A hybrid algorithm to minimize makespan for the permutation flow shop scheduling problem

Fardin Ahmadizar[*]
Department of Industrial Engineering, University of Kurdistan, Pasdaran Boulvard, Sanandaj, Iran

Farnaz Barzinpour
Department of Industrial Engineering, Iran University of Science and Technology, Narmak, Tehran 16846_13114, Iran
E-mail: barzinpour@iust.ac.ir

Received: 15-06-2010
Accepted: 27-10-2010

Abstract

This paper deals with the permutation flow shop scheduling problem. The objective is to minimize the maximum completion time, or makespan. To solve this problem which has been proved to be strongly NP-hard, a combination between an ant colony algorithm, a heuristic algorithm and a local search procedure is proposed and presented. The hybrid approach is to use artificial ants to construct solutions by applying a stochastic greedy rule based on the Gupta's heuristic and pheromone trails. A local search is then performed to improve the performance quality of constructed solutions. Once all ants have terminated their generations, the pheromone trails are modified according to a global updating rule. The proposed algorithm is applied to benchmark problems taken from the literature and compared with other metaheuristics. Computational experiments are given to demonstrate the superiority of the algorithm in the quality of solution and CPU time.

Keywords: Permutation flow shop; Makespan; Ant colony optimization; Gupta's heuristic; Local search.

1. Introduction

One of the most well-known scheduling problems is the classical flow shop problem, a multi-stage machine scheduling problem. This scheduling problem has some special cases such as the hybrid flow shop (see Refs. 1 and 2 for more details) and the permutation flow shop problems. The problem of sequencing a set of jobs on a set of machines in a permutation flow shop has been the subject of extensive research because of its applications in practice. For example, in a manufacturing or assembly setting in which jobs have to be processed on the same machines in the same sequence, if a material handling system transports the jobs from one machine to the next, then the same job sequence occurs on each machine, i.e., the environment is treated as a permutation flow shop[3]. Some exact algorithms have been proposed over the years that guarantee to obtain an optimal sequence to the permutation flow shop scheduling problem (PFSP) (e.g., see Refs. 4–7). Moreover, many heuristic algorithms have been developed for the PFSP (e.g., see Refs. 8–15). As the PFSP is a well-known NP-hard problem (Rinnooy Kan in 1976 proved[16] that the makespan minimization problem is NP-complete and the flowtime minimization one was proved[17] by Garey *et al*. in 1976), metaheuristics like tabu search (e.g., see Refs. 18–24), genetic algorithms (e.g., see Refs. 25–31), simulated annealing (e.g., see Refs. 32–35) and particle swarm optimization algorithms (e.g., see Refs. 36–40) have been used for solving the PFSP.

In recent years, ant colony optimization (ACO) algorithms have been used for solving the combinatorial optimization problems such as scheduling problems.

[*] Corresponding author. Tel./fax: +98-871-6660073; *E-mail: f.ahmadizar@uok.ac.ir*

The pioneering work has been done[41] by Dorigo in 1992, and an introduction to ACO algorithms has been dealt by Dorigo *et al*.[42] in 1996 and Dorigo and Gambardella[43] in 1997. An ACO algorithm is a population-based, cooperative search procedure derived from the foraging behavior of real ants. This behavior enables ants to find shortest paths between their nest and food sources. While walking from the nest to a food source and vice versa, they deposit pheromones on the ground. They can smell pheromone, and when choosing their path, they tend to choose, in probability, paths marked by strong pheromone concentrations. Mimicking the foraging behavior of real ants, ACO algorithms employ simple agents, called artificial ants, which search for good solutions to a given combinatorial optimization problem. An artificial ant constructs a complete solution by starting with a null solution and iteratively adding solution components until a complete solution is constructed. The solution construction process is stochastic and biased by the pheromone trails dynamically modified at run-time.

To solve the PFSP with the objective of minimizing the makespan, the first ACO algorithm which is a max–min ant system, called MMAS, has been proposed[44] by Stutzle in 1998. Rajendran and Ziegler in 2004 have developed[45] two ACO algorithms for the PFSP with the objective of minimizing the makespan/total flowtime of jobs; the first algorithm, called M-MMAS, has extended MMAS and the second one, called PACO, has developed M-MMAS. Moreover, two ACO algorithms have been proposed[46] by Rajendran and Ziegler in 2005 for minimizing the total flowtime. Ying and Liao in 2004 have proposed[47] an ant colony system, called ACS, for minimizing the makespan where a different representation of the pheromone trails is applied based on a disjunctive graph. An ACO algorithm has been developed[48] by Gajpal and Rajendran in 2006 with the objective of minimizing the completion-time variance of jobs. Furthermore, some researchers have proposed ACO algorithms for solving the PFSP with the objective of minimizing two or more criteria (e.g., see Refs. 49–52).

