Information Processing Letters 108 (2008) 204–209

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

An improved particle swarm optimization algorithm for flowshop scheduling problem

Changsheng Zhang [a,*], Jigui Sun [a], Xingjun Zhu [a], Qingyun Yang [b]

[a] Key Laboratory of Symbol Computation and Knowledge Engineering of the Ministry of Education, Changchun 130012, China
[b] Changchun Institute of Optics, Fine Mechanics and Physics, Chinese Academy of Sciences, China

ARTICLE INFO

Article history:
Received 15 February 2008
Received in revised form 30 April 2008
Available online 24 May 2008
Communicated by L. Boasson

Keywords:
Flow shop scheduling problem
Particle swarm optimization
Makespan
Combinatorial problems

ABSTRACT

The flowshop scheduling problem has been widely studied and many techniques have been applied to it, but few algorithms based on particle swarm optimization (PSO) have been proposed to solve it. In this paper, an improved PSO algorithm (IPSO) based on the "alldifferent" constraint is proposed to solve the flow shop scheduling problem with the objective of minimizing makespan. It combines the particle swarm optimization algorithm with genetic operators together effectively. When a particle is going to stagnate, the mutation operator is used to search its neighborhood. The proposed algorithm is tested on different scale benchmarks and compared with the recently proposed efficient algorithms. The results show that the proposed IPSO algorithm is more effective and better than the other compared algorithms. It can be used to solve large scale flow shop scheduling problem effectively.

1. Introduction

Exact algorithms [1,2] can hardly be designed to solve large sequencing and scheduling problems. In general, these problems belong to the class of combinatorial optimization problems characterized as NP-hard [3] and therefore the right way to process is with heuristic techniques [4–6]. It consists in the assignment of a set of jobs $N = \{J_1, \ldots, J_n\}$, each of which consists of a set of operations $J_j = \{O_{j1}, \ldots, O_{jm}\}$ to a set of machines $M = \{M_1, \ldots, M_m\}$. The parameter $t_{i,j}, 1 \leqslant i \leqslant n, 1 \leqslant i \leqslant m$ denotes the processing time of job i on machine j. Operations belonging to a job are ordered with respect to the precedence constraint $O_{jk} \triangleright O_{j,k+1}$. Each machine can process only one job at a time. The given technological machine sequence is identical for all jobs, and it is assumed that the sequence of jobs on machines is also the same. A schedule may therefore be represented as a permutation $\pi = \{\pi_1, \ldots, \pi_n\}$ of jobs which can be mapped into a schedule defining completion time C_{jk} for the operation O_{jk}. The completion time C_j of job J_j is then the completion of last operation $O_{jm}, C_j = C_{jm}$. The objective function for the FSSP corresponds to the minimization of the makespan in this paper. So the FSSP model can be defined as follows:

$$C_{\pi_1 1} = t_{\pi_1 1} \tag{1}$$

$$C_{\pi_j 1} = C_{\pi_{j-1} 1} + t_{\pi_j 1} \quad \forall j \in \{2, \ldots, n\} \tag{2}$$

$$C_{\pi_1 k} = C_{\pi_1 k-1} + t_{\pi_1 k} \quad \forall k \in \{2, \ldots, m\} \tag{3}$$

$$C_{\pi_j k} = \max\{C_{\pi_{j-1},k}, C_{\pi_j k-1}\} + t_{\pi_j k} \quad \forall j \in \{2, \ldots, n\}, \quad k \in \{2, \ldots, m\} \tag{4}$$

$$C_{\max} = \max C_j \tag{5}$$

Particle swarm optimization (PSO) is an evolutionary computation technique developed by Dr. Eberhart and Dr. Kennedy in 1995 [7,8]. It is inspired by the social behavior of bird flocking and possessing the properties of easy implementation and fast convergence. Clerc and Kennedy researched on the explosion stability and convergence in a multi-dimensional complex space of the particle

* Corresponding author. Tel.: +86-0431-85166487.
E-mail address: zcs820@yahoo.com.cn (C. Zhang).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2008.05.010

swarm in [9] and Trelea studied convergence analysis and parameter selection of the particle swarm optimization algorithm in [10]. Eberhart and Shi compared genetic algorithm with particle swarm optimization in [11]. In recent years there have been a lot of reported works focused on the modification PSO such as in [9,12] and other optimization algorithms in [13,14] to solve continuous optimization problems, but its being used to solve FSSP does not have rich literatures. By far, only several papers have been delivered to solve the FSSP based on PSO algorithm [15–17], and the experimental results show that they are more efficacious than the algorithms based on GA [18] and constructive heuristics [19–21], but these algorithms are still suffered from the problem of premature convergence and easily trapped into local optimum.

