International Journal of Production Research,
Vol. 43, No. 14, 15 July 2005, 2895–2929

Taylor & Francis
Taylor & Francis Group

Flowshop-scheduling problems with makespan criterion: a review

S. REZA HEJAZI*[†] and S. SAGHAFIAN[‡]

[†]Department of Industrial Engineering, Isfahan University of Technology, Isfahan, Iran
[‡]Department of Industrial Engineering, Sharif University of Technology, Tehran, Iran

(*Received November 2004*)

This paper is a complete survey of flowshop-scheduling problems and contributions from early works of Johnson of 1954 to recent approaches of metaheuristics of 2004. It mainly considers a flowshop problem with a makespan criterion and it surveys some exact methods (for small size problems), constructive heuristics and developed improving metaheuristic and evolutionary approaches as well as some well-known properties and rules for this problem. Each part has a brief literature review of the contributions and a glimpse of that approach before discussing the implementation for a flowshop problem. Moreover, in the first section, a complete literature review of flowshop-related scheduling problems with different assumptions as well as contributions in solving these other aspects is considered. This paper can be seen as a reference to past contributions (particularly in $n/m/p/c_{\max}$ or equivalently $F/prmu/c_{\max}$) for future research needs of improving and developing better approaches to flowshop-related scheduling problems.

Keywords: Flowshop; Makespan; Dynamic programming; Heuristics; Metaheuristics; Simulated annealing; Genetic algorithm; Tabu search; Ant colony optimization

1. Introduction

Because of many economic and industrial applications, the flowshop problem has been concentrated on by many researchers with diverse classical assumptions and different objective functions and by implementing various optimization techniques. The regular flowshop problem consists of two main elements: (1) a group of M machines and (2) a set of N jobs to be processed on this group of machine. Each of the N jobs has the same ordering of machines for its process sequence. Each job can be processed on one and only one machine at a time (which means no job splitting), and each machine can process only one job at a time. Each job is processed only once on each machine. Operations are not preemptable and set-up times of operations are independent of the sequences and therefore can be included in the processing time. The scheduling problem is to specify the order and timing of the processing of the jobs on machines, with an objective or objectives respecting above-mentioned assumptions. The flowshop problem with makespan criterion can

*Corresponding author. Email: rehejazi@cc.iut.ac.ir be shown by $n/m/F/c_{\max}$ or equivalently $F//c_{\max}$, where both show an (n-job m-machine) flowshop problem with makespan criterion that can be defined as completion time at which all jobs complete processing or equivalently as maximum completion time of jobs. The former notation is suggested by Conway *et al*. (1967) and the latter is according to notation introduced by Graham *et al*. (1979). For such a problem, there are generally $(n!)^m$ different alternatives for sequencing jobs on machines. However, most of research has focused on development of a permutation flowshop schedule, which can be considered as a classical flowshop problem by adding the assumption: the jobs must be processed in the same sequence by each of the m machines. Consequently, the search space reduces to $n!$ in permutation flowshop problem. In the classical flowshop-sequencing problem, queues of jobs are allowed at any of m machines in processing sequence. In other words, in the classical flowshop, an infinite buffer is assumed and jobs may wait on or between the machines (Allahverdi *et al*. 1999). There are variants of this problem wherein jobs are not allowed to form queues. Zero-buffer and no-wait flowshop problems are some examples. Having buffers of zero capacity, a job i just finishing on machine r cannot advance to machine $r+1$ if this machine is still processing job i's predecessor in job sequence, rather job i must remain at machine r, thus temporarily denying machine r job i's successor in the job sequence until such time as job i can advance to machine $r+1$.

Abadi and Sriskandarajah (1995) described the blocking flowshop problem as follows. The flowshop has no intermediate buffer therefore a job cannot leave a machine until the next machine downstream is free. If that is not the case, the job (and the machine as well) is said to be blocked. Aldowaisan and Allahverdi (1998) described the case in which once a job begins its processing on machine 1 of the production line, that job must continue without delay to be processed on each of the m machines in line. Not only are there no integer stage buffers to hold delay jobs, but also no job may wait on one machine until the subsequent machine in line is free to begin processing on that job. Aldowaisan and Allahverdi (1998) refer to this as the no-wait flowshop problem. More recently (2003), they proposed two heuristics based on Simulated Annealing and Genetic Algorithm for the no-wait flowshop problem to minimize makespan. However, the no-wait flowshop problem was discussed by Piehler (1960), Reddi and Ramamoorthy (1972), Bonney and Gundry (1976), King and Spachis (1980), Gangadharan and Rajendran (1993) and Rock (1984) (while the former focused on heuristic methods and the latter dealt with the NP-completeness for three machine no-wait flowshop) and also it was completely described in the survey of Hall and Sriskandarajah (1996) but, some earlier research, such as Stafford (1988), Stafford and Tseng (1990), and Wismer (1972), called this the NIQ (no intermediate queues) flowshop problem. In this problem according to Stafford and Tseng (2001), jobs are held before machine 1 and launched only when they can be sequentially processed by all m machines without delays at any of the machines. It is noteworthy that zero-buffer and no-wait flowshop problems are equivalent for the two-machine problem when set-up times are included in processing times. For separable set-up times, however, there are two cases in the zero-buffer problem. In the first case, the set-up of the next job on machine 1 is not allowed until the current job releases machine 1. In the second case, the set-up for the next job on machine 1 can start as soon as machine 1 completes its processing of the current job. The first case seems to be more practical. Notice that the zero-buffer in the first case is equivalent to the no-wait problem (Aldowaisan and Allahverdi 1988, Allahverdi *et al.* 1999). Hall and Sriskandarajah (1996) gave a survey of machine scheduling problems with blocking and no-wait in processes. In addition, to a review of the computational complexity of a wide variety of no-wait and blocking scheduling problems, they described several well-documented applications of no-wait and blocking scheduling models and illustrated some ways in which the increasing use of some manufacturing methods gives rise to other applications. The no-wait flowshop problems have also been concentrated in much research regarding different criteria (e.g. Bertolissi 2000, Aldowaisan and Allahverdi 2004, *passim*).

Hybrid flowshop problems have attracted much research during the last few years. Zhixing *et al.* (2002) describe the no-wait hybrid flowshop problem as follows. Given a list of machine centres, each having a fixed number of parallel machines (in such a way that at least one stage contains several machines), there are n jobs $\{p_i | \leq i \leq n\}$ to be processed with the same fixed processing order on the machine centres, and each process must be carried out on at most one machine in every centre, without any interruption on or between machines during the process. Also a detailed survey of the application and research on this problem is given by Hall and Sriskandarajah (1996) and also new results in this field can be found in Kumar *et al.* (2000), Sawik (2000), Abadi *et al.* (2000), Aldowaisan (2001) and Znixin *et al.* (2002) and in the case study paper of Andres *et al.* (2004), who considered a three-stage hybrid flowshop environment.

In more real-life flowshop environments, the case can be considered as a SDST (sequence-dependent set-up times) flowshop problem. As mentioned above, in the classical flowshop problem, a job's set-up time is assumed to be independent of that job's position in the sequence, i.e. set-up times of operations are independent of sequences and therefore they can be included in the processing time of jobs. However, in many real-life flowshop problems such as the case in SMT (surface mount technology) or as PCB (printed circuit board) manufacturing environments, which is concentrated in many recently carried out research, the set-up times of jobs are sequence-dependent and even the problem is considered in real-time mode. Srikar and Ghosh (1986) reported an MILP model for this problem; Stafford and Tseng (1990) also reported; and Rios-Mercado and Barod (1998) gave another article in this field. Tseng and Stafford (2001) also proposed two other new MILP models for the SDST flowshop and managed to solve instances of up to nine jobs and nine machines in about 5 min of CPU time on a Pentium III running at 800 MHz. Stafford and Tseng (2002) also proposed two MILP models (referred to as WST and SGST) for a family of four different m-machine, n-job flowshop sequencing problems. Each of their models may be used to solve the regular, no intermediate queues (NIQ), sequence-dependent set-up times (SDST), and SDST/NIQ flowshop problems. Moreover, Lee and Shaw (2000) worked on a PCB environment with the set-up-dependent flowshop problem of sequencing a set of jobs that arrive in different combinations over time and developed and compared heuristic and metaheuristic procedures regarding makespan for this problem. Ruiz *et al.* (2004) considered the flowshop-scheduling problem with sequence-dependent set-up times and makespan criterion. They proposed two advanced genetic algorithms as well as several adaptations of existing advanced metaheuristics that have shown superior performance when applied to regular flowshops. Allahverdi *et al.* (1999) gave a complete review of scheduling problems involving set-up considerations. The review by Cheng *et al.* (2000) for flowshop-scheduling involving set-up times is also very suitable for interested readers. Allahverdi *et al.* (1999) have divided the flowshop problems (involving set-up considerations) into four categories: sequence-independent non-batch set-ups; sequence-dependent non-batch set-ups; sequence-independent batch set-ups; and sequence-dependent batch set-ups. A batch set-up problem occurs when part types are grouped into batches (or product families) and a (major) set-up time is incurred when switching between part types belonging to different batches, and, in some applications, a (minor) set-up is incurred for switching between part types within batches (i.e. from the same product families). In other words, a major set-up time depends only on the batch being switched to, and the minor set-up time depends only on the part type being switched to (Tang 1990, Allahverdi *et al*. 1999). The major research in first category with makespan criterion include: Yoshida and Hitomi (1979), who were among the first to investigate the flowshop wherein set-up times were separated from processing times with their extension of Johnson's rule; Yoshida and Hitomi (1979), who extended Johnson's (1954) model to the case where set-up times are separable from processing times; Sule (1982), Sule and Huang (1983), Khurana Bagga (1985) and Allahverdi (1995), who extended Yoshida and Hitomi (1979); Proust *et al*. (1991), Park and Steudel (1991), Gupta and Tunc (1994), Cao and Bedworth (1992) Allahverdi (1997), Rajendran and Ziegler (1997) and Gupta *et al*. (1997) also addressed different assumptions for sequence-independent non-batch set-ups to minimize makespan; and Lee and Shaw (2000) worked on a PCB manufacturing environment to minimize this objective function. Moreover, when time lags are considered in addition to set-up times, Khurana and Bagga (1984) provided an optimal solution for a two-machine flowshop to minimize makespan under two conditions: (1) the start and stop lags are equal for each job, and (2) once the processing time of a job is greater (smaller) on machine 1 than its processing time on machine 2, then the sum of processing and set-up times of this job on machine 1 is also greater (smaller) than that of the job on machine 2. Szwarc (1986) used the formula he developed in a previous paper (Szwarc 1983) to find an optimal solution for the same problem without the two conditions. In addition, he provided an approximate solution for the multiple machine case. Nabeshima and Maruyama (1984) also extended the two-machine problem of Szwarc by also considering separate removal and transportation times. They additionally addressed the three-machine problem under certain conditions. Rajendran and Ziegler (1997) investigated the problem of scheduling in a flowshop, where set-up, processing and removal times are separable with the objective of minimizing makespan. They developed heuristic algorithms by the introduction of simplifying assumptions into the scheduling problem under study.

In second category, i.e. sequence-dependent non-batch set-ups, Corwin and Esogbue (1974) addressed the two machine flowshop problem with a makespan objective, where the set-up times are sequence-dependent on the first (second) machine and sequence-independent on the second (first) machine. They showed the optimality of permutation schedules for both cases and established a dynamic programming formulation for each problem. Under the assumption that set-up cost is directly proportional to set-up time, Uskup and Smith (1975) provided a branch-and-bound solution for the two-machine flowshop problem with a total set-up cost criterion such that the schedule meets all dead lines. Gupta and Darrow (1985) generalized Corwin and Esogbue's (1974) problem by considering sequence-dependent set-up times on both machines for permutation schedules and established two heuristics for the problem. Gupta and Darrow (1986) considered the same problem and showed it to be strongly NP-hard even when set-up time is sequence-dependent on only one of the two machines. Other research in this category with makespan criterion includes: Gupta (1986), Szwarc and Gupta (1987), Rajagopalan and Karimi (1987), Gupta (1988), Simons (1992), Gupta *et al.* (1995) and Rios-Mercado and Bard (1996, 1997). In third category, i.e. regarding sequence-independent batch set-up problems with makespan criterion, research includes Hitomi and Ham (1976), Hitomi *et al.* (1977), Ham *et al.* (1985), Baker (1990), Vakharia and Chang (1990a, b), Logendran and Sriskandarajah (1993), Skiron-Kapov and Vakharia (1993), Sridhar and Rajendran (1994), Zdrzalka (1994), Stoskov (1996), Li (1997) and Danneberg *et al.* (1998). When regarding flowshop problems with sequence-dependent batch set-up times and makespan criterion, some known results are those of Vakharia *et al.* (1995) and Schaller *et al.* (1997) who presented branch-and-bound approaches as well as several heuristics. Moreover, note that in some cases, parallel set-ups may be possible. As Stafford and Tseng (1990) described, if job *i*'s predecessor in the sequence has completed its processing on machine *r*, then the operator of this machine should begin the set-up for job *i* in anticipation that job *i* will be the next job to arrive for processing at that machine. There is no reason to set machine *r* stand idle for a time and then do the job *i* set-up after job *i* is free to be processed on machine *r*. Another important class of these scheduling problems is constrained by the no-idle production environment, where machines work continuously without any interruption from the start of the first job processing to the last job completion. Such no-idle environment, regarding make-span as the objective function, is particularly important when a very expensive equipment is used or when its using cost depends on time consumption (Saadani *et al.* 2004). There are not so many papers dealing with $F/no-idle/C_{\max}$ (no-idle flowshop problem with makespan criterion as notation introduced by Graham *et al.* 1979). The complexity of this problem has only been mentioned in Tanarv (1994). Baptiste and Hguny (1997) have proved the NP-hardness of a three-machine no-idle flowshop problem with makespan criterion. More recently, Saadani *et al.* (2004) have investigated the $F/no-idle/C_{\max}$ and modelled it as a travelling salesman problem. They proposed an adaptation of the well-known nearest insertion rule to solve the problem.