In this paper, we present an efficient ant colony algorithm (ACA) for the PFSP. In this algorithm, a solution is constructed by applying a pseudo-stochastic rule based on both the heuristic information calculated using the Gupta's heuristic[53] and the pheromone trails. An interesting local search is then performed to improve the constructed solution. Once all ants have terminated their generations, the pheromone trails are modified using a new global updating rule. The proposed hybrid algorithm is tested on a set of benchmark problems proposed[54] by Taillard in 1993, and then compared with other metaheuristics. We would like to emphasize that the goal of this paper is to propose a new ACO algorithm that can compete with well-known ACO algorithms available in the literature.

The rest of the paper is organized as follows. In Sec. 2, the problem statement is introduced. The proposed algorithm is described in Sec. 3. Section 4 provides computational results on the Taillard's benchmark problems as well as performance comparisons with other metaheuristics. Finally, the conclusions are presented in Sec. 5.

2. PFSP Formulation

The PFSP consists in scheduling N different jobs (1, 2, ..., N) with given processing times on a set of M machines (1, 2, ..., M), where the sequence of processing a job on all machines is identical. Each job has exactly one operation to be processed on each machine. It is assumed that each job can be processed on at most one machine at a time and that each machine can process at most one job at a time. Furthermore, preemption is not allowed, the jobs are available and ready for processing at time zero, and the setup times are sequence independent. A schedule of this type is called a permutation schedule and defined with a complete sequence of all jobs.

Assuming that each operation is to be processed as soon as possible, for a given sequence of jobs the completion times of the operations can be found as follows. Let P_{mj} and C_{mj} be, respectively, the processing time and the completion time of job j on machine m. In this study, the objective of minimizing the maximum completion time, or makespan, is considered. Given the job permutation {1, 2, ..., N }, C_{mj} (m = 1, …, M; j = 1, …, N) and makespan are then calculated as follows:

$$C_{m1} = \sum_{a=1}^{m} P_{a1}, \qquad m = 1, 2, ..., M, \tag{1}$$

$$C_{1j} = \sum_{b=1}^{j} P_{1b}, \qquad j = 1, 2, ..., N, \tag{2}$$

$$C_{mj} = \max\left\{C_{(m-1)j}, C_{m(j-1)}\right\} + P_{mj}, \quad m = 2,\dots,M,\ j = 2,\dots,N, \tag{3}$$

$$C_{\max} = C_{MN}. \tag{4}$$

3. Proposed Hybrid Algorithm

The main idea in ACO algorithms is to mimic the pheromone trails used by real ants searching for feed as a medium for communication and feedback. In the ACA proposed to solve the PSFP, each artificial ant starts with an empty sequence and chooses one of the jobs. Then, the ant iteratively appends an unscheduled job to the partial sequence until a complete solution is constructed. At each step, a job is chosen by applying a transition rule based on the Gupta's heuristic and the pheromone trails. In other words, each ant builds a tour, i.e., a feasible solution to the PSFP, by repeatedly applying a pseudo-stochastic rule. This definition for moving artificial ants has been used in most applications of ACO to scheduling problems. The performance quality of the constructed solution is then improved by means of a local search procedure. Finally, once all ants have terminated their tours, the pheromone trails are modified according to a global updating rule in order to make the search more directed.

The general structure of the proposed ACA is then represented as follows:

```
Step 1. Initialize the pheromone trails, and set
        parameters.
Step 2. While the termination condition is not met, do:
    2.1. For each ant in the colony do:
        (a) Construct a solution by repeatedly applying
            the transition rule;
        (b) Improve the solution by the local search;
        (c) In case of an improved solution, update the
            best solution constructed so far and the
            corresponding objective value as well.
    2.2. Modify the pheromone trails by applying the
         global updating rule.
Step 3. Return the best solution found.
```

3.1. *Transition rule*

Let τ_{ij} and η_{ij} be, respectively, the pheromone trail and the heuristic information which denote the desire of placing job j in the position i of a sequence. The pheromone trails form a kind of adaptive memory of previously found solutions, whereas the heuristic information represents *a priori* information about the problem instance definition provided by a source different from the ants. At the beginning of the ACA, a fixed value τ_0 is assigned to all initial pheromone trails. Then, at run-time, these intensities are regularly modified with regard to the quality of solutions found.