Recently, van den Bergh [22] found a dangerous property of the PSO algorithm: if a particle's current position coincides with the global best position, the particle will only move away from this point if its previous velocity and the inertia weight are non-zero. If their previous velocities are very close to zero, then all particles will stop moving once they catch up with the global best particle, which may lead to premature convergence of the algorithm. To pre-actively counter this behavior in a particle swarm, an improved particle swarm optimization algorithm is proposed in which the shift mutation operator is used to search its neighborhood when a particle is going to stagnate. This cannot only improve the particle's exploration capability but also make the swarm escape from local minima. Furthermore, a fast makespan computation method based on matrix is designed to improve the algorithm speed. Finally, the IPSO algorithm is tested on different scale benchmarks and compared with the recently proposed efficient algorithms.

2. IPSO algorithm

One of the key issues when designing the PSO algorithm lies in its solution representation where particles bear the necessary information related to the problem domain on hand. FSSP is set in a discrete space. It is obvious that standard PSO equations cannot be used to generate a discrete job permutation since positions and velocities are real-valued. So the most important issue in applying PSO successfully to FSSP is to develop an effective problem mapping and solution generation mechanism. If these two mechanisms are devised successfully, it is possible to find good solutions for a given optimization problem in acceptable time.

2.1. The IPSO model

For a flow shop scheduling problem consisting of n jobs, according to the character of FSSP, if the ith particle's current position $X_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\}, x_{ij} \in N$ and velocity $V_i = \{v_{i1}, v_{i2}, \ldots, v_{in}\}, v_{ij} \in N$ are both denoted as permutations of all jobs which must satisfy the "*alldifferent*" constraint [23], then the crossover operator can be used to redefine the model of the original PSO algorithm [8]. Since the behavior of a particle is mainly influenced by its momentum term, cognitive component and social component. So we have the following iterative formulas:

$$V_i(k+1) = V_i(k) \otimes P_{gbest} \otimes P_{ibest} \tag{6}$$

$$X_i(k+1) = X_i(k) \otimes V_i(k+1) \tag{7}$$

where $\otimes$ denotes the crossover operation. By analyzing the above equations, it can be obtained that, each particle follows two "best" positions, the current global best position P_{gbest} and the personal best position P_{ibest}. Like the original PSO algorithm, it possesses the properties of fast convergence, simple to compute and easy to implement as the later experiments showed. However, because the velocities of particles rapidly approach to zero that each particle's velocity and current position have the same permutation, the above iterative formulas exhibit the disadvantage of easy to be trapped in local optima like the similar particle swarm optimization algorithm proposed in paper [16].

From Eqs. (6) and (7), we can get the following two sights: First, when a particle's personal best position is the same as the current global best position and its current speed is close to zero, it is going to stagnate; second, when the achieved new speed $V_i(k+1)$ is the same as the current position $X_i(k)$, the particle will not move. In order to improve the algorithm's performance, for the first case, we make the particle search the neighborhood of the current global position as Eq. (8) shows, and for the second case, the neighborhood of the particle's current position is explored as Eq. (9) shows.

$$X_i(k+1) = mutation\big(P_{gbest}(k)\big) \tag{8}$$

$$X_i(k+1) = mutation\big(X_i(k)\big) \tag{9}$$

For a flow shop problem with n jobs, a schedule $\pi = \{\pi_1, \ldots, \pi_n\}$ can be looked as a point of n dimensions space. Given a job $\pi_i, 1 \leqslant i \leqslant n$, a new point or schedule can be got by inserting it into any other positions. The overall points found through this way form the neighborhood of point π. Obviously, it includes $n(n-2)+1$ points. So, the shift mutation [24] can be used to perform the neighborhood search process.