As mentioned above, the traditional serial flowshop captures the essence of sequential processing. Lee *et al.* (1993) and Potts *et al.* (1995) introduced the concept of concurrency in a flowshop environment by proposing a two-stage flowshop with concurrent processing in the first stage. In that two-stage assembly flowshop, a number of concurrent operations are performed in the first stage followed by a single (assembly) operation in the second stage. The assembly operation, in accordance with the flowshop principle of serial processing, can commence only after all concurrent first-stage operations are completed (Koulamas and Kyparisis 2004). In Koulamas and Kyparisis (2004) the concept of concurrency by introducing a new shop (the concurrent flowshop where each job consists of several components, each of which is processed on a dedicated flowshop) regarding makespan as the objective function is extended. More than the concurrent flowshop environment, in some research the reduction of jobshop to flowshop problems has been concentrated. Guinet (2000) studied the problem of scheduling N independent jobs in a jobshop environment in which each job must be processed on at most M machines according to individual routes, regarding the makespan as the criterion. He proposed first to reduce the jobshop problem to a flowshop problem with job precedence constraints. Then, an extension of Johnson's rule (it will be discussed below) was defined to solve it. He showed the optimality of the extended Johnson's rule for two machine jobshop problems and in addition to the rule efficiency for some three and four machine jobshop problems.

Researches in flowshop problem also can be assessed from the viewpoint of different objective functions and various optimization techniques. Most research is concentrated on makespan (or maximum flow time), which can be defined as the completion time at which all jobs complete processing. More specifically it is well-known in the literature that the $c_{\max}$ (π), i.e. the makespan of the processing order given by π can be found by:

$$c_{\max}(\pi) = \max_{1 \le t_1 \le t_2 \le \cdots \le t_{m-1} \le n} \left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right)$$

where $\pi(j)$ is the jth element of permutation π. In our survey, we mainly consider this criterion. However, fewer papers deal with objectives involving the due dates of jobs. Gelders and Sambandam (1978) suggested four heuristics with the objective of minimizing the sum of tardiness and flow time. Ow (1985) developed a scheduling procedure to minimize the sum of waited tardiness for a special case in which the processing times are proportionate. Grabowski *et al.* (1983), Sen *et al.* (1989) and Townsend (1977) sought optimal solutions with branch-and-bound techniques, in which Sen *et al.* considered minimizing the mean tardiness on two machine flowshops, while Grabowski used the maximum lateness among all jobs as the performance criterion, and Townsend tried to minimize the maximum tardiness on m-machine flowshops. Yeong-Dae Kim (1993) worked on heuristic for minimizing mean tardiness for flowshop-scheduling in his article and Zhu and Meady (2000) by trying to minimize the sum of earliness/tardiness in multi-machine scheduling with MIP approach and Seo *et al.* (1999) by working on minimizing mean-squared deviation of completion time with maximum tardiness constraint, include some other objective function and criteria in generally scheduling, and flowshop problem as well. Although there are many more contributions regarding different objective functions (such as mean or total or sum of flow time) which can be reviewed through their own surveys. Moreover, there are some papers wherein bicriteria and multi-objective flowshop problems are concentrated. A survey of multicriteria scheduling problems is provided by T'kindt and Billaut (2001). Multicriteria scheduling problems can be modelled in three different ways. First, when the criteria are equally important, all the efficient solutions for the problem can be generated. Then by using multi-attribute decision methods, tradeoffs can be made between these solutions. Second, when the criteria are weighted differently, an objective function can be defined as the sum of weighted functions and transform the problem into a single criterion-scheduling problem. Finally, when there is a hierarchy of priority levels for the criteria, the problem can first be solved for the first priority criterion, ignoring the other criteria and then solved for the second priority criterion under the constraint that the optimal solution of the first priority criterion does not change (Gupta *et al.* 2001). In many practical situations, scheduling problems generally involve multiple objectives (Gupta and Dudek 1971, Panwalker *et al.* 1973, Gupta *et al.* 2001). Because of such situations, multicriteria scheduling problems are receiving much attention recently (Nagar *et al.* 1995, Gupta *et al.* 2001). Some research in the field of multiple objectives scheduling include Daniels and Chambers (1990), Rajendran (1992, 1994), Dileepan (1998), Fry *et al.* (1989), Hoogeveen (1992), Nagar *et al.* (1995, 1996), T'kindt *et al.* (2002, 2003) and Gupta *et al.* (1999–2002). Nagar *et al.* (1995) proposed a branch-and-bound approach to solve the two-machine flowshop problem with makespan and total flow time objectives. Moreover, Nagar *et al.* (1996) proposed a combined branch-and-bound and genetic algorithm based approach to solve the two machine flowshop problem with objectives of makespan and mean flow time with respect to the linearly combined objectives Neppalli *et al.* (1996) applied the genetic algorithm to solve the two machine flowshop problem with objectives of makespan and total flow time. They searched the effective solutions through partitioning the population into two sub-populations according to the criteria, i.e. makespan and total flow time. Murata *et al.* (1996) proposed the multi-objective genetic algorithm (MOGA) to search the Pareto optimal solutions of bi-objective (makespan and total tardiness) and tri-objective (makespan, total flow time, and total tardiness) scheduling problems. Ishibuchi and Murata (1998) proposed a hybrid genetic algorithm by introducing local search into the procedures of genetic algorithm. Sridhar and Rajendran (1996) applied genetic algorithm to the flowshop and cellular manufacturing systems. They considered makespan, total flow time, and machine idle time as the performance measures. Rajendran (1992) considered the two-stage flowshop-scheduling problem with the objective of minimizing total flow time subject to obtaining the optimal makespan. He developed a branch-and-bound and two heuristic algorithms for the problem. Many other algorithms are also developed and applied to the bi- and multi-objective scheduling problem including makespan as one of objective functions. Ho and Chang (1991) proposed a heuristic approach based on the CDS algorithm (CDS algorithm will be described in this paper) to solve the $n/m/F/C_{\max}$, ΣF (n-job m-machine flowshop problem with makespan and total flow time objectives). Gangadharan and Rajendran (1994) applied the simulated annealing method to solve $n/m/F/C_{\max}, \sum F$. Rajendran (1995) proposed a heuristic method with respect to the same problem by interchanging the potential jobs to improve the system performance. Jin *et al.* (2001) investigated the problem of multi-objective evolution strategies by adapting weighted aggregation. Gupta *et al.* (2001) considered the two-machine flowshop-scheduling problem where it is desired to find a minimum total flow time schedule subject to the condition that the makespan of the schedule is minimum. Chang *et al.* (2002) proposed the gradual-priority weighting (GPW) approach to search the Pareto optimal solutions for the multi-objective flowshop-scheduling problem respecting makespan, total flow time, total tardiness, and maximum tardiness. Their approach search feasible solution space from the first objective at the beginning and towards the other objectives step by step. T'kindt *et al.* (2003) developed mathematical programming formulations, a branch-and-bound algorithm, and a heuristic algorithm for solving the two-machine flow-shop-scheduling problem with the objective of minimizing total completion time, subject to the constraint that the makespan is minimum. Allahverdi and Aldowaisan (2004) recently addressed the m-machine no-wait flowshop-scheduling problem with a weighted sum of makespan and maximum lateness criteria. They proposed a hybrid simulated annealing and a hybrid genetic heuristics, which can be used for the single criterion of makespan or maximum lateness, or the bicriteria problem. Rajendran and Ziegler (2004) recently proposed two ant colony optimization algorithms for the problem of scheduling in permutation flowshops with the objective of minimizing the makespan, followed by the consideration of minimization of total flow time of jobs. Chakravarthy and Rajendran (1999) developed a heuristic algorithm for a flowshop problem with bicriteria of makespan and maximum tardiness problem. More recently, Allahverdi (2004) considered the *m*-machine flowshop problem with the objective of minimizing a weighted sum of makespan and maximum tardiness. Two types of the problem are addressed in this research. The first type is to minimize the objective function subject to the constraint that the maximum tardiness should be less than a given value. The second type is to minimize the objective without the constraint. He developed a new heuristic that is proposed and compared with two existing heuristics. There are also many other papers in the field of multi-criteria and/or bi-criteria flowshop problems. However, as mentioned above most research in flowshop-scheduling has concentrated on makespan as their single performance criterion. Respecting this objective function, various heuristic methods have been developed for a flowshop problem. Ashour (1970), Barany (1981), Bonney and Gundry (1976), Campbell *et al.* (1970), Dannenbring (1977), Gupta (1971, 1972), Ho and Chang (1991), Hundal and Rajgopal (1988), King and Spachis (1980), Nawaz *et al.* (1983), Obgu and Smith (1990), Osman and Potts (1989), Page (1961), Palmer (1965), Park *et al.* (1984), Taillard (1990), priority rules in MacCarthy and Liu (1993), neighbourhood tabu search in Widmer and Hertz (1989), and the use in scheduling Kanban-controlled systems by Ramanan and Rajendran (2003) are just some examples of developers of heuristics and metaheuristics for a flowshop problem with makespan as objective function. There are also some exact methods to find the optimum solution include dynamic programming (Held and Karp 1962), branch-and-bound (Ignall and Schrage 1965, Lomnicki 1965, Lageweg *et al.* 1978, Daniels and Mazzola 1994), elimination rules as in Baker (1975), row generation algorithms as in Frieze and Yadegar (1989) and Dudek *et al.* (1992), integer programming and complete enumeration as well. We will discuss some of these exact methods in section 2.

Because the *n*-job *m*-machine flowshop sequencing problems belong to the class of NP-hard problems (Rinnooy Kan 1976, Lentra *et al.* 1977, Gonzalez and Sahni 1978), the computational requirements for obtaining an optimal solution increase exponentially as problem size increase, i.e. in viewpoint of the combinatorial complexity and time constraints, most of the large problems can be solved only by heuristic methods. Consequently, some constructive heuristics were developed for the problem. As these contributions have been clues for later developers, to give a better view of scheduling techniques for flowshop problem with makespan criterion, we will cover most of constructive heuristics (such as Palmer, Gupta, CDS and NEH) as well as briefly remarking on many other constructive and improvement heuristics in section 3 of this paper. Additionally, some modern heuristics or so-called metaheuristics and some evolutionary algorithms have been implemented to solve the flowshop problem in much research. These include SA (Simulated Annealing) with different cooling and perturbation schedules, GA (Genetic Algorithm) with diverse crossover and mutation operators, TS (Tabu Search) methods with different approaches, ACS (Ant Colony System), ANN (Artificial Neural Network) approaches with different designs and some hybrid algorithms and a few of other neighbourhood search approaches. In section 4, we will focus on such approaches by introducing and implementing SA, GA, TS and ACS. In these approaches, we will consider the most recent research and developed techniques. Moreover, while some scheduling techniques have been clues for later research, in each section, unlike many other surveys, we will also deal with details of scheduling techniques as well as noting the contributions in that field.

2. Exact methods and small size problems

As mentioned above, flowshop problems belong to a class of NP-hard problems. Therefore, exact methods are just suitable for some small size problems. Some demonstrated properties can be used in these methods. The two most important are: (1) with respect to any regular measure of performance, it is sufficient to consider only schedules in which the same job sequence occurs on machines 1 and 2; and (2) with respect to makespan as a measure of performance, it is sufficient to consider only schedules in which same job sequence occurs on machines $m-1$ and m. These two properties lead to constitute a dominant set of $(n!)^{m-1}$ schedules for any regular objective function and of $(n!)^{m-2}, m \geq 3$ for makespan problems.

Hereby, to become more familiar with early contributions, we describe dynamic programming approach and also, the well-known algorithm in the literature, which implements Johnson's rule. Both can be used for $m=2$ makespan problems or $n/2/F/c_{\max}$ ($n/m/F(p)/c_{\max}$ or equivalently $F/prmu/c_{\max}$ shows n-job, m-machine flowshop (permutation flowshop) problem with makespan criterion while the former is suggested by Conway *et al.* (1967) and latter is according to notation introduced by Graham *et al.* (1979) and leading same optimal solutions. Moreover, the latter has been extended for $m=3$ or $n/3/F/c_{\max}$. In addition, readers can refer to branch-and-bound approaches (Lageweg *et al.* 1978, Ignall and Schrage 1965, Lomnicki 1965, 1978, Daniels and Mazzola 1994), elimination rules (Baker (1975), row generation algorithms (Frieze and Yadegar 1989, Dudek *et al.* 1992) and even MILP models for SDST (sequence-dependent set-up times) and SSIST (separable, sequence-independent set-up times) problems (Srikar and Ghosh 1985, and research by Tseng and Staffored).

2.1 *Dynamic programming*

This approach concentrated in some papers for small size flowshop problems including Held and Karp (1962). Here we describe this method for $n/2/F/c_{\max}$. Let a_i, be representative for t_{1i} (the processing time of job i on machine1) and b_i for t_{2i}. Suppose $\sigma_1, \sigma_2, \ldots, \sigma_k$ is a permutation prefix defining a schedule for jobs $T_1, T_2, \ldots, T_k$. For this schedule let f_1 and f_2 be the time at which the processing of jobs $T_1, T_2, \ldots, T_k$ is completed on processors (machines) 1 and 2, respectively. Let $t = f_2 - f_1$. The state of the processors (machines) following the sequence decisions is completely characterized by t. Let $g(s, t)$ be the length of an optimal schedule for the subset of jobs s under the assumption that machine 2 is not available until time t. The length of an optimal schedule for job set $\{1,2,\ldots,n\}$ is $g(\{1,2,\ldots,n\},0)$. Since the principle of optimality holds, we obtain:

$$g(\{1,2,\ldots,n\},0) = \min_{1 \leq i \leq n}\{a_i + g(\{1,2,\ldots,n\} - \{i\}, b_i)\}$$

This equation can be simply generalized for arbitrary s and t as follows:

$$g(s,t) = \min_{i \in s}\{a_i + g(s - \{i\}, b_i + \max\{t - a_i, 0\})\}.$$

The term $\max\{t - a_i, 0\}$ comes into above equation as task t_{2i} cannot start until $\max\{a_i, t\}$. Hence, $f_2 - f_1 = b_i + \max\{a_i, t\} - a_i = b_i + \max\{t - a_i, 0\}$.

