., Dudek, R.A., and Smith, M.L. (1970) “A Heuristic algorithm for the n-Job m-Machine sequencing problem”, *Management Science*, 16(10): 630–637.

Dannenbring, D.G. (1977) “An evaluation of flow shop sequencing heuristics”, *Management Science*, 23(11): 1174-1182.

Gourgand, M., Grangeon, N., and Norre, S. (1999) “Metaheuristics for the deterministic hybrid flow shop problem,” In: *Proceeding of the International Conference on Industrial Engineering and Production Management,* IEPM’99, Glasgow: 136–145.

Guinet, A., Solomon, M.M., Kedia, P.K., and Dussauchoy, A. (1996) “A computational study of heuristics for two-stage flexible flowshops”, *International Journal of Production Research,* 34(5): 1399–1415.

Gupta, J.N.D. (1971) “A functional heuristic algorithm for the flow-shop scheduling problem”, *Operations Research Quarterly*, 22(1): 39-47.

Gupta, J.N.D. (1988) “Two-stage, hybrid flow shop scheduling problem”, *Journal of Operational Research Society*, 39(4): 359–364.

Gupta, J.N.D., Krüger, K., Lauff, V., Werner, F., and Sotskov, Y.N. (2002) “Heuristics for hybrid flow shops with controllable processing times”, *Computers and Operations Research*, 29(10): 1417–1439.

Gupta, J.N.D., and Tunc, E.A. (1994) “Scheduling a two-stage hybrid flowshop with separable setup and removal times”, *European Journal of Operational Research*, 77(3): 415–428.

Hsieh, J.C., Chang, P.C., and Hsu, L.C. (2003) “Scheduling of drilling operations in printed circuit board factory”, *Computers & Industrial Engineering*, 44(3): 461–473.

Jin, Z., Yang, Z., and Ito, T. (2006) “Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem”, *International Journal of Production Economics*, 100(2): 322–334.

Karacapilidis, N.I., and Pappis, C.P. (1996) “Production planning and control in textile industry: A case study”, *Computers in Industry*, 30(2): 127–144.

Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P. (1984), “Optimization by simulated annealing”, *Science*, 220(4598): 671–68.

Koulamas, C., Antony, S.R. and Jaen, R. (1994) “A survey of simulated annealing applications to operations research problems”, *Omega International Journal of Management Science*, 22(1): 41–56.

Kurz, M.E., and Askin, R.G. (2003) “Comparing scheduling rules for flexible flow lines”, *International Journal of Production Economics*, 85(3): 371–388.

Lin, H.T., and Liao, C.J. (2003) “A case study in a two-stage hybrid flow shop with setup time and dedicated machines”, *International Journal of Production Economics*, 86(2): 133–143.

Linn, R., and Zhang, W. (1999) “Hybrid flow shop scheduling: A survey” *Computers & Industrial Engineering*, 37(1–2): 57–61.

Lundy, M., and Mees, A. (1986) “Convergence of an annealing algorithm”, *Mathematical Programming*, 34(1): 111–124.

Moursli, O., and Pochet, Y. (2000) “Branch and bound algorithm for the hybrid flowshop”, *International Journal of Production Economics*, 64(1–3): 113–125.

Nawaz, M., Enscore, Jr. E., Ham, I. (1983) “A heuristic algorithm for the m-machine, n-job flowshop sequencing problem”, *Omega International Journal of Management Science*, 11(1)pp 91–95.

Nowicki, E., and Smutnicki, C. (1998) “The flow shop with parallel machines: A tabu search approach”, *European Journal of Operational Research*, 106(2–3): 226–253.

Palmer, D.S. (1965) “sequencing jobs through a multi-stage process in the minimum total time--a quick method of obtaining a near optimum”, *Operations Research Quarterly*, 16(1): 101–107.

Pinedo, M., and Chao, X. (1999) “Operations scheduling with applications in manufacturing and services”, Irwin/McGraw-Hill, New York.

Salvador, M.S. (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.

Santos, D.L., Hunsucker, J.L., and Deal, D.E. (1996) “An evaluation of sequencing heuristics in flow shops with multiple processors”, *Computers & Industrial Engineering*, 30(4): 681-691

Sriskandarajah, C, and Sethi, S.P. (1989) “Scheduling algorithms for flexible flowshops: worst case and average case performance”, *European Journal of Operational Research*, 43(2): 143–160.

Wang, W. (2005) “Flexible flow shop scheduling: Optimum, heuristics, and artificial intelligence solutions”, *Expert Systems*, 22(2): 78–85.

Wang, W., and Hunsucker, J.L. (2003) “An evaluation of the CDS heuristic in flow shops with multiple processors”, *Journal of the Chinese Institute of Industrial Engineers*, 20(3): 295–304.

Youssef, H., Sait, S.M., and Adiche, H. (2001) “Evolutionary algorithms, simulated annealing and tabu search: a comparative study”, *Engineering Applications of Artificial Intelligence*, 14(2): 167–181.