While constructing a solution, an ant k at the current position i (i = 1, ..., N) chooses the next job j by applying a pseudo-stochastic rule. This transition rule depends on q_0, a parameter between 0 and 1, determining the relative importance of exploitation versus exploration. A random number q uniformly distributed in [0, 1] is then generated. If $q \le q_0$, the unscheduled job j is selected as follows (see Refs. 43 and 55 for more details):

$$j = \arg\max\left[\left(\tau_{ij}\right)^{\alpha}\left(\eta_{ij}\right)^{\beta}\right], \tag{5}$$

where α and β are two positive parameters denoting the relative importance of the pheromone trail versus the heuristic information. In this case, the job with the maximum desirability of placing in the position i is chosen (exploitation). Otherwise, an unscheduled job j is selected according to a probability distribution as follows (exploration):

$$p_{ij}^{k} = \frac{\left(\tau_{ij}\right)^{\alpha}\left(\eta_{ij}\right)^{\beta}}{\sum_{u \in N_i^k}\left(\tau_{iu}\right)^{\alpha}\left(\eta_{iu}\right)^{\beta}}, \quad j \in N_i^k, \tag{6}$$

where N_i^k is the feasible neighborhood of ant k in the position i, that is, the set of jobs that the ant has not yet selected.

3.2. *Heuristic information*

In this study, the heuristic information is calculated using the Gupta's heuristic[53], a heuristic approach for the PFSP with the objective of minimizing the makespan. In this heuristic method, the jobs are ordered in descending order of S_j, in which:

$$S_j = \frac{e_j}{\min\limits_{1 \le m \le M-1}\left[P_{mj} + P_{(m+1)j}\right]}, \quad j = 1,2,\dots,N, \tag{7}$$

where,

$$e_j = \begin{cases} 1, & \text{if } P_{1j} < P_{Mj} \\ -1, & \text{otherwise} \end{cases}. \qquad (8)$$

The Gupta's heuristic is a generalization of the Johnson's rule for $M > 2$. Without loss of generality, it can be considered that all of the processing times are at least one. Clearly,

$$-0.5 \leq S_j \leq 0.5, \qquad j = 1, 2, \ldots, N. \qquad (9)$$

In view of the fact that the heuristic information should be positive and from the above inequality, the heuristic information may then be calculated as follows:

$$\eta_{ij} = S_j + 0.51, \qquad i, j = 1, 2, \ldots, N. \qquad (10)$$

Note that according to Eq. (10) the desirability of placing a job in all positions of a sequence is the same in regard to the heuristic information. Since jobs are chosen from the first position to the last one, if job j has higher η_{ij} (or S_j) than another one, it will be sequenced in the first positions with a higher probability (in case of the same pheromone trails). But, based on the Gupta's heuristic, if job j has higher S_j than another one, it will certainly be sequenced first.

3.3. *Local search procedure*

A new local search procedure is developed in order to improve constructed solutions. When a complete sequence of jobs is generated by an ant (before globally updating the pheromone trails), this procedure is applied on the sequence by moving a job to another position without any change in the other sequence.

As searching a large neighborhood requires more computational time, the proposed local search is based on the trade off between the performance quality of solutions (after conducting local search) and the number of algorithm iterations, i.e., the number of solutions constructed by ants. To handle this issue, we use a threshold parameter *Pr* denoting the probability that each of the jobs is chosen to move to the other positions. Higher value of *Pr* suggests that the number of solutions evaluated in neighborhood of the current solution is likely further, that is, more computational efforts are required. In other words, *Pr* determines the relative importance of the local search procedure versus the ACO algorithm. In order to achieve a good trade off, we set *Pr* equal to 0.01 (based on preliminary experiments).