It is easy to show that above equation leads to the well-known Johnson's rule although that has another proof itself which is available in almost all sequencing books (e.g. Baker 1974).

2.2 *Johnson's rule and extensions*

In the area of flowshop problems, scheduling theory has been strongly influenced by Johnson's early works (1954). The pivotal influence of his works has had some definite advantages, first by emphasizing the properties of permutation schedules; he focused flowshop research on problems of manageable size. Second, the two-machine analysis seemed to have captured the essence of larger problems. Thus, where Johnson's rule provided a basis for the development of a heuristic approach to larger problems (as in Gupta and CDS procedures) it was remarkably successful.

The Johnson's rule which is the most well known optimal rule applicable to a large class of flowshop problems says that job i precedes job j in an optimal sequence if: $\min\{t_{i1},t_{j2}\} \leq \min\{t_{i2},t_{j1}\}$. Implementing Johnson's rule, the flowshop problem for $m=2$ and makespan as performance criterion or $n/2/F/c_{\max}$ can be optimally solved by the following famous algorithm:

Step 1: Find $\min_i\{t_{i1}, t_{i2}\}$.
Step 2a: If the minimum processing time requires machine 1, place the associated job in the first available position in sequence. Go to Step 3.
Step 2b: If the minimum processing time requires machine 2, place the associated job in the last available position in sequence. Go to Step 3.
Step 3: Remove the assigned job from consideration and return to Step 1 until all positions in sequence are filled. Another shape of this algorithm may be described as follows:

Step 1: Let $U=\{j|t_{j1}<t_{j2}\}$ and $V=\{j|t_{j1}\geq t_{j2}\}$.
Step 2: Arrange the members of set U in non-decreasing order of t_{j1} and members of set V in non-increasing order of t_{j2}.
Step 3: An optimal sequence is the ordered set U followed by the ordered set V.

Moreover, there are some extensions of Johnson's rule, one is for $m=3$ and makespan criterion. He showed that a generalization is possible when the second machine is dominated (i.e. when no bottleneck could possibly occur on the second machine). This extension is described below:

1. If $\min_k\{t_{k1}\} \geq \max_k\{t_{k2}\}$ then job i precedes job j in an optimal schedule if:
$$\min\{t_{i1}+t_{i2}, t_{j2}+t_{j3}\} \leq \min\{t_{i2}+t_{i3}, t_{j1}+t_{j2}\}.$$
2. If $\min_k\{t_{k3}\} \geq \max_k\{t_{k2}\}$ then job i precedes job j in an optimal schedule if:
$$\min\{t_{i1}+t_{i2}, t_{j2}+t_{j3}\} \leq \min\{t_{i2}+t_{i3}, t_{j1}+t_{j2}\}.$$

To apply these results in an algorithm, it is possible to use the main algorithm implementing Johnson's rule, described previously, with first step of seeking a minimum in the form of $\min\{t_{i1}+t_{i2},\ t_{i2}+t_{i3}\}$ instead of seeking minimum processing time. Additionally, if there is no dominance present, it is also known that if that main algorithm implementing Johnson's rule, yields the same optimal sequence for two-machine sub problems represented by the set $\{t_{i1},t_{i2}\}$and $\{t_{i2},t_{i3}\}$then that sequence is optimal for the full three-machine problem.

There are other types of generalization for two-machine case, such as analyses of Mitten (1959) for a two-machine problem involving the addition of start- and stop-lags. As Baker described of mitten works, in many practical situations, it maybe possible to begin operation 2 before operation 1 is entirely complete. This structure may appear where jobs consist of large lots that may be split into sublots and where completed sublots may proceed to a subsequent operation without waiting for the fall lot to be processed. This over structure, often called lap phasing, may also arise where the flow of work is conveyor-driven. With lap phasing there is a specific interval a_j, called a start-lag, such that operation $j2$ can be started as soon as a_j time units after operation $j1$ begins. In particular, the strict precedence requirement of the flowshop model requires that $a_j = t_{j1}$ but lap phasing allows $a_j < t_{j1}$.

Analogously, Mitten allowed for an interval b_j called a stop-lag such that operation $j2$ must not complete any earlier than b_j time units often operation $j1$ completes. Allowing for $a_j > 0$ and for all b_j, Mitten extended Johnson's rule as described in the following algorithm:

Step 1: Let $U = \{j|t_{j1} < t_{j2}\}$ and $V = \{j|t_{j1} \geq t_{j2}\}$.
Step 2: Define $y_j = \max\{a_j - t_{j1}, b_j - t_{j2}\}$. Arrange the members at set U in non-decreasing order of $t_{j1} + y_i$ and members of set V in non-increasing order of $t_{j2} + y_j$.
Step 3: An optimal sequence is the ordered set U followed by the ordered set V.

Moreover, Mitten's result can be employed for larger flowshop problems in which the first and last machines are the only bottleneck machines. Suppose all jobs require the first operation to be performed on machine 1 and the third operation on machine 3. If the intervening operation on each job can be performed independently of all other jobs (as if, for example, n parallel machines were available for this purpose), the intervening operation could be treated as part of a start-lag and Mitten's result employed obviously. This reasoning could be extended to larger flow shops, as long as only machines 1 and m are bottleneck machines. However, it is remarkable that Mitten's result holds only for permutation schedules and may not be an optimal schedule for such large flowshop problems.

Moreover, the two-machine flowshop problem has been considered in much research with different objectives and assumptions. For instance, some recent examples and applications are as follows. Allahverdi *et al.* (2002) used a two-machine flowshop problem with maximum lateness objective to model data objects for WWW applications. He has also worked on some other applications of two-machine flowshop problems (Allahverdi *et al.* 2000, 2001) such as the relation of a three-tired Internet database and this problem; Cheng and Liu (2003) and Wang and Chang (2001) considered a two-machine no-wait flowshop with availability constraints; Dileepan (2003) also focused on a two-machine no-wait flowshop; T'kindt *et al.* (2003) worked on a two-machine with secondary criterion; Sung and Kim (2002) considered a two-machine flowshop regarding makespan and with dynamic arrivals; Lin (2001) worked on a two-machine flowshop with due date constraints; Gupta *et al.* considered minimizing total flow time in a two-machine flowshop problem with minimum makespan; Yong and Chern (2000) considered two-machine flowshop group scheduling; Allahverdi (1995) concentrated on a two-machine flowshop problem with sequence-independent set-up times to minimize mean flow time; Blazewicz *et al.* (2001) developed heuristic algorithms for a two-machine flowshop problem with limited machine availability; and Presman *et al.* (2001) considered a stochastic two-machine flowshop with limited WIP for average cost optimal policy. Also batch scheduling in the no-wait two-machine flowshop to minimize the makespan (Lin and Cheng 2001) and scheduling in a two-machine flowshop with batch processing machines for earliness/tardiness measure under a common due date (Sung and Min 2001), are some other titles of recent research that all show the considerations on a two-machine flowshop problem. However, although there is much research on this type of flowshop problem, they do not demonstrate much real-world applications of the two-machine flowshop problem. A lack of these real-world applications and case studies in this field still seems perceivable and future work should focus more on industrial and other uses of this problem such as in the works of Allahverdi *et al.*

3. Constructive heuristic algorithms

As discussed above, belonging to the NP-hard class (Rinnooy Kan 1976, Lentra *et al.* 1977, Gonzalez and Sahni 1978), computational requirements of exact methods such as dynamic programming, branch-and-bound, elimination approaches, row generation algorithms and integer programming and complete enumeration, for flowshop problem, are severe for large problems. Consequently, heuristic approaches have been proposed in much research including Page (1961), Dudek and Teuton (1964), Palmer (1965), Campbell *et al.* (1970), Gupta (1971), Gundry (1976), Dannenbring (1977), King and Spachis (1980), Stinson and Smith (1982), Nawaz *et al.* (1983), Rajgopal (1988), Koulamas (1998), Sarin and Lefoka (1993), Davoud Pour (2001) and Framinan (2003). In addition, to such constructive heuristics, there are a few improvement heuristics including: Dannenbring (1977), Ho and Chang (1991) and Suliman (2000). Hereby, we describe some early constructive heuristic algorithms that have been clues for researchers to develop many other well-known scheduling techniques in the literature. Other above-mentioned procedures may be reviewed by interested readers themselves. Moreover, in a recent review by Ruiz and Maroto (2004), evaluations of such procedures in addition to some other metaheuristics also can be found.

3.1 *Palmer algorithm*

Palmer (1965), suggested his algorithm using the concept of a 'slope index' for each job that is a measure of whether a job proceeds from a shorter to a longer processing time in the sequence. The sequence is then constructed with the descending slope indices, with the idea that jobs that tend to proceed from shorter to longer processing times in the sequence of operations are processed earlier. While there might be several ways of implementing this precept, Palmer proposed a slope index for job i, SI_i, as:

$$SI_i = -\sum_{j=1}^{m} [m - (2j - 1)] t_{ij}/2$$

where t_{ij} is the processing time for job i on machine j and m is the number of machines. Then a permutation schedule is determined using the job ordering:

$$SI_{[1]} \geq SI_{[2]} \geq \cdots \geq SI_{[n]}$$

Obviously, for $m = 2$, this algorithm is slightly different from Johnson's algorithm. Moreover, a later developed heuristic showed that the Palmer heuristic is not that effective.

3.2 *Gupta algorithm*

Gupta (1971) proposed that for $m > 2$, the job index can be calculated as:

$$SI_i = e_i / \min_{1 \leq k \leq m-1} \{t_{ik} + t_{i(k+1)}\}$$

where

$$e_i = \begin{cases} 1 & \textit{if} \quad t_{i1} < t_{im} \\ -1 & \textit{if} \quad t_{i1} \geq t_{im} \end{cases}$$

Then the job ordering:

$$SI_{[1]} \geq SI_{[2]} \geq \cdots \geq SI_{[n]}$$

makes a good permutation schedule. He also noted that when Johnson's rule is optimal in the three-machine case, it is in the form of the above-mentioned calculations for $m = 3$. He then compared this heuristic with Palmer's and found it generated better schedules than that in a substantial majority of cases. Additionally, Gupta investigated a set of other heuristics that are also based on construction via transitive rules.

3.3 *CDS algorithm*

Campbell *et al.* (1970) proposed an algorithm for makespan problems, which is called the CDS algorithm. Using two main principles, this procedure makes good solutions: (1) it uses Johnson's rule in a heuristic way and (2) it generally creates several schedules, the best one of which should be chosen.

The CDS algorithm creates $m - 1$ artificial two-machine problems and then solves them by implementing Johnson's two-machine algorithm. Then, the best $m - 1$ obtained solution becomes the best solution for the main m-machine makespan problem. Generally, processing times for the artificial two-machine subproblems for ith job on jth machine at stage k should be calculated as:

$$t'_{i1} = \sum_{j=1}^{k} t_{ij} \quad \text{and} \quad t'_{i2} = \sum_{j=m-k+1}^{m} t_{ij}$$

Campbell *et al.* then tested their algorithm and compared the results with Palmer's heuristic. They found that the CDS algorithm was generally more effective for both small and large problems. In addition, computing times were of the same magnitude for $n \leq 20$.

3.4 *NEH algorithm*

Nawaz *et al*. (1983), proposed their so-called new heuristic based on the assumption that a job with more total processing time on all machine should be given higher priority than jobs with less total processing time on all machines. Therefore, they proposed initially to arrange jobs in descending order of total processing time (which is simply calculated by $\sum_{j=1}^{m} t_{ij}$ for job *i*th). Then, NEH heuristic uses the idea of partial sequencing based on this mentioned order. A sequence is constructed by introducing one job at a time from that unscheduled order into the partial sequence. If there were k jobs in the previous partial sequence, at each job introduction the best partial sequence among $k+1$ possible partial sequences is chosen. That is, the new job can be placed in one of the $k+1$ possible places in the partial sequence and after choosing the best place of this job, regarding the obtained makespan, in the partial sequence, this partial sequence is fixed for the remaining procedure, which is introduction of a new job. As for the Nawaz *et al*. proposal, this introduction is based on the descending order of total processing times on all machines. By $n(n-1)/2-1$ total enumeration, the heuristic sequence and consequently the heuristic solution for makespan problem can be obtained. Moreover, computational studies such as Nawaz *et al*. (1983) and Turner and Booth (1987) showed that the NEH heuristic performs best among several well-known constructive heuristics, which is discussed here. However, the procedure of Koulamas (1998) (not assuming permutation hypothesis), Sarin and Lefoka (1993) and Pour (2001) are claimed to be superior to NEH.

4. Metaheuristic and evolutionary approaches

Previous sections discussed some famous constructive heuristics. Hereby, we focus on some implemented modern heuristics, which are improvement algorithms, for flowshop problem. The aim of using such algorithms is usually to guide the search to overcome local optimality and improve initial feasible solutions. There are many of these algorithms that has been implemented in flowshop-scheduling problems including: Simulated Annealing (SA), Genetic Algorithms (GA), Tabu Search (TS) methods, Greedy approaches, Variable-Depth search approach, Pilot methods, Hill climbing procedures, and Ant Colony Optimization (ACO)-based methods. Additionally, hybrid algorithms, combining some of these algorithms, have been developed in much research. The following sections will concentrate on some strong and basic metaheuristic approaches, and in addition, the following will be discussed: metaheuristics. The use of ANNs by Lee and Shaw (2000) for real-time sequencing and by El-Bouri *et al*. (2000) for the implementation of an ANN approach to sequencing jobs on a single machine can be referred to by interested readers. This review will consider recent research and the developments of these approaches for flowshop permutation problem regarding makespan as performance criterion.

4.1 *Simulated annealing approach*

Kirkpatrick *et al*. (1983) introduced Simulated Annealing (SA) and Creny (1985) considered the analogy between the annealing process of solids and the process of solving combinatorial optimization problems. However, it was originally developed as a simulation model for a physical annealing process of condensed matter (Metropolis *et al.* 1953). Laarhoven and Aarts (1987) gave a comprehensive discussion of the theory and review of various applications. In addition, they showed that the simulated annealing process converges to the set of global optimal solutions under certain conditions. Koulamas *et al.* (1994) also applied SA to a large number of optimization problems in a variety of application areas.