2.2. Fitness computation

In order to speed up the IPSO algorithm, a fast fitness computation method is devised in this paper. For a scheduling problem containing n jobs and each job consisting of m operations, processing time matrix of a valid schedule $S = \langle s_1, s_2, \ldots, s_n \rangle$ is defined as

$$T = \begin{bmatrix} t_{1,s_1} & t_{1,s_2} & \ldots & t_{1,s_n} \\ t_{2,s_1} & t_{2,s_2} & \ldots & t_{2,s_n} \\ \vdots & \vdots & \vdots & \vdots \\ t_{m,s_1} & t_{m,s_2} & \ldots & t_{m,s_n} \end{bmatrix}$$

where $s_i \in J$, t_{jk} is the processing time of job j on machine $k, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$.

In the IPSO algorithm, each particle's fitness value is expressed by the makespan of the corresponding schedule. It can be computed as follows: First, get the processing time matrix P of the permutation through swapping the column

of original processing time matrix. Second, update the matrix P according to the following rule.

$$t_{i,j} = \begin{cases} t_{1,1} & \text{if } i = 1,\ j = 1 \\ t_{1,j-1} + t_{1j} & \text{if } i = 1,\ j \neq 1 \\ t_{i-1,1} + t_{i,1} & \text{if } i \neq 1,\ j = 1 \\ t_{i-1,j} + t_{i,j} & \text{if } t_{i-1,j} > t_{i,j-1} \\ t_{i,j-1} + t_{i,j} & \text{if } t_{i-1,j} < t_{i,j-1} \end{cases} \tag{10}$$

Then, the resulted value of element $p_{m,n}$ is the makespan of the schedule.

2.3. Algorithm description

To ensure convergence to desirable, better makespans in a reasonable amount of time, the initialization procedure in the IPSO algorithm is based on the NEH algorithm [24] and can be explained as follows:

1. The total processing times for the jobs on the m machines are calculated:
$$P_i = \sum_{j=1}^{m} p_{ij}, \quad i = 1, \ldots, n$$
2. The jobs are sorted in descending order of P_i to form an ordered list and make it as each particle's current position.
3. For each particle, pick two random jobs and exchange them for the two first jobs from its current position. Then, the first two jobs are taken and the two possible schedules containing them are evaluated.
4. For each particle, take job $i, i = 3, \ldots, n$ and find the best schedule by placing it in all possible i positions in the sequence of jobs that have already scheduled. The obtained best sequence would be selected for the next iteration.
   Furthermore, each particle's personal best position is initialized the same as its current position. Give each particle a maximal initial velocity being equal to the reverse order of its current position, calculate its fitness and set the global best. The termination criterion for the iterations is determined according to whether the max generation is reached or the maximum computation time has arrived. The framework of IPSO algorithm can be described as follows:

Algorithm IPSO (*improved PSO algorithm*)
Input: MaxG (the maximal generation number).
T (the maximum computation time)
N (size of the population)
Output: S (the obtained best schedule).

Step 1: *Initialization.*

Generate an initial population POP with N particles. For each particle, set its current position as its current personal best position and set the current generation number $k = 0$ and $t = T$.

Step 2: Update the current global best position P_{gbest}.

for each particle $P_i[k]$**do**
if $C_{max}(P_{ibest}) < C_{max}(P_{gbest})$ **then**
$P_{gbest} = P_{ibest}$ **endif**;
endfor.

Step 3: Stopping criterion.

if(g==MaxG or t==0) **then**
output the obtained global best position and **stop**
endif.

Step 4: Swarm evolution.

for each particle $P_i[k]$**do**
//update the speed and position of particle i
if $(V_i[k] == X_i[k]) \&\& (P_{ibest} == P_{gbest})$ **then**
$X_i[k+1] =$ mutation(P_{gbest});
$V_i[k+1] = V_i[k]$;
else
$V_i[k+1] = V_i[k] \otimes P_{gbest} \otimes P_{ibest}[k]$;
if $V_i[k+1] == X_i[k]$ **then**
$X_i[k+1] =$ mutation($X_i[k]$);
else
$X_i[k+1] = V_i[k+1] \otimes X_i[k]$;
endif.
endif.
//update personal best
if $C_{max}(X_i[k+1]) < C_{max}(P_{ibest})$ **then**
$P_{ibest} = X_i[k+1]$;
endfor.