The proposed local search procedure is then represented as follows:

```
Step 1.  For each job $j$ ($j$ = 1, ..., $N$) do:
    1.1. Generate a random number $R$ uniformly
         distributed in [0, 1];
    1.2. If $R \leq Pr$, then for each position $i$ ($i$ = 1, ..., $N$)
         do:
       If job $j$ is not in the position $i$, then:
         (a) Insert it in this position without any change
             in the other sequence;
         (b) Compute the makespan of the newly
             generated solution.
Step 2.  Determine the best sequence that has been
         obtained.
Step 3.  If the makespan is improved, replace the current
         sequence by the best one found.
```

As an example, let us consider the application of the above local search procedure to a very simple instance consisting of five jobs. Assume that the current constructed sequence is given as
S: 1, 2, 3, 4, 5.
Considering job 1, we generate a random number *R*. Let the random number generated be 0.138217. Since $R > Pr$, this job is not chosen to move to the other positions.
Considering job 2, let the random number generated be 0.002198. Since $R \leq Pr$, job 2 is chosen to move to the other positions. Therefore, the solutions that have to be evaluated in neighborhood of the current solution are given as
*NS*1: 2, 1, 3, 4, 5
*NS*2: 1, 3, 2, 4, 5
*NS*3: 1, 3, 4, 2, 5
*NS*4: 1, 3, 4, 5, 2.
After generating each of these solutions, its objective function value is computed.
Considering the other jobs, let the random numbers generated be 0.913474, 0.754551 and 0.094834, respectively. As seen, none of these jobs is chosen to move.
Now, the best of the four locally generated solutions is compared to the current sequence and if there is an improvement, *S* is replaced by the former. Otherwise, the current sequence is not changed.

3.4. *Global updating of the pheromone trails*

As mentioned earlier, a kind of adaptive memory of previously found solutions is formed by means of the pheromone trails. The global updating rule is proposed to increase the pheromone values on solution components that have been found in good solutions and hence, such components are more likely to be used by the ants in the next iterations of the algorithm.

Once all ants have built their solutions (and after performing the local search), each pheromone trail that is compatible to the sequence of ant k (for each ant in the colony) is modified by applying a new global updating rule as follows:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\frac{Z_1}{C_{\max}^{k}}, \qquad (11)$$

where ρ, a parameter between 0 and 1, is the pheromone trail evaporation rate, $C_{\max}^{k}$ is the makespan of the complete sequence of ant k, and Z_1 is a nonnegative parameter determining the relative importance of the solutions found in the current iteration. As seen, if the performance quality of a solution is higher (i.e., its corresponding makespan is smaller) than another solution, a greater amount of pheromone is deposited on solution components compatible to the former.

Then, each pheromone trail compatible to the best solution obtained so far is updated according to Eq. (12).

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho\frac{Z_2}{C_{\max}^{best}}, \qquad (12)$$

where $C_{\max}^{best}$ is the makespan of the best solution up to the current iteration and Z_2 is a positive parameter determining the relative importance of the best solution.

4. Computational Results

The proposed algorithm has been coded in Visual C^{++} and run on a Pentium 4, 2 GHz PC with 256 MB memory. To evaluate the ACA, we select 120 benchmark problems from Taillard[54] in 12 different sizes: from 20 jobs and 5 machines to 500 jobs and 20 machines (denoted as Ta001 to Ta120). Taillard has produced a set of problems for the PSFP to minimize the makespan. In order to propose problems that are as difficult as possible, Taillard has generated many instances of problems in different sizes and then chosen 10 instances for each size of problems. Therefore, there are 10 instances for each problem size and 120 problem instances in all. Integer processing times have been generated from the uniform distribution [1, 99] for each instance. Subsequently, each of these instances is given with the following information: initial value of the random generator's seed, a lower bound and an upper bound of the optimal makespan. In recent years, it has been shown that the Taillard's solutions are equal or near to the optimum.

4.1. *Parameter settings*

For setting the algorithm parameters, seven different *ant size* (1, 5, 10, 20, 30, 50 and 70), different values of q_0 (0.8, 0.85, 0.9, 0.95 and 0.99), different values of α and β (0.1, 0.5, 1, 1.5 and 2), different values of Z_1 (0, 1, 2, 5 and 10) and different values of Z_2 (1, 2, 5, 10 and 15) have been considered. In addition, parameter ρ has been tested between 0.05 and 0.3 in increments of 0.05. We set the initial pheromone τ_0 equal to 10^{-6}. In the preliminary experiments, the following values of the parameters have been superior and used for all further studies: *ant size* = 5, $q_0 = 0.99$, $\alpha = \beta = 1$, $\rho = 0.25$, $Z_1 = 2$ and $Z_2 = 10$. The algorithm terminates when the total number of iterations in Step 2 (in Sec. 3) reaches 1000.