More research has been done on the application of SA to sequencing problems including Osman and Potts (1989), Matsuo *et al.* (1989), Obgu and Smith (1990), Ishibuchi *et al.* (1995), Zegordi *et al.* (1995), Liu (1997, 1999), Wodecki and Bozejko (2001) and Wang and Zheng (2003). The main procedure of SA can be described as follows. It starts from an initial solution to the problem and then generates a new trial solution from the neighbourhood at the current solution. If the new solution is better than the current solution, it is accepted and used as the new current solution. Otherwise, it may be accepted or rejected depending on an acceptance probability, which is determined by the difference between objective function of the two solutions and by a control parameter called temperature, following the convention in thermodynamics. This process then continues from the new current solution. Initially, the temperature is set at high level, as in annealing, so that almost all moves will be accepted. It is then decreased slowly during the procedure until almost no move will be accepted. In other words, SA procedure can be generally described as follows:

1. Initialization: parameters of annealing schedule.
2. Select an iteration mechanism: a simple prescription to generate a transition from current state to another state by a small perturbation.
3. Evaluate the new state, compute $\Delta E =$ (value of current state − value of new state).
4. If the new state is better, make it current state, otherwise probabilistically accept or reject it (with a determined probability function usually called acceptance probability function).
5. Based on stopping rules, either stop or continue iterations at Step 2.

For implementing an SA in particular applications, such as for flowshop problem, ways of generating neighbourhood solutions, often called a perturbation scheme, and the so-called cooling schedule, which is a temperature decreasing scheme, as well as a trade-off between solution quality and computation time, are very important to be customized. Moreover, latter consists of determining following parameters:

- Initial value of the temperature.
- Function to determine how the temperature is to be changed (decreased).
- Condition of the Metropolis equilibrium under each temperature, i.e. the number of iterations to be performed at each temperature.
- Stopping criterion to terminate the algorithm, i.e. the final value of the temperature parameter.

For a flowshop problem, different perturbation schemes have been considered. Sridhar and Rajendran (1993) made use of three perturbation schemes (adjacent interchange, insertion scheme and random interchange scheme). Osman and Potts (1989) adopted interchange neighbourhood and shift neighbourhood.

Ogbu and Smith (1990) chose insertion and the pairwise exchange as the perturbation scheme. Liu (1999) worked on neighbourhood size and its effect on the simulated annealing for a flowshop problem. He used a variable size neighbourhood for his proposed SA algorithm and designed that to vary as a function of number of trials so far: $size = 1 + (n/2-1)(1-t/T)^{0.4}$, where t is the number of trials so far and T is the total number of trials for the whole process. The neighbourhood size for an exchange method can be characterized by the distance between the position of the two jobs being exchanged in the sequence. While starting from any of these two jobs, it is possible to count to left or right. The distance between the two jobs in the sequence can be considered by the smaller number. Pengtian *et al.* (1999) used six perturbation schemes as follows:

- Interchanging two adjacent jobs.
- Interchanging two jobs.
- Moving a single job.
- Moving a subsequent of jobs.
- Reversing a subsequent of jobs.
- Reversing and/or moving a subsequence of jobs.

Choosing appropriate cooling scheme or annealing schedule is also related to trade-off between solution quality and computation time. As mentioned above, it consists of determining four items. Usually cooling occurs by a reducing factor such as α and a recursive function such as $\beta_n = \alpha\beta_{n-1}$, where n is the stage in which cooling schedule is placed. Acceptance probability is usually determined by the function $\exp(-\delta/k\beta_n)$, where δ is the difference between a new solution, S_0, and a previous solution S, i.e. regarding makespan as objective criterion $\delta = c_{\max}(s) - c_{\max}(s_0)$. Additionally, it is known that if c_{ij} denotes the completion time of ith job in the sequence on machine $1, 2, \ldots, j$, for different m-machine n-job flowshop problems with different interstage storage policies, $c_{\max}$ is c_{nm}, where c_{ij} can be defined as follows:

- For unlimited intermediate storage (UIS): $c_{ij} = \max[c_{(i-1)j}, c_{i(j-1)}] + t_{ij}$.
- For finite intermediate storage (FIS): $c_{ij} = \max[c_{(i-1)j}, c_{i(j-1)}, c_{(i-z_j-1)(j+1)} - t_{ij}] + t_{ij}$.
- For no intermediate storage (NIS): $c_{ij} = \max[c_{(i-1)j}, c_{i(j-1)}, c_{(i-1)(j+1)} - t_{ij}] + t_{ij}$.

Therefore, we can simply achieve δ for different makespan flowshop problems. Moreover, k in the function $\exp(-\delta/k\beta_{\mathrm{n}})$ is a Boltzman constant, which at the beginning of SA can be estimated by $k = [c_{\max}(s) - c_{\max}(s_0)]/\ln p_0/\beta_0$, where p_0 is the initial value of the acceptance probability.

However, Matsuo *et al.* (1989) and Ogbu and Smith (1990) showed that a SA approach with an acceptance probability function which is independent of the change in the objective function value, δ, provided as good quality solutions as the conventional (Metropolis) scheme. So the acceptance probability function, for $\delta > 0$, i.e. accepting worse solution, can also be given by some functions such as $AP(k) = p_0(rf)^{k-1}$, where k is the stage, p_0 is the initial value of the acceptance probability at stage $k = 1$ and $rf < 1$ is the reducing factor.

Additionally, to compare different SA procedures, there are some proposed performance measures. For instance, one is optimality of the final solution: $(c_{\max}(T) - c^*_{\max})/c^*_{\max}$, where T is the total number of trials and $c_{\max}(T)$ is the makespan in the final solution. Another performance measure can be proposed for the effectiveness of the whole process such as:

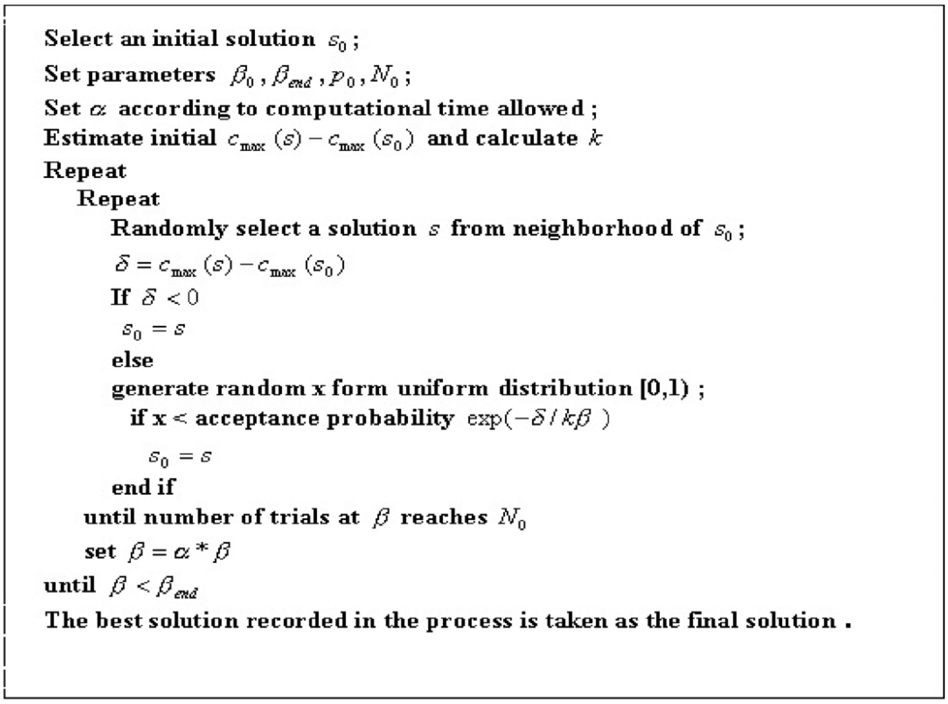
$$\int_0^T [(c_{\max}(t) - c^*_{\max})/(c_{\max}(0) - c^*_{\max})]\mathrm{d}t$$

where $c_{\max}(t)$ is the makespan of the best solution found by t trials. The effectiveness given above is the area under the curve of optimality against number of trials between 0 and T trials.

The SA approach and its implementation for a flowshop problem were discussed in this section. Figure 1 shows the SA procedure for makespan flowshop problem with the parameters and details described.

4.2 *Genetic algorithm approach*

Genetic Algorithms (GAs) were described by Holland (1975) as being based on the model of biological evolution process. He had a double aim: to improve the understanding of natural adaptation process, and to design artificial systems with properties similar to natural systems (Goldberg 1988). Goldberg described GA as a search algorithm for optimization; and Davis (1991) also applied GAs. During the last decade GAs have been widely applied to many optimization fields such as combinatorial optimization problems (for instance in TSP and sequencing and scheduling problems: Jog *et al.* 1989, Starkweather *et al.* 1991, Ulder *et al.* 1991, Cleveland and



```
Select an initial solution $s_0$ ;
Set parameters $\beta_0$ , $\beta_{end}$ , $p_0$ , $N_0$ ;
Set $\alpha$ according to computational time allowed ;
Estimate initial $c_{\max}(s) - c_{\max}(s_0)$ and calculate $k$
Repeat
  Repeat
    Randomly select a solution $s$ from neighborhood of $s_0$ ;
    $\delta = c_{\max}(s) - c_{\max}(s_0)$
    If $\delta < 0$
     $s_0 = s$
    else
    generate random x form uniform distribution [0,1) ;
      if x < acceptance probability $\exp(-\delta / k\beta)$
        $s_0 = s$
    end if
   until number of trials at $\beta$ reaches $N_0$
   set $\beta = \alpha * \beta$
until $\beta < \beta_{end}$
The best solution recorded in the process is taken as the final solution .
```

Figure 1. The framework of described SA.

Smith 1989, Bagchi *et al*. 1991, Gabbert *et al*. 1991, Nakano and Yamada 1991, Fox and McMahon 1991, Syswerda and Palmucci 1991, Ishibuchi 1994, etc.).

Implementation of GAs for a flowshop-scheduling problem is seen in many papers (e.g. Syswerda 1991, Reeves 1995, Chen *et al*. 1995, Murata *et al*. 1996, Reeves and Yamada 1998, Ponnambalam *et al*. 2001, Wang and Zheng 2003, Ruiz *et al*. 2003). The major one was proposed by Reeves, who compared the performance of SA and GA for flowshop test problems ranging from 20 jobs and five machines to 500 jobs and 20 machines. The main conclusion was that SA out performed GA in most tests where the numbers of jobs was 50 and fewer; on the other hand, GA provided mostly superior solution to those obtained by SA for large problems. Murata *et al*. (1996) showed similar findings and also compared various crossover and mutation operators and showed that the two-point crossover and shift mutation operators are effective for a flowshop problem. They also considered two hybrid variants of the GA, namely genetic local search and genetic simulated annealing algorithms, and showed high performance for the hybrid models. Bagchi (1999) listed several earliest versions at the GA and reported improvement in performance. Lee and Shaw (2000) used GA with edge recombination crossover operator (as seen in Whitley *et al*. 1989, Lee *et al*. 1997) and obtained good quality solutions by GA. They also used a hybrid neural network and GA algorithm and showed a good performance.

The GA approach for solving flowshop problem, usually uses integer representation (sometimes referred to as a path representation), in which chromosomes are vectors of job indices, which represent sequences of jobs. This means that if a job is in position i of chromosome then it is in the ith position of that sequence.

The search space is comprised of all current chromosomes in a population with a fixed number of chromosomes such as N. N is referred to as the population size. The initial population can consist of N randomly generated sequences or can be obtained by previously described constructive heuristics. The fitness function of a chromosome reflects the efficacy of the makespan that corresponds to the sequence of jobs it represents (for makespan problems). Simply, a fitness value assigned to a string (chromosome) i can be proportional to $f_i = r_i / \sum_j r_j$, where r_i is the reciprocal of the makespan of chromosome i in the population. By randomly selecting chromosomes of current population (usually two parents) with a probability proportional to their fitness value and then mating them, an offspring that inherits genetic traits from parents can be constructed. The population size is maintained constant by the removal of one chromosome every time a new offspring is introduced to the population. This removed chromosome can be selected randomly or based on its fitness value. This simulates a real scenario where chromosomes representing solutions with good makespan have a higher probability of remaining in the population and of being selected for further mating. Mating is done by crossover and mutation operators to form the next generation. The main purpose of crossover is to exchange information between randomly selected parent chromosomes with the aim of producing better offspring and intending to search for better genes. Two selected parents for mating are copied as they are with probability $1 - p_c$ and with probability p_c, called crossover probability, a random point (or points) is selected and sections from different parents are joined to create a new chromosome. The difficulty in applying crossover operators to chromosomes that are non-binary, as in the integer representation of flowshop problem, is that recombination usually results in an infeasible string in which jobs appear twice or not at all in the offspring.

For example, in a standard GA a single-point crossover (called 1X operator) between two parents is carried out by choosing a number k at random between 1 and $l-1$, where l is the length of the strings (chromosomes). Two new strings are created by swapping all characters from position $k+1$ to l. This crossover operator obviously can result in such infeasible strings.