Step 5: $g = g + 1$; $t = t - 1$; **goto** *step 2.*

Generally, the definition of convergence is that a system or process reaches a stable state. For the population based optimization algorithm, the convergence of algorithm can be defined in terms of either individual or the whole swarm. F. Van den Bergh gave the convergence definition of original PSO in [22]. Based on it, the convergence definition for the proposed IPSO algorithm can be described as follows: For a given flow shop scheduling problem P and its corresponding problem search space Ω, $gbest(t) \in \Omega$ is best position found in time t or in tth generation, $gbest^*$ is a fixed position in Ω. The convergence is written as,

$$\lim_{t \to \infty} gbest(t) = gbest^*$$

It implies that, if *gbest* output by IPSO does not change any more, then the convergence is achieved. If the *gbest* is the global best position, then the algorithm attains the global best convergence. Otherwise, the algorithm is stuck in local optima.

3. Validation on benchmarks

In this part, we provide a comprehensive experimental evaluation and comparison of the proposed IPSO algorithm with other powerful methods. The well known standard benchmark set of Taillard [24] that is composed of 120 instances ranging from 20 jobs and five machines to 500 jobs and 20 machines is used. To compare the performance of

the IPSO algorithm with known techniques from the literature, we compared it with nine classical or recent, well-performing algorithms. These various methods are listed in Table 1.

Each algorithm is run twenty independent times for the selected different scale test instances with the swarm size $N = 60$ for all the problems except NEHT algorithm, for which only single run was done since it is a deterministic algorithm. The stopping criterion for all algorithm alternatives in the experiment is set to a maximum computation time of n*(m/2)*120 milliseconds or a fixed fitness evaluation numbers computed as n*m*500 for each test instance. Setting the time limit in this way allows more computation time or fitness evaluation numbers as the number of jobs or the number of machines increases. We implemented them using the C++ language. They were all executed on my PC with Pentium(R) processor (2.8 GHz) and 1 GB memory. Regarding the performance measures, we measure for each instance the average relative percentage deviation ($\overline{RPD}$). They are computed as follows:

$$\overline{RPD} = \sum_{i=1}^{L}\left(\frac{Sol_i - Optimal}{Optimal}\cdot 100\right)\Big/ L$$

Where L is the run times, Sol_i is the makespan obtained for the ith running of an algorithm and *Optimal* is the known minimum makespan for the problem or the lowest known upper bound for Taillard's instances as of April 2005. The comparative results in terms of computation time and fitness evaluation numbers for all the algorithms (averaged by instance size) are provided in Tables 2 and 3.

Table 1
List of various methods used in the comparison

Method	Time
NEH with the improvements of Taillard (NEHT) [24]	1990
Genetic algorithm of Ruize et al. (GA-RMA) [25]	2006
Simulated annealing of Osman and Potts (SA_OP) [26]	1989
Tabu search algorithm of Widmer and Herz (SPIRIT) [27]	1989
Genetic algorithms of Andreas (GA_AN) [28]	2004
Genetic algorithm of Aldowaisan and Allahvedi (GA_AA) [29]	2003
Hybrid genetic algorithm of Murata et al. (GA_M1T) [30]	1996
Novel particle swarm optimization of Lian (NPSO) [17]	2008
Ant colony optimization algorithm of Betul (ACO) [31]	2008

From Table 2, we can see that the NEHT constructive heuristic yields an ARPD of 3.35%, which is much better than the meta-heuristics Spirit and GA_AN. Moreover, NEHT is very quick; it takes only 186 ms on average to solve the largest instances with 500 jobs and 20 machines. In fact, recent studies have confirmed the superiority of NEHT over most recent constructive heuristics. The meta-heuristics SPIRIT and GA_AN obtain rather poor results, even worse than NEHT. Much better performance is obtained by SA_OP, even though it starts the search from a random solution and not a NEHT generated one. SA_OP is one of the easiest to implement algorithms and it can reach a level of performance comparable to several genetic algorithms including GA_AA and GA_MIT which is a fairly complex GA that is hybrid with local search. Among the compared genetic algorithm, the best performance is obtained by GA_RMA algorithm which includes several population operators and initiates with NEH algorithm. It achieved an ARPD 1.13%, better than the recently proposed ACO algorithm. There are two algorithms that manage less than a 1% ARPD. They are NPSO, and the proposed IPSO algorithm. The IPSO gets the smallest ARPD and much better than the other compared algorithms under the same elapsed time as a stopping criterion. The ARPD obtained by IPSO algorithm is also smallest than others when compared in terms fitness evaluation numbers which can be seen from Table 3. Compared with Table 2, the obtained ARPDs of NEHT and GA_AN changes little, the ARPDs obtained by NPSO and IPSO both become a little worse, and the ARPDs achieved by other compared algorithms get smaller. This is mainly for that some local search strategy is used or the population needs sorted often in these algorithms but in NPSO and IPSO algorithm, only simple operation is used and in essence the evolution speed of PSO based algorithm is faster the algorithm based on GA. The population of NPSO algorithm used in this paper is also initialized using the NEH algorithm like IPSO.