4.2. *Contribution of the proposed local search*

An important question that may arise is whether the local search procedure really enhances the performance of the ACO algorithm, i.e., whether the algorithm does not perform better without local search but with constructing an additional number of solutions. To show the effect of the proposed local search, an experimental test has been conducted: some of the difficult problem instances (those with M = 20) have been solved with (ACA) and without (ACA-LS) local search. Table 1 provides the minimum, average and maximum makespan achieved by the ACA and ACA-LS over five runs (UB denotes Taillard's upper bound). It should be noted that, to make a fair comparison, for ACA-LS has been allowed the same CPU time as the algorithm with local search. As seen, the ACA is significantly better than the version without local search.

Table 1. Computational results with and without local search.

			ACA			ACA-LS		
Instance	$N\|M$	UB	Min	Average	Max	Min	Average	Max
Ta051	50|20	3886	3946	3985.2	4014	3992	4060.0	4128
Ta081	100|20	6330	6450	6494.4	6539	6527	6582.0	6625
Ta101	200|20	11393	11518	11577.8	11617	11635	11687.0	11813

4.3. *Performance analysis of the algorithm*

Due to the stochastic nature of the proposed algorithm, for each size of problem, we have tested each of the problem instances for five trials. The best trial has been chosen and the ten instances for the same problem size have been averaged. It should be noted that, if the makespan of an obtained solution is $C_{\max}$, the solution quality is then measured by the mean percentage difference from Taillard's upper bound as follows:

$$\text{Quality} = \frac{C_{\max} - \text{UB}}{\text{UB}} \times 100. \qquad (13)$$

The final results are shown in Table 2, which gives a comparison with ACS, simulated annealing (SA) and genetic algorithm (GA) (best values are indicated in boldface). ACS, proposed[47] by Ying and Liao, is an implementation of ant colony system, a particular version of ACO proposed[43] by Dorigo and Gambardella. Ying and Liao have represented the PFSP by a disjunctive graph and proposed a pheromone model based on this graph, i.e., used a new definition for moving artificial ants. They have tested each of the problem instances for five trials and chosen the best one. The results of ACS have then been compared with those of other metaheuristics such as SA and GA proposed[25] by Reeves.

Table 2. Comparison of the ACA with ACS, GA and SA.

N|*M*	ACA	ACS	GA	SA
20|5	**0.368**	1.19	1.61	1.27
20|10	**0.831**	1.70	2.29	1.71
20|20	0.944	1.60	1.95	**0.86**
50|5	**0.085**	0.43	0.45	0.78
50|10	**1.241**	1.89	2.28	1.98
50|20	**1.990**	2.71	3.44	2.86
100|5	**0.070**	0.22	0.23	0.56
100|10	**1.059**	1.22	1.25	1.33
100|20	**1.833**	2.22	2.91	2.32
200|10	**0.434**	0.64	0.50	0.83
200|20	**1.236**	1.30	1.35	1.74
500|20	1.444	1.68	**-0.22**	0.85
Average	**0.961**	1.40	1.50	1.42

Moreover, Table 3 gives a comparison with other well-known ACO algorithms to minimize the makespan for the PFSP from the literature such as MMAS, M-MMAS and PACO. MMAS, proposed[44] by Stutzle, is an implementation of max–min ant system, another particular version of ACO (see Ref. 55 for more details). M-MMAS as well as PACO has been proposed[45] by Rajendran and Ziegler. M-MMAS has extended MMAS by incorporating the concept of summation rule and a new local search procedure, and PACO has developed M-MMAS. The results of M-MMAS and PACO have been compared with those of MMAS. In these algorithms, 9 problems with different sizes have been considered and each of the problem instances has been tested for one trial. In other words, the three large size problems, 200 jobs and 10 machines, 200 jobs and 20 machines, and 500 jobs and 20 machines, have not been examined in those methods.

Table 3. Comparison of the ACA with MMAS, M-MMAS and PACO.