However, many crossover operators and modifications are suggested and demonstrated to be appropriate for a flowshop problem in much research, including two-point crossover (2X), linear order crossover (LOX), partially mapped crossover (PMX), cycle crossover (CX), C1 operator, NABEL operator, edge recombination crossover operator, a few multi-parent crossover (MPX) and longest common subsequence crossover (LCSX). Hereby we briefly describe some of these crossover operators:

- 2X. In case of a two-point crossover, a parent chromosome is cut at two random points. The jobs on the extreme ends of the two cuts are inherited by the child, and in the exact locations and order in which they appear in the chromosome. The central section in the child is then filled with outstanding jobs, and in the order of their appearance in the other parent.
- LOX. First, two cutting sites of the parents, e.g. (2 6 4 7 3 5 8 9 1) and (4 5 2 1 8 7 6 9 3), are chosen randomly, e.g. 2 and 5. Second, the symbols that appear in the cross-section of the first parent (the area situated between the two cutting sites) are removed from the second parent leaving some ‘holes’, i.e. (H 5| 2 18| H 6 9 H) and (H 6| 4 7 3 |5 H 9 H). Then the holes are slid from the extremities towards the centre until they reach the crossover section, i.e. (5 2| H H H| 1 8 6 9) and (6 4| HH H|7 3 5 9). Finally, the cross-section is substituted with that of the corresponding parent to obtain the children, i.e. (5 2| 47 3| 1 8 6 9) and (6 4| 2 1 8| 7 3 5 9). It was considered that LOX could preserve as much as possible of the relative positions between the genes and the absolute positions relative to the extremities of the chromosome.
- PMX. The PMX operator may be the most popular crossover operator for operating the permutation. It chooses first two crossover points and exchanges the subsection of the parents between the two points, and then it fills the chromosomes by partial mapping. Considering the above two parents, if the two crossover points are 3 and 7, then the children will be (2 3 4| 1 8 7 6| 9 5) and (4 1 2| 7 3 5 8| 9 6). It has been demonstrated that PMX satisfies the fundamental properties enunciated by Holland, namely that it behaves in such a way that the best schemata reproduce themselves better than the others.
- C1 Operator. It chooses first one crossover point randomly and then takes the subsection of the first parent before the crossover point, and then fills up the chromosome by taking in order, each ‘legitimate’ element from the second parent. Considering the above two parents, if the crossover point is 3, then the children will be (2 6 4| 5 1 8 7 9 3) and (4 5 2| 6 7 3 8 9 1). The rationale for this crossover operator is that it preserves the absolute positions taken from the first parent and the relative positions from the second parent, so it was expected that this would provide enough scope for modification of the chromosome without excessively disrupting it.
- NABEL operator. It is created by the non-Abel group theory. If the parents are a (1) and b (1), then the children are c (1) = a (b (1) and d (1)) = b (a (1)),

$i = 1, 2, \ldots, n$. Considering these two parents, the children will be (7 3 6 2 9 8 5 1 4) and (5 7 1 6 2 8 9 3 4). This operator is very easy to implement.

- Edge recombination crossover operator. It constructs an offspring by exclusively using edges present in the parent strings, thus inheriting as much information as possible from the parent structures. To store all the connections from two parents, an edge map is created. For example, for the two parent strings given below the following edge map has been created.

Parent 1: 1 2 3 4 5 6 7 8 9 10
Parent 2: 9 4 10 6 5 2 8 1 3 7

Edge map:

Job type	1	2	3	4	5	6	7	8	9	10
Parent 1		1	2	3	4	5	6	7	8	9
Parent 2	8	5	1	9	6	10	3	2	4	

In the above edge map, the blank cell indicates that it has been left without any continuing edges. To use the edge map, we always choose the job type that has the fewest edges since it has the higher probability of being isolated. Suppose the child string is initialized with one of two final job types of parent strings. In our example, both job types 10 and 7 have two edges. Therefore, we make a random choice. Suppose that seven have been chosen randomly. Then, job type 7 is removed from the edge map. Next, candidate job types are 6 and 3, both of which have the same number of edges. Suppose that six is chosen randomly. Both candidate job types 5 and 10 have two unused edges. Then, suppose that 10 is chosen randomly. Next, candidate job type 9 is chosen because it has fewer numbers of edges. Job type 8 is chosen next since it is the only candidate. Continue this process until we get a complete sequence. The resulting sequence is composed of edges entirely from the two parent strings.

Child 1: 1 3 4 5 2 8 9 10 6 7

- MPX. It is used to increase the expected convergence and power of the recombination operator, a three-parent crossover can be performed. The way in which to do this is similar to that of the two-parent crossover; however, up to three different elements may be inserted at each stage instead of two. There would now be eight cases instead of four and the complexity and size of the algorithm would be substantially increased. To reduce the complexity and code required for a three-parent crossover, a more simplified approach was proposed. The three-parent crossover was split into two 'two-parent' crossovers. First, parents 1 and 2 were crossed, and then the result was crossed with parent 3 to obtain an offspring. Another approach could be to cross parent 1 with parent 2, and parent 1 and parent 3 separately, and then to cross the results of the first two crossings. Given that a three-parent crossover can be defined, and which incorporates two 'two-parent' crossovers, a MPX can also be classified in a similar way, by performing, say, *P*,1 two-parent crossovers, where *P* is the total number of population members. The algorithm for an MPX is shown in figure 2. The main idea or emphasis behind the MPX is that

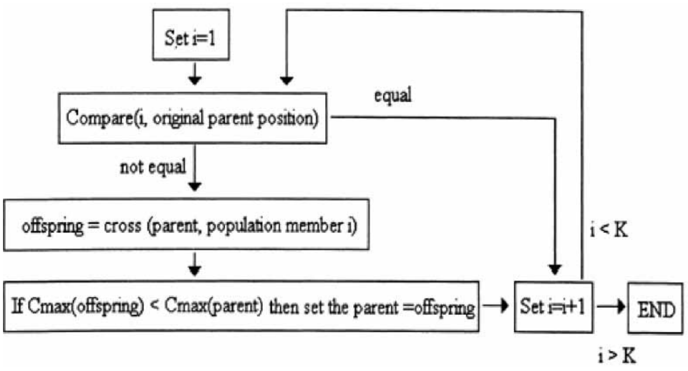

Set i=1
Compare(i, original parent position)
equal
not equal
offspring = cross (parent, population member i)
If Cmax(offspring) < Cmax(parent) then set the parent =offspring
Set i=i+1
END
i < K
i > K

Figure 2. MPX algorithm.

all population member information is used and exploited so that if any improvement is made, it is propagated throughout the whole population.

- LCSX. Iyer and Saxena (2002) proposed this operator to preserve that part of the genetic code of the parent chromosome which is responsible for their relatively high fitness and at the same time provide freedom to search a greater part of the domain for good strings. They suggested that the important information in chromosome is the relative position of jobs and used an operator. That preserver is only the longest common subsequence (LCS) in the parents. The crossover operator first considers the LCS as they are in the parents (which can be found using dynamic programming in polynomial time). Then elements not belonging to LCS are swapped in the following way. The first element of the parent 2 which is not present in the longest common subsequence is copied at the first available place in the child 1 and the element available at that position in the parent 1 is copied at the first available position in child 2. This procedure is continued to get the two children.

In addition, to crossover, mutation operators are implemented in GA approaches. Mutation is a way of enlarging the search space. It acts to prevent the selection and crossover from focusing on a narrow area of the search space or from the GA getting stuck in a local optimum. For each child obtained from crossover, the mutation operator is applied independently with a small probability, called mutation probability. A number of classical re-ordering operators exist and have been successfully implemented in other solution techniques such as SA and TS, as we discussed some of them in SA section. They also can and have already been used as mutation operators in GA approaches. Some of the most famous of them are as follows (we reviewed a more complete list of them in the SA section):

- Exchange/swap: two distinct elements are randomly selected and exchanged.
- Reversion: reverse a chosen subsequence at the sequence.
- Transport: choose a subsequence and insert it somewhere else in the sequence.
- Insert: choose an element and then insert somewhere else after removing that.

The genetic approach and its implementation for a flowshop problem were discussed in this section. Moreover, figure 3 shows the GA algorithm in a general manner, which can be used with described details.

4.3 *Tabu search approach*

The tabu search procedure were proposed and discussed by Glover (1986, 1989), Hansen (1986), Glover *et al.* (1993) and Glover and Laguna (1997). This metaheuristic approach, which has been applied to solve different combinatorial optimization problems, starts with an initial solution and then applies a move mechanism to search the neighbourhood of the current solution to choose the most appropriate one. A neighbour (a corresponding move of current solution) is admissible if it is not tabu or if an aspiration criterion (such as allowing all moves that lead to a neighbour with better objective function than encountered so far) is fulfilled. To use the information about search history, selected moves are stored in a data structure called tabu list. This list contains l elements at a time and once a move is entered the oldest one is deleted, i.e. the selected move is put in the tabu list and remains there for the next l iterations. This parameter, l, is called tabu list size. To determine this parameter, there are currently almost three alternatives: to determine and set constant value, choose randomly from a given range and dynamically changing through certain adjustments. Despite the simplicity of TS, it is still an art defining neighbourhood, searching among neighbours, setting tabu list size etc. Construction of such components influences the algorithm performance, speed of convergence, running time, etc.

Several TS-based approaches have been proposed by different researchers for a flowshop problem, including Taillard (1990), Reeves (1993), Mocellin (1995),

```
The GA procedure
  initialize;
    determine a population size, $n$;
    generation = 0;
    generate an initial population, and assign it to an old population, old_pop;
    compute the fitness of each member in the old_pop;

  repeat;
    repeat;
      stochastically select two parents with probability
        proportional to their fitness; /*reproduction*/

      randomly select a crossover point and
        perform crossover on two parents with probability $P_c$
        to generate two offspring; /*crossover*/

      mutate each bit of each offspring with probability $P_m$; /*mutation*/
      place the two offspring in a new population, new_pop;

    until the new population is filled with $n$ individuals;
    generation = generation + 1;
    old_pop = new_pop;
    compute the fitness of each member in the old_pop;

  until gen $\geq$ max_gen or other termination criteria are met; /*termination*/
    end.
```

Figure 3. General GA procedure.

Nowicki and Smutnicki (1996), Ben-Daya and Al-Fawzan (1998), Moccellin and dos Santos (2000), and the famous SPIRIT algorithm proposed by Widmer and Hertz (1989). Nowicki and Smutnicki (1996) used block properties to explore different sequences that ensure that a considerable number of moves cannot be effective and therefore should be ignored. Ben-Daya and Al-Fawzan (1998) implemented a tabu search algorithm with some extra features such as intensification and diversification schemes that provide better moves in the tabu search process. The algorithm proposed provides similar results as the TS of Taillard (1990) and slightly better results than the SA of Ogbu and Smith (1990). Moccellin and dos Santos (2000) presented a hybrid tabu search-simulated annealing heuristic that is compared with simple tabu search and simple simulated annealing implementations from the same authors, showing advantages for the hybrid approach.

Implementing block properties, Solimanpur *et al.* (2003) proposed a neuro-tabu search, and Grabowski and Wodecki (2004) a fast tabu search approach, both for permutation flowshop problem with makespan criterion. Here, we briefly overview some definitions and properties.

As given before, it is well known in the literature that $c_{\max}(\pi)$, i.e. the makespan of the processing order given by π, can be found by:

$$c_{\max}(\pi) = \max_{1\le t_1\le t_2\le\cdots\le t_{m-1}\le n}\left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m}\right)$$

where $\pi(j)$ is the jth element of permutation π. In other words, a makespan associated with π can be considered as the longest (critical) path from node (1, 1) to node (m, n) in a grid graph such as given in figure 4. Each path in this graph is composed of horizontal and vertical sub-paths. Each horizontal sub-path is called a block. In other words, if we consider path $u=(u_1, u_2, \dots, u_{m-1})$ such that:

$$c_{\max}(\pi) = c(\pi, u) = \sum_{j=1}^{u_1} p_{\pi(j)1} + \sum_{j=u_1}^{u_2} p_{\pi(j)2} + \cdots + \sum_{j=u_{m-1}}^{n} p_{\pi(j)m}$$

then

$$B_k = (\pi(u_{k-1}), \pi(u_{k-1}+1), \dots, \pi(u_k))$$

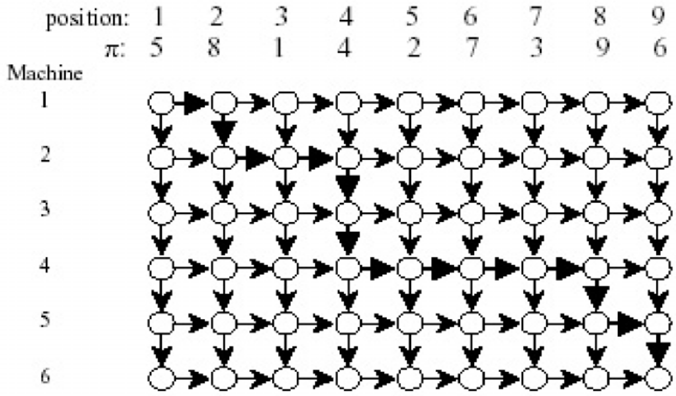


Figure 4. Grid graph of a flowshop scheduling problem with six machines and nine jobs.

which is a critical path respect to π a sequence of jobs is called kth block in π, which is dependent on π but for simplicity, it is represented as B_k.

As discussed above, here again we can consider many types of reordering operators such as exchanging two adjacent or none adjacent jobs or removing a job from its place in sequence and insert that in another position. Taillard (1990) experimentally and Grabowski (1997, 1982) using some theorems suggested that the latter, move $v=(x,y)$, which is removing job placed at position x and inserting that to position y, is more efficient and effective than others. Moreover, as the total number of moves to be evaluated in each iteration is $(N-1)^2$ (and high for large problems), to reduce the number of moves some reduction policies based on block properties and observations can be used. For example, an elementary property and corollary (Grabowski 1979, 1982) roughly speaking, shows that moving jobs in blocks, with the discussed move operators cannot make a better makespan. Consequently, they are not interesting moves and the number of moves can be reduced. Also, experimental findings of Nowicki and Smutnicki (1996) showed that few insertions in the adjacent blocks can be more promising than other positions, which can be implemented for further reduction of moves such as what Solimanpur *et al.* (2003) has used in their neuro-tabu search heuristic for a flowshop problem. Additionally, for instance, Grabowski and Wodecki (2003), to reduce further computational requirements, proposed to use lower bounds on the makespan instead of computing the makespan, for selecting the best solution to construct a very fast tabu search algorithm for a flowshop problem with makespan criterion.

This section discussed tabu search approaches for a flowshop problem and briefly described some reduction properties. Moreover, figure 5 shows a very generic tabu search heuristic for a given starting solution S and a tabu criterion that is represented by the object tabu memory.