N|*M*	ACA	MMAS	M-MMAS	PACO
20|5	**0.368**	0.408	0.762	0.704
20|10	0.831	**0.591**	0.890	0.843
20|20	0.944	**0.410**	0.721	0.720
50|5	**0.085**	0.145	0.144	0.090
50|10	1.241	2.193	1.118	**0.746**
50|20	1.990	2.475	2.013	**1.855**
100|5	**0.070**	0.196	0.084	0.072
100|10	1.059	0.928	0.451	**0.404**
100|20	1.833	2.238	1.030	**0.985**
Average	0.936	1.065	0.801	**0.713**

For the mean percentage deviation, an average of 0.961 in all of the problem sizes (and 0.936 in 9 problem sizes) with a maximum 1.99 has been achieved by the ACA. It is shown that in most problem sizes, the deviation from the optimal solution is low in the proposed algorithm. It can be seen that the ACA is superior in all problem sizes compared to ACS, in 11 out of 12 problem sizes compared to GA (except for 500 jobs and 20 machines) and in 10 out of 12 problem sizes compared to SA (except for 20 jobs and 20 machines, and 500 jobs and 20 machines). The ACA is also superior in 6 out of 9 problem sizes compared to MMAS, in 5 out of 9 problem sizes compared to M-MMAS and in 4 out of 9 problem sizes compared to PACO. On an average, the ACA outperforms ACS, GA, SA and MMAS, whereas M-MMAS and PACO outperform the ACA.

Another criterion to evaluate algorithms is the computational time. Since only the CPU times of ACS are provided by the literature, we compare the time results in Table 4. ACS has been coded in Visual C^{++} and run on an AMD 700 MHz PC, which is approximately two to three times slower than a 2 GHz

Pentium 4 PC. Therefore, to make a fair comparison, the CPU times of ACS reported in Ref. 47 have been divided by the transformation factor three in Table 4. With regard to this matter, it is seen that in all of the different sizes the proposed algorithm is much faster than ACS. Finally, it can be seen that the proposed algorithm can get very good solutions at a reasonable CPU time.

Table 4. Comparison of the CPU times (in seconds).

N|*M*	ACA	ACS
20|5	0.44	3.67
20|10	0.50	4.00
20|20	0.63	5.33
50|5	2.77	14.67
50|10	3.73	18.00
50|20	5.91	24.33
100|5	14.15	54.33
100|10	21.93	65.67
100|20	37.79	88.00
200|10	141.52	275.33
200|20	254.06	631.67
500|20	3744.25	5133.00
Average	352.31	526.5

5. Conclusions

In this paper, an efficient ant colony optimization algorithm is developed to solve the permutation flow shop scheduling problem with the objective of minimizing makespan. At first, an initial value is assigned to all pheromone trails. Each artificial ant constructs a solution by applying a pseudo-stochastic rule based on both the Gupta's heuristic and the pheromone trails. Then, the constructed solution is improved by the proposed local search procedure. Once all ants have terminated their generations, the pheromone trails are modified using the global updating rule. To evaluate the performance of the proposed hybrid algorithm, it has been tested on the benchmark problems due to Taillard, and then compared with other ant colony optimization algorithms, genetic algorithm and simulated annealing available in the literature. The results are given to show the power of the proposed approach for producing very good solutions at a reasonable CPU time. The results incite the extension of the algorithm for other scheduling problems.

References

1. I. Ribas, R. Leisten and J. M. Framinan, Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective, *Comput. Oper. Res.* **37** (2010) 1439–1454.
2. C. Kahraman, O. Engin, I. Kaya and M. K. Yilmaz, An application of effective genetic algorithms for Solving Hybrid Flow Shop Scheduling Problems, *Int. J. Comput. Int. Sys.* **1**(2) (2008) 134–147.
3. M. Pinedo, *Planning and Scheduling in Manufacturing and Services* (Springer Series in Operations Research, Springer, 2005).
4. E. Ignall and L. Schrage, Application of the branch-and-bound technique to some flowshop scheduling problems, *Oper. Res.* **13** (1965) 400–412.
5. Z. A. Lomnicki, A branch and bound algorithm for the exact solution of the three-machine scheduling problem, *Oper. Res. Quarterly* **16**(1) (1965) 89–100.
6. S. P. Bansal, Minimizing the sum of completion times of *n*-jobs over *M*-machines in a flowshop — a branch and bound approach, *AIIE Trans.* **9** (1977) 306–311.
7. E. F. Stafford, On the development of a mixed integer linear programming model for the flowshop sequencing problem, *J. Oper. Res. Soc.* **39** (1988) 1163–1174.
8. M. Nawaz, Jr. E. E. Enscore and I. Ham, A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem, *OMEGA–Int. J. Manage. S.* **11**(1) (1983) 91–95.
9. C. Rajendran and H. Ziegler, An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs, *Eur. J. Oper. Res.* **103** (1997) 129–138.
10. J. M. Framinan, R. Leisten and R. Ruiz-Usano, Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimization, *Eur. J. Oper. Res.* **141** (2002) 559–569.
11. J. M. Framinan and R. Leisten, A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness, *Int. J. Prod. Econ.* **99** (2006) 28–40.
12. P. J. Kalczynski and J. Kamburowski, An improved NEH heuristic to minimize makespan in permutation flow shops, *Comput. Oper. Res.* **35** (2008) 3001–3008.
13. D. Laha and S. C. Sarin, A heuristic to minimize total flow time in permutation flow shop, *OMEGA–Int. J. Manage. S.* **37** (2009) 734–739.
14. X. Li, Q. Wang and C. Wu, Efficient composite heuristics for total flowtime minimization in permutation flowshops, *OMEGA–Int. J. Manage. S.* **37** (2009) 155–164.
15. I. Ribas, R. Companys and X. Tort-Martorell, Comparing three-step heuristics for the permutation flow shop problem, *Comput. Oper. Res.* **37** (2010) 2062–2070.
16. A. H. G. Rinnooy Kan, *Machine Scheduling Problems: Classification, Complexity, and Computations* (Martinus Nijhoff, The Hague, 1976).