4.4 *Ant colony approach*

The ant colony system (ACS) first proposed by Dorigo and Gambardella (1977) is one of the most recent and hopeful metaheuristics for combinatorial optimization problems. Ant colony optimization (ACO) simulates the collective foraging habits of ants, venturing out for food and bringing it back to the nest. Real ants are capable of finding the shortest path from a food source to their nest without using visual cues as

```
TabuSearch($s$, TabuMemory):

initialize TabuMemory
while ( stopping criterion not fulfilled )
  $s'$ = BestAdmissibleNeighbour( $s$, Neighbourhood, TabuMemory )
  TabuMemory.add( move($s, s'$) )
  $s = s'$
  TabuMemory.add( $s$ )
  if ( escape triggered by TabuMemory )
      perform a diversifying move
```

Figure 5. Generic Tabu Search(TS) heuristic.

they have poor vision. They communicate information concerning food sources via an aromatic essence. This chemical substance deposited by ants as they travel is called a pheromone. A greater amount of pheromone on the path gives an ant a stronger stimulation and thus a higher probability to follow it. They essentially move randomly, but when they encounter a pheromone trail, they decide (which is dependent on the amount of pheromone on the potential path) whether or not to follow it, and if they do so, they deposit their own pheromone on the trail, which reinforces the path. Since ants passing through a food source by a shorter path will come back to the nest sooner than ants via longer paths, the shorter path will have a higher traffic density and therefore will make the quantity of pheromone laid down on the shorter path grow faster. Moreover, with time, the pheromone evaporates and as the ants taking the shorter path will return to nest first with food, the shorter path has the most pheromone. Consequently, the shorter path becomes a more attractive alternative for other ants. However, there is always a probability that an ant will not follow a well-marked pheromone trail. This probability (although perhaps small) allows for exploration of other trails.

The described foraging behaviour of real ant colonies can be used to solve combinatorial optimization problems by simulation: The objective value corresponding to the quality of the food source, artificial ants searching the solution space simulating real ants searching their environment, and an adaptive memory corresponding to the pheromone trail. In addition, artificial ants are equipped with a local heuristic function to guide their search through the set of feasible solutions.

ACO has been applied to solve different types of combinatorial optimization problems including the TSP (Dorigo and Gambardella 1997), QAP (Quadratic Assignment Problem) (Gambardella *et al.* 1999, Solimanpur *et al.* 2003), the Scheduling Problem (Colorni *et al.* 1994), VRP (Vehicle Routing Problem) (Bullnheimer and Hartt 1999), the graph colouring problem (Costa and Hertz 1997), the Partitioning Problem (Kuntz *et al.* 1994, 1997) and the telecommunications net works problem (Schoorderwoerd *et al.* 1997).

There are a few research of implementing ACO for flowshop problem. T'kindt *et al.* (2003) proposed an ACO algorithm to solve a two-machine bicriteria flowshop problem. They proposed their algorithm to minimize total completion time. Rajendran and Ziegler (2004) considered the problem of scheduling in permutation flowshops with the objective of minimizing the makespan, followed by the consideration of minimization of total flow time of jobs. They proposed two ant colony optimization algorithms. The first algorithm extends the ideas of the ant colony algorithm by Stützle's so-called max–min ant system (MMAS), by incorporating the summation rule suggested by Merkle and Middendorf (2000) and a newly proposed local search technique. The first papers of applying ACS for the $n/m/p/c_{\max}$ or equivalently $F/prmu/c_{\max}$ (the problem by which we also discussed other approaches) was done by Stützle (1998) and Ying and Liao (2003). The latter's computational experiments, which are conducted on the well-known benchmark problem set of Taillard (1993), showed that the ACS approach is an effective metaheuristic for this problem. Hereby, we briefly discuss the ACS approach for $n/m/p/c_{\max}$ problem based on this paper.

First, we describe the representation of permutation flowshop problem with makespan criterion in ACS. This problem can be represented in ACS by a disjunctive graph such as $G=(O,C,D)$, where O is a set of nodes, C is a set of conjunctive directed (solid) arcs and D is a set of disjunctive undirected (broken) arcs.

The set O stands for all of the processing operations O_{ij} acted upon the n jobs, C corresponds to the precedence relationships between the processing operations of a single job and D is the machine constraint of operations belonging to different jobs. In addition, there are a nest N and a food source F, which are dummy nodes. Node N has conjunctive directed arcs emanating to the first operations of the n jobs, and F has conjunctive directed arcs coming from all the final operations. We have therefore $(nm+2)$ nodes, where all the nodes of the same machine are pairwise connected in both directions.

Figure 6 shows an instance of $3/4/p/c_{\max}$ (Ying and Liao 2003). The disjunctive undirected arcs D from m cliques, one for each machine, can determine the processing order of operation on the machine; and while in permutation problems all jobs have the same ordering sequence on each machine and it is enough to find the first clique's sequence.

In an ACS approach, first a set of artificial ants is initially positioned on starting nodes according to some initialization rule (e.g. randomly). Each ant constructs a tour that is a feasible solution to the problem $(n/m/p/c_{\max})$. This is done by repeatedly applying a stochastic greedy rule, which is called the state transition rule. According to what Dorigo and Gambadella (1997) proposed and Ying and Liao (2003) used, we describe this rule. When building a tour in ACS, an ant k at the current position of node i chooses the next node j to move to by applying the state transition rule given by the following equation:

$$j = \begin{cases} \arg\max\limits_{u \in S_k(i)} \{[\tau(i,u)][\eta(i,u)]^{\beta}\} & \text{if} \quad q \le q_0 \\ J & \text{otherwise} \end{cases}. \tag{1}$$

where $\tau(i,u)$ is the pheromone trail of edge (i,u), the heuristic desirability $\eta(i,u) = 1/\delta(i,u)$ is the inverse of the length from node i to node $u(\delta(i,u))$, $S_k(i)$ is the set of nodes that remain to be visited by ant k positioned on node i (to make the solution feasible). In addition, β is a parameter that determines the relative importance of pheromone versus distance ($\beta > 0$) and is a random number uniformly distributed in (0, 1), and q_0 is a parameter ($0 \le q_0 \le 1$) which determines the relative importance of exploitation versus exploration. In addition, J is a random variable that gives the probability with which ant k in node i chooses to move to node j that is selected

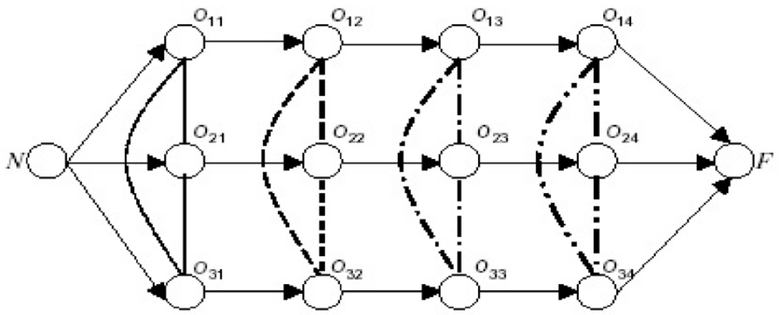


Figure 6. An instance of $3/4/P/c_{\max}$. Legend: (· · ·) arcs for machine 1, (- - -) arcs for machine 2, (- · · -) arcs for machine 3, and (- · · · -) arcs for machine 4.

according to the probability distribution, called a random-proportional rule, given in the following equation:

$$p_k(i,j) = \begin{cases} \frac{[\tau(i,j)][\eta(i,j)]^\beta}{\sum_{u \in S_k(i)} [\tau(i,u)][\eta(i,u)]^\beta} & \text{if} \quad j \in S_k(i) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

When building their tours, the chosen edges are guided by both heuristic information and pheromone information. The state transition rule resulting from equations (1) and (2) favour the choice of nodes connected by shorter edges with a greater amount of pheromone. Every time an ant in node i has to choose a node j to move to, it samples a random number q. If $q \le q_0$, then the best edge (according to equation (1)) is chosen (exploitation), otherwise an edge is chosen according to equation (2) (biased exploration).

With revising the slope index of Palmer's idea (described in section 3.1), Ying and Liao (2003) define the length between node i and node u of the first clique by:

$$\delta(i,u) = 1/\eta(i,u)$$

where

$$\eta(i,u) = (m-1)p_{u,m} + (m-3)p_{u,m-1} + \cdots - (m-3)p_{u,2} - (m-1)p_{u,1} \\ - \min_u\{\eta(i,u)\} + 1 \quad (i \neq u).$$

While constructing its tour, an ant also updates the amount of pheromone on visited edge by applying the local updating rule as follows:

$$\tau(i,j) := (1-\rho)\tau(i,j) + \rho\tau_0$$

where τ_0 is the initial pheromone level and $0 < \rho < 1$ is the pheromone evaporating parameter. The effect of local updating rule is to make the desirability of edges change dynamically to shuffle the tour. If ants explore different paths, then there is a higher probability that one of them will find an improving solution than they all search in a narrow neighbourhood of the previous best tour. Every time an ant constructs a path, the local updating rule will make its visit edges' pheromone diminish and become less attractive. Hence, the nodes in one ant's tour will be chosen with a lower probability in building other ants' tours. Consequently, ants will favour the exploration of edges not yet visited and prevent converging to a common path.

Finally, once all ants have terminated their tours, pheromone trails on the edges are modified again by applying the global updating rule. To make the search move directed, this rule is intended to provide a greater amount of pheromone to shorter tours and reinforce them. Therefore, only the globally best ant that found the best solution (i.e. the shortest best) up to current iterations of the algorithm is permitted to deposit pheromone. The pheromone level can be modified by:

$$\tau(i,j) := (1-\alpha)\tau(i,j) + \alpha\Delta\tau(i,j), \tag{3}$$

where

$$\Delta\tau(i,j) = \begin{cases} (L_{gb})^{-1} & \textit{if} \quad (i,j) \in \text{global best tour} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

In the equation (3), $\alpha (0 < \alpha < 1)$ is the pheromone evaporating parameter. Moreover, in equation (4), L_{gb} is the length of the globally best tour found up to the current iteration. Thus, equation (3) modifies the pheromone level to make search move directed by reinforcing shorter tours.

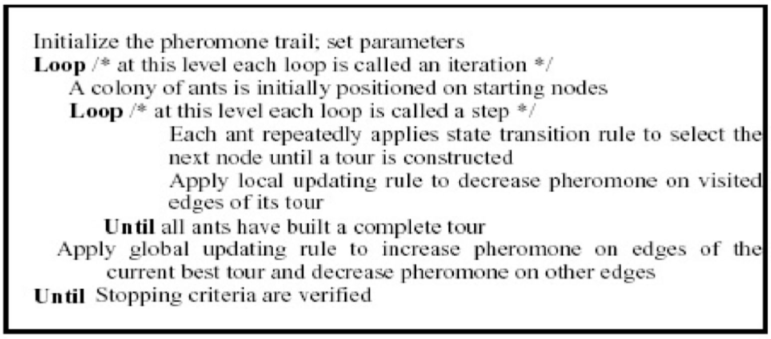
This section considered the most recent metaheuristic method, the ACS approach, for solving the flowshop permutation problem with makespan criterion. Figure 7 shows the generic ACS algorithm, which can be applied with described details.

5. Conclusions and future research

In this paper, we reviewed contributions in solving flowshop problem. We mainly considered flowshop problem with makespan criterion and surveyed some exact methods, constructive heuristics and developed improving metaheuristic and evolutionary approaches as well as some well-known prosperities and rules for this problem. In our survey, we considered much research from the early paper of Johnson (1954) to recent contributions of 2003 in ACS, TS, GA, etc. approaches.

In each part (especially in section 4), we tried to give a brief literature review by noting contributions and gave a glimpse of that approach before discussing the implementation for a flowshop problem. While some scheduling techniques have given clues for later research, in sections 2–4, unlike many other surveys, we also covered details of scheduling techniques as well as noting the contributions in that field. Moreover, section 1 gave a complete literature review of flowshop-related scheduling problems (with different assumptions) and contributions in solving such other aspects as well.

This paper can be used as a reference of past contributions (particularly in $n/m/p/c_{\max}$ or equivalently $F/prmu/c_{\max}$), for future research needs of improving and developing better approaches (such as improving operators and developing stronger properties and faster algorithms) to flowshop-related scheduling problems. Future research may also focus on creating and developing new ideas of implementing the discussed approaches for a flowshop problem as applied to the recent needs of real-world applications—such as web servers and e-commerce needs, IT-based requirement, Virtual Factories (VF) needs, Lot sizing and inventory control, Supply Chain Management (SCM), Assembly Line Balancing (ALB), Robotic Flowshop Scheduling (RFS) and other related emerging trends—in industries or



```
Initialize the pheromone trail; set parameters
Loop /* at this level each loop is called an iteration */
    A colony of ants is initially positioned on starting nodes
    Loop /* at this level each loop is called a step */
            Each ant repeatedly applies state transition rule to select the
            next node until a tour is constructed
            Apply local updating rule to decrease pheromone on visited
            edges of its tour
        Until all ants have built a complete tour
    Apply global updating rule to increase pheromone on edges of the
        current best tour and decrease pheromone on other edges
Until Stopping criteria are verified
```

Figure 7. The ACS algorithm.

even social needs (such as traffic applications) rather than theoretical aspects of flowshop-related scheduling problems. These new implementations may occur by recognizing similar characteristics of above-mentioned situations and flowshop-related scheduling problems.

References

Abadi, I.N.K., Hall, N.G. and Sriskandarajah, C., Minimizing cycle time in a blocking flowshop. *Oper. Res.*, 1996, **48**(1), 177–180.

Aldowaisan, T. and Allahverdi, A., New heuristics for *m*-machine no-wait flowshop to minimize total completion time. *Omega*, 2004, **32**, 345–352.

Aldowaisan, T. and Allahverdi, A., New heuristics for no-wait flowshops to minimize makespan. *Comput. Oper. Res.*, 2003, **30**, 1219–1231.

Aldowaisan, T. and Allahverdi, A., Total flowtime in no-wait flowshops with separated set-up times. *Comput. Oper. Res.*, 1988, **25**, 757–765.