17. M. R. Garey, D. S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling, *Math. Oper. Res.* **1** (1976) 117–129.
18. C. R. Reeves, Improving the efficiency of tabu search for machine sequencing problem, *J. Oper. Res. Soc.* **44**(4) (1993) 375–382.
19. E. Nowicki and C. Smutnicki, A fast tabu search algorithm for the permutation flowshop problem, *Eur. J. Oper. Res.* **91** (1996) 160–175.
20. M. Ben-Daya and M. Al-Fawzan, A tabu search approach for the flow shop scheduling problem, *Eur. J. Oper. Res.* **109** (1998) 88–95.
21. J. Grabowski and J. Pempera, New block properties for the permutation flow-shop problem with application in TS, *J. Oper. Res. Soc.* **52** (2001) 210–220.
22. J. P. Watson, L. Barbulescu, L. D. Whitley and A. E. Howe, Contrasting structured and random permutation flowshop scheduling problems: Search space topology and algorithm performance, *ORSA J. Comput.* **14**(2) (2002) 98–123.
23. J. Grabowski and M. Wodecki, A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion, *Comput. Oper. Res.* **31**(11) (2004) 1891–1909.
24. B. Eksioglu, S. D. Eksioglu and P. Jain, A tabu search algorithm for the flowshop scheduling problem with changing neighborhoods, *Comput. Ind. Eng.* **54** (2008) 1–11.
25. C. R. Reeves, A genetic algorithm for flowshop sequencing, *Comput. Oper. Res.* **22**(1) (1995) 5–13.
26. T. Murata H. Ishibuchi and H. Tanaka, Genetic algorithms for flowshop scheduling problems, *Comput. Ind. Eng.* **30**(4) (1996) 1061–1071.
27. C. R. Reeves and T. Yamada, Genetic algorithms, path relinking and the flowshop sequencing problem, *Evol. Comput.* **6** (1998) 45–60.
28. M. S. Nagano, R. Ruiz and L. A. N. Lorena, A Constructive Genetic Algorithm for permutation flowshop scheduling, *Comput. Ind. Eng.* **55** (2008) 195–207.
29. Y. Zhang, X. Li and Q. Wang, Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization, *Eur. J. Oper. Res.* **196** (2009) 869–876.
30. L. Y. Tseng and Y. T. Lin, A hybrid genetic local search algorithm for the permutation flowshop scheduling problem, *Eur. J. Oper. Res.* **198** (2009) 84–92.
31. G. I. Zobolas, C. D. Tarantilis and G. Ioannou, Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, *Comput. Oper. Res.* **36** (2009) 1249–1267.
32. I. Osman and C. Potts, Simulated annealing for permutation flow shop scheduling, *OMEGA–Int. J. Manage. S.* **17**(6) (1989) 551–557.
33. F. Ogbu and D. Smith, The application of the simulated annealing algorithm to the solution of the $n/m/C_{\max}$ flowshop problem, *Comput. Oper. Res.* **17**(3) (1990) 243–253.
34. F. Ogbu and D. Smith, Simulated annealing for the permutation flow-shop problem, *OMEGA–Int. J. Manage. S.* **19**(1) (1991) 64–67.
35. H. Ishibuchi, S. Misaki and H. Tanaka, Modified simulated annealing algorithms for the flow shop sequencing problems, *Eur. J. Oper. Res.* **81** (1995) 388–398.
36. C. J. Liao, C. T. Tseng and P. Luarn, A discrete version of particle swarm optimization for flowshop scheduling problems, *Comput. Oper. Res.* **34** (2007) 3099–3111.
37. M. F. Tasgetiren, Y. C. Liang, M. Sevkli and G. Gencyilmaz, A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, *Eur. J. Oper. Res.* **177** (2007) 1930–1947.
38. B. Jarboui, S. Ibrahim, P. Siarry and A. Rebai, A combinatorial particle swarm optimization for solving permutation flowshop problems, *Comput. Ind. Eng.* **54** (2008) 526–538.
39. Z. Lian, X. Gu and B. Jiao, A novel particle swarm optimization algorithm for permutation flow-shop scheduling to minimize makespan, *Chaos Soliton. Fract.* **35** (2008) 851–861.
40. J. Zhang, C. Zhang and S. Liang, The circular discrete particle swarm optimization algorithm for flow shop scheduling problem, *Expert Syst. Appl.* **37** (2010) 5827–5834.
41. M. Dorigo, *Optimization, learning and natural algorithm*, in Italian (PhD thesis, DEI, Politecnico di Milano, Itally, 1992).
42. M. Dorigo, V. Maniezzo and A. Colorni, The ant system: Optimization by a colony of cooperating agents, *IEEE T. Syst. Man Cy. B* **26** (1996) 29–41.
43. M. Dorigo and L. M. Gambardella, Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE T. Evolut. Comput.* **l** (1997) 53–66.
44. T. Stutzle, An ant approach for the flow shop problem, in *Proc. 6th Eur. Cong. Intelligent Techniques and Soft Computing*, EUFIT '98 (Aachen: Verlag Mainz 3, 1998), pp. 1560–1564.
45. C. Rajendran and H. Ziegler, Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, *Eur. J. Oper. Res.* **155** (2004) 426–438.
46. C. Rajendran and H. Ziegler, Two ant-colony algorithms for minimizing total flowtime in permutation flowshops, *Comput. Ind. Eng.* **48** (2005) 789–797.
47. K. C. Ying and C. J. Liao, An ant colony system for permutation flow-shop sequencing, *Comput. Oper. Res.* **31** (2004) 791–801.
48. Y. Gajpal and C. Rajendran, An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops, *Int. J. Prod. Econ.* **101** (2006) 259–272.

49. V. T'kindt, N. Monmarche, F. Tercinet and D. Laugt, An Ant Colony Optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem, *Eur. J. Oper. Res.* **142** (2002) 250–257.
50. B. M. T. Lin, C. Y. Lu, S. J. Shyu and C. Y. Tsai, Development of new features of ant colony optimization for flowshop scheduling, *Int. J. Prod. Econ.* **112** (2008) 742–755.
51. B. Yagmahan and M. M. Yenisey, Ant colony optimization for multi-objective flow shop scheduling problem, *Comput. Ind. Eng.* **54** (2008) 411–420.
52. B. Yagmahan and M. M. Yenisey, A multi-objective ant colony system algorithm for flow shop scheduling problem, *Expert Syst. Appl.* **37** (2010) 1361–1368.
53. J. N. D. Gupta, Heuristic algorithms for multistage flowshop scheduling problem, *AIIE Trans.* **4** (1972) 11–18.
54. E. Taillard, Benchmarks for basic scheduling problems, *Eur. J. Oper. Res.* **64** (1993) 278–285.
55. M. Dorigo and T. Stutzle, The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, in *Handbook of Metaheuristics*, eds. F. Glover and G. Kochenberger (Kluwer Academic Publishers, 2003), pp. 251–285.