Allahverdi, A., A new heuristic for *m*-machine flowshop-scheduling problem with bicriteria of makespan and maximum tardiness. *Comput. Oper. Res.*, 2004, **31**, 157–180.

Allahverdi, A., Scheduling in stochastic flowshops with independent set-up, processing and removal times. *Comput. Oper. Res.*, 1997, **24**, 955–960.

Allahverdi, A., Two-stage production scheduling with separated set-up times and stochastic breakdowns. *J. Oper. Res. Soc.*, 1995, **46**, 896–904.

Allahverdi, A. and Al-Anzi, F.S., Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. *Comput. Oper. Res.*, 2002, **29**, 971–994.

Allahverdi, A. and Aldowaisan, T., No-wait flowshops with bicriteria of makespan and maximum lateness. *Eur. J. Oper. Res.*, 2004, **152**, 132–147.

Allahverdi, A., Gupta, J.N.D. and Aldowaisan, T., A review of scheduling research involving set-up considerations. *Omega*, 1999, **27**, 219–239.

Andres, G., Albarracin, J.M., Tormo, G., Vicens, E. and Garcia-Sabater, J.P., Group technology in a hybrid flowshop environment: a case study. *Eur. J. Oper. Res.*, 2005, **167**, 272–281.

Baker, K.R., *Introduction to Sequencing and Scheduling*, 1974 (Wiley: New York).

Baker, K.R., Scheduling groups of jobs in the two-machine flowshop. *Math Comput. Model*, 1990, **13**, 29–36.

Baptiste, P. and Hguny, L.K., A branch-and-bound algorithm for the F/no-idle/$C_{\max}$, in *Proceedings of the International Conference on Industrial Engineering and Production Management*, IEPM'97, 1997, Vol. 1, pp. 429–438.

Ben-Daya, M. and Al-Fawzan, M., A tabu search approach for the flow shop scheduling problem. *Eur. J. Oper. Res.*, 1998, **109**, 88–95.

Bertolissi, E., Heuristic algorithm for scheduling in the no-wait flow-shop. *J. Mater. Process. Tech.*, 2000, **107**, 459–465.

Bonney, M.C. and Gundry, S.W., Solution to the constrained flowshop sequencing problem. *Operat. Res. Quart.*, 1976, **24**, 869–883.

Burdett, R.L. and Kozan, E., Evolutionary algorithms for flowshop sequencing with non-unique jobs. *Int. Trans. Oper. Res.*, 2000, **7**, 401–418.

Campbell, H.G., Dudek, R.A. and Smith, M.L., A heuristic algorithm of the *n*-job, *m*-machine sequencing problem. *Manag. Sci.*, 1970, **16**, B630–B637.

Cao, J. and Bedworth, D.C., Flow shop scheduling in serial multi-product processes with transfer and set-up times. *Int. J. Prod. Res.*, 1992, **30**, 1819–1830.

Chakravarthy, K. and Rajendran, C., A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization, *Prod. Plan. Contr.*, 1996, **10**, 707–714.

Chang, P.-C., Hsieh, J.-C. and Lin, S.G., The development of gradual-priority weighting approach for the multi-objective flowshop-scheduling problem. *Int. J. Prod. Econ.*, 2002, **79**, 171–183.

Cheng, T.C.E., Gupta, J.N.D. and Wang, G., A review of flowshop-scheduling research with set-up times. *Prod. Oper. Manag.*, 2000, **9**, 262–282.

Conway, R.W., Maxwell, W.L. and Miller, L.W., *Theory of Scheduling*, 1967 (Addison-Wesley: Reading, MA).

Corwin, B.D. and Esogbue, A.O., Two-machine flowshop-scheduling problems with sequence-dependent set-up times: a dynamic programming approach. *Naval Res. Logist. Quart.*, 1974, **21**, 515–524.

Daniels, R.L. and Chambers, R.J., Multi-objective flow shop scheduling. *Naval Res. Logist. Quart.*, 1990, **37**, 981–995.

Danneberg, D., Tautenhahn, T. and Werner, F., A comparison of heuristic algorithms for flow shop scheduling problems with set-up times and limited batch size, 1998. Preprint (Otto-von-Guericke University, Magdeburg).

Dannenbring, D.G., An evaluation of flow shop sequencing heuristics. *Manag. Sci.*, 1977, **23**, 1174–1182.

Dorigo, M. and Gambardella, L.M., Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evolut. Comput.*, 1997, **1**, 53–66.

Dudek, R.A. and Teuton Jr, O.F., Development of *m*-stage decision rule for scheduling *n* jobs through *m* machines. *Oper. Res.*, 1964, **12**, 471–497.

El-Bouri, A., A hybrid genetic algorithm for flowshop-scheduling. Available online at: http://www.umoncton.ca/cie/conferences/29thconf/29thICCIE/papers/paper043.PDF

Fink, A. and Vob, S., Applications of modern heuristic search methods to continuous flow-shop scheduling problems. Available online at: http://citeseer.ist.psu.edu/91064.html

Framinan, J.M., Leisten, R. and Rajendran, C., Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *Int. J. Prod. Res.*, 2003, **41**, 121–148.

Gangadharan, R. and Rajendran, C., A simulated annealing heuristic for scheduling in a flowshop with bicriteria. *Comput. Ind. Eng.*, 1994, **27**, 473–476.

Gangadharan, R. and Rajendran, C., Heuristic algorithms for scheduling in the no-wait flow-shop. *Int. J. Prod. Econ.*, 1993, **32**, 285–290.

Glover, F. and Laguna, M., *Tabu Search*, 1997 (Kluwer: Boston).

Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989 (Addison Wesley: Reading, MA).

Goldberg, D.E., *Genetic Algorithms*, 1988 (Addison Wesley: Reading, MA).

Gourgand, M., Grangeon, N. and Norre, S., A contribution to the stochastic flow shop scheduling problem. *Eur. J. Oper. Res.*, 2004, **31**, 1891–1909.

Grabowski, J. and Pempera, J., New block properties for the permutation flow-shop problem with application in TS. *J. Oper. Res. Soc.*, 2001, **52**, 210–220.

Grabowski, J. and Wodecki, M., A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Comput. Oper. Res.*, 2004, **31**, 1891–1909.

Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annal. Discret. Math.*, 1979, **5**, 287–326.

Guinet, A., Efficiency of reductions of job-shop to flow-shop problems. *Eur. J. Oper. Res.*, 2000, **125**, 469–485.

Gupta, J.N.D., A functional heuristic algorithm for the flow-shop scheduling problem. *Oper. Res.*, 1971, **22**, 39–47.

Gupta, J.N.D., Flowshop schedules with sequence-dependent set-up times. *J. Oper. Res. Soc.*, 1986, **29**, 206–219.

Gupta, J.N.D., Flowshop-scheduling with sequence-dependent set-up times, in *Proceedings of the ORSA/TIMS National Meeting*, Washington, DC, USA, 1988.

Gupta, J.N.D. and Darrow, W.P., Approximate schedules for the two-machine flowshop with sequence-dependent set-up times. *Indian J. Manag. Sys.*, 1985, **1**, 6–11.

Gupta, J.N.D. and Darrow, W.P., The two-machine sequence-dependent flowshop-scheduling problem. *Eur. J. Oper. Res.*, 1986, **24**, 439–446.

Gupta, J.N.D., Das, S.R. and Ghosh, S., Flowshop Scheduling with Sequence Dependent Setup Times, 1995. Working Paper (Department of Management, Ball State University, Muncie).

Gupta, J.N.D. and Dudek, R.A., Optimality criteria for flow shop schedules. *AIIE Trans.*, 1971, **3**, 199–205.

Gupta, J.N.D., Neppalli, V.R. and Werner, F., Minimizing total flow time in a two-machine flowshop problem with minimum makespan. *Int. J. Prod. Econ.*, 2001, **69**, 323–338.

Gupta, J.N.D., Strusevich, V.A. and Zwaneveld, C., Two-stage no-wait scheduling models with set-up and removal times. *Comput. Oper. Res.*, 1997, **24**, 1025–1031.

Gupta, J.N.D. and Tunc, E.A., Scheduling a two-stage hybrid flowshop with separable set-up and removal times. *Eur. J. Oper. Res.*, 1994, **77**, 415–428.

Hall, N.G. and Sriskandarajah, C., A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.*, 1996, **44**, 510–525.

Ham, I., Hitomi, K. and Yoshida, T., *Group Technology*, 1985 (Kluwer-Nijhoff: Boston).

Hitomi, K. and Ham, I., Operations scheduling for group technology applications. *CIRP Ann.*, 1976, **25**, 419–422.

Hitomi, K., Nakamura, N., Yoshida, T. and Okuda, K., An experimental investigation of group production scheduling, in *Proceedings of the 4th International Conference on Production Research*, Tokyo, Japan, 1977, pp. 608–617.

Ho, J.C. and Chang, Y.L., A new heuristic for the *n*-job, machine flow-shop problem. *Eur. J. Oper. Res.*, 1991, **52**, 194–202.

Ishibuchi, M., Misaki, S. and Tanaka, H., Modified simulated annealing for the flow shop sequencing problems. *Eur. J. Oper. Res.*, 1995, **81**, 388–398.

Ishibuchi, H. and Murata, H., Multi-objective genetic local search algorithm and its applications to flowshop-scheduling. *IEEE Trans. Sys. Man Cybern.*, 1998, **28**, 392–403.

Iyer, S.K. and Saxena, B., Improved genetic algorithm for the permutation flowshop-scheduling problem. *Comput. Oper. Res.*, 2004, **31**, 593–606.

Jin, Y., Okabe, T. and Sendhoff, B., Adapting weighted aggregation for multiobjective evolutionary strategies, in *Proceedings of the First Conference on Evolutionary Multi-Criterion Optimization*, 2001, pp. 96–110.

Johnson, S.M., Optimal two- and three-stage production schedules with set-up times included. *Naval Res. Logist. Quart.*, 1954, **1**, 61–68.

Khurana, K. and Bagga, P.C., Scheduling of job-block with deadline in nX2 flowshop problem with separated set-up times. *Indian J. Pure Appl. Math.*, 1985, **16**, 213–224.

Khurana, K. and Bagga, P.C., Minimizing the makespan in a two-machine flowshop with time lags and set-up conditions. *Z. Oper. Res.*, 1984, **28**, 163–174.

Kim, Y.D., Heuristics for a flowshop-scheduling problems minimizing mean tardiness. *J. Oper. Res. Soc.*, 1993, **44**, 19–28.

King, J.R. and Spachis, A.S., Heuristics for flow-shop scheduling. *Int. J. Prod. Res.*, 1980, **18**, 345–357.

Kirkpatrick, S., Gellat, C.D. and Vecchi, M.P., Optimization by simulated annealing. *Science*, 1983, **220**, 671–680.

Koulamas, C., A new constructive heuristic for a flowshop-scheduling problem. *Eur. J. Oper. Res.*, 1998, **105**, 66–71.

Koulamas, C. and Kyparisis, G.J., Concurrent flowshop-scheduling to minimize makespan. *Eur. J. Oper. Res.*, 2004, **156**, 524–529.

Laarhoven, V.P. J.M. and Aarts, E.H.L., *Simulated Annealing: Theory and Applications*, 1988 (Kluwer: Dordrecht).

Lee, C.-Y., Cheng, T.C.E. and Lin, B.M.T., Minimizing the makespan in the 3-machine assembly-type flowshop-scheduling problem. *Manag. Sci.*, 1993, **39**, 616–625.

Lee, I. and Shaw, M.J., A neural-net approach to real-time flow-shop sequencing. *Comput. Ind. Eng.*, 2000, **38**, 125–147.

Leisten, R., Flow-shop sequencing problems with limited buffer storage. *Int. J. Prod. Res.*, 1990, **28**, 2085–2100.

Li, S., A hybrid two-stage flowshop with part family, batch production, major and minor set-ups. *Eur. J. Oper. Res.*, 1997, **102**, 142–156.

Liu, J., Controlling the parameters in simulated annealing for flowshop-scheduling, in *Proceedings of the 14th International Conference of Production Research*, 1997, pp. 1478–1481.

Liu, J., The impact of neighbourhood size on the process of simulated annealing: computational experiments on the flowshop-scheduling problem. *Comput. Ind. Eng.* 1999, **37**, 285–288.

Logendran, R. and Sriskandarajah, C., Two-machine group scheduling problem with blocking and anticipatory set-ups. *Eur. J. Oper. Res.*, 1993, **69**, 467–481.

Lomnicki, Z.A., A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem. *Operat. Res. Quart.*, 1965, **16**, 89–100.

Merkle, D. and Middendorf, M., An ant algorithm with a new pheromone evaluation rule for total tardiness problems, in *Proceedings of the EvoWorkshops*, Springer, Berlin, 2000, pp. 287–296.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E., Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 1953, **21**, 1087–1092.

Moccellin, J.A.V., A new heuristic method for the permutation flow shop scheduling problem. *J. Oper. Res. Soc.*, 1995, **46**, 883–886.

Murata, T., Ishibuchi, H. and Tanaka, H., Multi-objective genetic algorithm and its applications to flowshop-scheduling. *Comput. Ind. Eng.* 1996, **30**, 957–968.

Nabeshima, I. and Maruyama, S., Two- and three-machine flowshop makespan scheduling problems with additional times separated from processing times. *J. Oper. Res. Soc.*, 1984, **27**, 348–366.

Nagar, A., Haddock, J. and Heragu, S., Multiple and bicriteria scheduling: a literature survey. *Eur. J. Oper. Res.*, 1995, **81**, 88–104.

Nagar, A., Heragu, S.S. and Haddock, J., A combined branch-and-bound and genetic algorithm based approach for a flowshop-scheduling problem. *Annal. Oper. Res.*, 1996, **63**, 397–414.

Nawaz, M., Enscore Jr, E. and Ham, I., A heuristic algorithm for the *m*-machine, *n*-job flowshop sequencing problem. *OMEGA Int. J. Manag. Sci.*, 1983, **11**, 91–95.

Negenman, E.G., Local search algorithms for the multiprocessor flow shop scheduling problem. *Eur. J. Oper. Res.*, 2001, **128**, 147–158.

Neppalli, V.R., Chen, C.L. and Gupta, J.N.D., Genetic algorithms for the two-stage bicriteria flowshop problem. *Eur. J. Oper. Res.*, 1996, **95**, 356–373.

Nowicki, E., The permutation flow shop with buffers: a tabu search approach. *Eur. J. Oper. Res.*, 1999, **116**, 205–219.

Nowicki, E. and Smutnicki, C., A fast tabu search algorithm for the permutation flow-shop problem. *Eur. J. Oper. Res.*, 1996, **91**, 160–175.

Ogbu, F.A. and Smith, D.K., Simulated annealing for the permutation flowshop problem. *Omega*, 1991, **19**, 64–67.

Ogbu, F.A. and Smith, D.K., The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem. *Comput. Oper. Res.*, 1990, **17**, 243–253.

Osman, I.H. and Potts, C.N., Simulated annealing for permutation flow-shop scheduling. *Omega*, 1989, **17**, 551–557.

Palmer, D.S., Sequencing jobs through a multistage process in the minimum total time: a quick method of obtaining a near optimum. *Oper. Res. Quart.*, 1965, **16**, 101–107.

Panwalker, S.S., Dudek, R.A. and Smith, M.L., Sequencing research and the industrial scheduling problem. In *Symposium on the Theory of Scheduling and its Applications*, edited by S.E. Elmaghraby, pp. 29–38, 1973 (Springer: New York).

Park, T. and Steudel, H.J., Analysis of heuristics for job sequencing in manufacturing flow line work-cells. *Comput. Ind. Eng.*, 1991, **20**, 129–140.

Pinedo, M., *Scheduling: Theory, Algorithms and Systems*, 2nd edn, 2002 (Prentice-Hall: Englewood Cliffs, NJ).

Ponnambalam, S.G., Aravindan, P. and Chandrasekaran, S., Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. *Prod. Plan. Contr.*, 2001, **12**, 335–344.

Popadirnitriou, C. and Kanellaksi, P.C., Flowshop-scheduling with limited temporary storage. *J. Assoc. Comput. Mach.*, 1980, **27**, 533–549.

Potts, C.N., Sevastjanov, S.V., Strusevich, V.A., Wassenhove, L.N. and Zwaneveld, C.M., The two-stage assembly scheduling problem: complexity and approximation. *Oper. Res.*, 1995, **43**, 346–355.

Potts, O.N., An adaptive branching rule for permutation flow-shop problem. *Eur. J. Oper. Res.*, 1980, **8**, 19–25.

Pour, H.D., A new heuristic for the *n*-job *m*-machine flow-shop problem. *Prod. Plan. Contr.*, 2001, **12**, 648–653.

Presman, E.L., Sethi, S.P., Zhang, H. and Bisi, A., Average cost optimal policy for a stochastic two-machine flowshop with limited work-in-process. *Nonlinear Analysis*, 2001, **47**, 5671–5678.

Proust, C., Gupta, J.N.D. and Deschamps, V., Flowshop-scheduling with set-up, processing and removal times separated. *Int. J. Prod. Res.*, 1991, **29**, 479–493.

Raaymakers, W.H.M. and Weijters, A.J.M.M., Makespan estimation in batch process industries: A comparison between regression analysis and neural networks. *Eur. J. Oper. Res.*, 2003, **145**, 14–30.

Rajagopalan, D. and Karimi, I.A., Scheduling in serial mixed storage multiproduct processes with transfer and set-up times. In *Foundations of Computer Aided Process Operations*, edited by G.V. Reklaitis, and H.D. Spriggs, pp. 679–686, 1987 (Elsevier: New York).

Rajendran, C., A heuristic for scheduling in flowshop and flowline-based manufacturing cell with multi-criteria. *Int. J. Prod. Res.*, 1994, **3**, 2541–2558.

Rajendran, C., Heuristics for scheduling in flowshop with multiple objectives. *Eur. J. Oper. Res.*, 1995, **82**, 540–555.

Rajendran, C., Two-stage flowshop-scheduling problem with bicriteria. *J. Oper. Res. Soc.*, 1992, **43**, 871–884.

Rajendran, C. and Ziegler, H., Heuristics for scheduling in a flowshop with set-up, processing, and removal times separated. *Prod. Plan. Contr.*, 1997, **8**, 568–576.

Rajendran, C. and Ziegler, H., Ant-colony algorithms for permutation flowshop-scheduling to minimize makespan/total flowtime of jobs. *Eur. J. Oper. Res.*, 2004, **155**, 426–438.

Ramanan, G.V. and Rajendran, C., Scheduling in Kanban-controlled flowshops to minimize makespan of containers. *Int. J. Adv. Manuf. Tech.*, 2003, **21**, 348–354.

Reddi, S.S. and Ramamoorthy, C.V., On the flowshop sequencing with no wait-in-process. *Operat. Res. Quart.*, 1972, **23**, 323–331.

Reeves, C., A genetic algorithm for flowshop sequencing. *Comput. Oper. Res.*, 1995, **22**, 5–13.

Reeves, C. and Yamada, T., Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evol. Comput.*, 1998, **6**, 45–60.

Reeves, C.R., Improving the efficiency of tabu search for machine scheduling problems. *J. Oper. Res. Soc.*, 1993, **44**, 375–382.

Rinnooy Kan, A.H.G., *Machine Scheduling Problems: Classification, Complexity and Computations*, 1976 (Martinus Nijhoff: The Hague).

Rios-Mercado, R.Z. and Bard, F.B., Computational experience with a branch-and-cut algorithm for flowshop-scheduling with set-ups. *Comput. Oper. Res.*, 1988, **25**, 351–366.

Rios-Mercado, R.Z. and Bard, F.B., *The Flowshop Scheduling Polyhedron with Setup Times.* Technical Report ORP96-07, 1996 (Graduate Program in Operations Research, University of Texas at Austin).

Rock, H., The three-machine no-wait flowshop is NP-complete. *J. Assoc. Comput. Mach.*, 1984, **31**, 336–345.

Ruiz, R. and Maroto, C., A comprehensive review and evaluation of permutation flowshop heuristics. *Eur. J. Oper. Res.*, 2005, **165**, 479–494.

Ruiz, R., Maroto, C. and Alcaraz, J., Solving the flowshop-scheduling problem with sequence-dependent set-up times using advanced metaheuristics. *Eur. J. Oper. Res.*, 2004 (in press).

Saadani, N.H., Guinet, A. and Moalla, M., A travelling salesman approach to solve the F/no-idle/$C_{\max}$ Problem. *Eur. J. Oper. Res.*, 2004 (in press).

Salhi, S., Designing tabu list size and aspiration criterion within tabu search methods. *Comput. Oper. Res.*, 2002, **29**, 67–86.

Sarin, S. and Lefoka, M., Scheduling heuristic for the *n*-job *m*-machine flow shop. *Omega*, 1993, **21**, 229–234.

Schaller, J.E., Gupta, J.N.D. and Vakharia, A.J., Group scheduling with sequence-dependent set-ups, in *Proceedings of the Annual Decision Sciences Institute Meeting, San Diego*, CA, 1997, pp. 1141–1143.

Simons, J.V., Heuristics in flow shop scheduling with sequence-dependent set-up times. *Omega*, 1992, **20**, 215–225.

Skorin-Kapov, J. and Vakharia, A.J., Scheduling a flow-line manufacturing cell: a tabu search approach. *Int. J. Prod. Res.*, 1993, **31**, 1721–1734.

Solimanpur, M., Vrat, P. and Shankar, R., A neuro-tabu search heuristic for the flow shop scheduling problem. *Comput. Oper. Res.*, 2004, **31**, 2151–2164.

Sotoskov, Y.N., Tautenhahn, T. and Werner, F., Heuristics for permutation flowshop-scheduling with batch set-up times. *OR Spektrum*, 1996, **18**, 67–80.

Sridhar, J. and Rajendran, C., A genetic algorithm for family and job scheduling a flowline based manufacturing cell. *Comput. Ind. Eng.*, 1994, **27**, 469–472.

Sridhar, J. and Rajendran, C., Scheduling in flowshop and cellular manufacturing systems with multiple objectives, a genetic algorithmic approach. *Prod. Plan. Contr.*, 1996, **7**, 374–382.

Stafford, E.F. and Tseng, F.T., On the Srikar–Ghosh MILP model for the N X M SDST flowshop problem. *Int. J. Prod. Res.*, 1990, **28**, 1817–1830.

Stafford, E.F. and Tseng, F.T., Two MILP models for the SSIST flowshop sequencing problem, paper presented at DSI National Meetings, 17–20 November 2001.

Stafford, E.F. and Tseng, T.F., Two models for a family of flowshop sequencing problems. *Eur. J. Oper. Res.*, 2002, **142**, 282–293.

Stinson, J.P., Smith, A.W., A heuristic programming procedure for sequencing the static flowshop. *Int. J. Prod. Res.*, 1982, **20**, 753–764.

Stützle, T., An ant approach for the flow shop problem. In *EUFIT'98*, Aaeban, Germany, pp. 1560–1564, 1998,

Sule, D.R., Sequencing *n* jobs on two machines with set-up, processing and removal times separated. *Naval Res. Logist. Quart.*, 1982, **29**, 517–519.

Sule, D.R. and Huang, K.Y., Sequence on two and three machines with set-up, processing and removal times separated. *Int. J. Prod. Res.*, 1983, **21**, 723–732.

Sung, C.S. and Min, J.I., Scheduling in a two-machine flowshop with batch processing machine (s) for earliness/tardiness measure under a common due date. *Eur. J. Oper. Res.*, 2001, **131**, 95–106.

Szwarc, W., Flowshop problems with time-lags. *Manag. Sci.*, 1983, **29**, 477–481.

Szwarc, W., The flowshop problem with time lags and separated set-up times. *ZOR-Z Oper. Res.*, 1986, **30**, B15–B22.

Szwarc, W. and Gupta, J.N.D., A flow-shop problem with sequence-dependent additive set-up times. *Naval Res. Log.*, 1987, **34**, 619–627.

T'kindt, V. and Billaut, J.-C., Multicriteria scheduling problems: a survey. *RAIRO Oper. Res.*, 2001, **35**, 143–163.

T'kindt, V., Gupta, J.N.D. and Billuat, J.-C., Two-machine flowshop-scheduling with a secondary criterion. *Comput. Oper. Res.*, 2003, **30**, 505–526.

T'kindt, V., Monmarch, N., Tercinet, F. and Laugt, D., An Ant Colony optimization algorithm to solve a (2) machine bicriteria flowshop-scheduling problem. *Eur. J. Oper. Res.*, 2002, **142**, 250–257.

Taillard, E., Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.*, 1993, **64**, 278–285.

Taillard, E., Some efficient heuristic methods for the flow-shop sequencing problem. *Eur. J. Oper. Res.*, 1990, **47**, 65–74.

Tanarv, V.S., Sotskov, Y.N. and Strusevitch, V.A., *Scheduling Theory Multi-stage System*, 1994 (Kluwer: Dordrecht).

Tang, C.S., Scheduling batches on parallel machines with major and minor set-ups. *Eur. J. Oper. Res.*, 1990, **46**, 28–37.

Townsend, W., Sequencing *n* jobs on *m* machines to minimize maximum tardiness: a branch-and-bound solution. *Manag. Sci.*, 1977, **23**, 1016–1019.

Tseng, F.T. and Stafford, E.F., Two MILP models for the N X M SDST flowshop sequencing problem. *Int. J. Prod. Res.*, 2001, 39, 1777–1809.

Turner, B. and Booth, D., Comparison of heuristics for flow shop sequencing. *OMEGA Int. J. Manag. Sci.*, 1987, **15**, 75–78.

Uskup, G. and Smith, S.B., A branch-and-bound for two-stage production sequencing. *Oper. Res.*, 1975, **23**, 118–136.

Vakharia, A.J. and Chang, Y.-L., A simulated annealing approach to scheduling a manufacturing cell. *Naval Res. Log.*, 1990a, **37**, 559–577.

Vakharia, A.J. and Chang, Y.-L., Model for the NxM SDST flowshop problem. *Int. J. Prod. Res.*, 1990b, **28**, 1817–1830.

Vakharia, A.J., Schaller, J.E. and Gupta, J.N.D., Designing and scheduling manufacturing cells, in *Proceedings of the INFORMS National Meeting*, New Orleans: LA, USA, 1995.

Wang, L. and Zheng, D.Z., An effective hybrid heuristic for flow shop scheduling. *Int. J. Adv. Manuf. Tech.*, 2003, **21**, 38–44.

Whitley, D., Starkweather, T. and D'Ann, F., Scheduling problems and traveling salesman, in *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, Arlington, VA, USA, 1989, pp. 133–140.

Widmer, M. and Hertz, A., A new heuristic method for the flow shop sequencing problem. *Eur. J. Oper. Res.*, 1989, **41**, 186–193.

Wismer, D.A., Solution of the flowshop sequencing problem with no intermediate queues. *Oper. Res.*, 1972, **20**, 689–697.

Wodecki, M. and Bozejko, W., Solving the flow shop problem by parallel simulated annealing. In *Parallel Processing and Applied Mathematics*, edited by R. Wyrzykowski, J. Dongarra, M. Paprzycki and J. Waasniewski, 4th International Conference, PPAM 2001, pp. 236–244, 2002. (Springer: Berlin).

Ying, K.C. and Liao, C.J., An ant colony system for permutation flow-shop sequencing. *Comput. Oper. Res.*, 2004, **31**, 791–801.

Yoshida, T. and Hitomi, K., Optimal two-stage production scheduling with set-up times separated. *AIIE Trans.*, 1979, **11**, 261–263.

Zdrzalka, S., *Two-Machine Flow Shop Scheduling Problem with Family Setup Times*. Report No. 16/95, 1995 (Institute of Engineering Cybernetics, Technical University of Wroclaw).

Zegordi, S.H., Itoh, K. and Enkawa, T., Minimizing makespan for flowshop-scheduling by combining simulated annealing with sequencing knowledge. *Eur. J. Oper. Res.*, 1995, **85**, 515–531.