In order to check whether these observed differences in the $\overline{RPD}$ values are indeed statistically significant, we performed an analysis of variance. This analysis has a single factor which is the type of algorithm with 10 levels. The reposed variable is 120 relative percentage deviation (RPD)

Table 2
Average relative percentage deviation over the optimum solution or lowest known upper bound for Taillard's instances obtained by the methods evaluated with the running time

Instance	NEHT	GA_RMA	SA_OP	SPIRIT	GA_AN	GA_AA	GA_MIT	ACO	NPSO	IPSO
20 × 5	3.35	0.24	1.17	3.91	3.95	0.94	0.84	0.62	0.21	**0.04**
20 × 10	5.02	0.62	2.69	5.41	5.18	1.70	1.96	2.04	0.37	**0.23**
20 × 20	3.73	0.37	2.21	4.51	4.26	1.31	1.66	1.32	0.24	**0.19**
50 × 5	0.84	0.06	0.45	1.99	2.01	0.37	0.30	0.21	**0.01**	0.05
50 × 10	5.12	1.79	3.71	5.95	6.54	3.60	3.50	2.06	1.85	**1.05**
50 × 20	6.26	2.67	4.57	7.64	7.74	4.66	5.07	3.56	**1.59**	1.83
100 × 5	0.46	0.07	0.33	0.98	1.35	0.26	0.25	0.17	**0.03**	**0.03**
100 × 10	2.13	0.65	1.52	3.13	3.86	1.65	1.54	0.85	**0.37**	0.39
100 ×20	5.23	2.78	4.79	6.65	8.15	4.92	4.99	3.41	2.19	**2.06**
200 ×10	1.43	0.43	1.08	2.08	2.78	1.08	1.14	0.55	0.37	**0.32**
200 × 20	4.41	2.35	4.11	5.00	7.05	3.95	4.19	2.84	1.92	**1.80**
500 × 20	2.24	1.43	2.34	9.87	4.76	2.06	2.68	1.66	1.19	**1.16**
Average	3.35	1.12	2.42	4.76	4.80	2.21	2.34	1.60	0.86	0.76

Table 3
Average relative percentage deviation over the optimum solution or lowest known upper bound for Taillard's instances obtained by the methods evaluated with the fitness evaluation numbers

Instance	NEHT	GA_RMA	SA_OP	SPIRIT	GA_AN	GA_AA	GA_MIT	ACO	NPSO	IPSO
20 × 5	3.35	0.20	1.12	4.58	3.94	0.84	0.54	0.58	0.26	**0.04**
20 × 10	5.02	0.62	2.63	5.43	5.18	1.42	1.73	0.96	0.42	**0.36**
20 × 20	3.73	0.31	2.26	4.69	4.23	1.25	1.36	0.86	0.34	**0.28**
50 × 5	0.84	**0.06**	0.45	2.06	2.01	0.36	0.23	0.12	0.25	0.08
50 × 10	5.12	1.45	3.58	5.71	6.54	3.41	3.42	1.95	2.46	**1.31**
50 × 20	6.26	2.50	4.51	7.52	7.76	4.68	4.73	2.81	2.58	**2.06**
100 × 5	0.46	0.06	0.34	0.95	1.35	0.26	0.22	0.12	**0.05**	0.08
100 × 10	2.13	0.53	1.50	3.12	3.86	1.59	1.45	0.86	0.59	**0.39**
100 × 20	5.23	**2.51**	4.69	6.56	8.25	4.81	4.65	3.32	2.68	2.53
200 × 10	1.43	0.40	0.94	1.96	2.66	0.99	1.01	0.45	0.61	**0.38**
200 × 20	4.41	**2.16**	3.96	5.03	6.93	3.95	3.95	2.19	2.27	2.20
500 × 20	2.24	1.39	2.26	6.47	4.79	1.96	2.41	1.58	2.38	**1.37**
Average	3.35	1.01	2.35	4.50	4.79	2.13	2.14	1.31	1.24	0.92

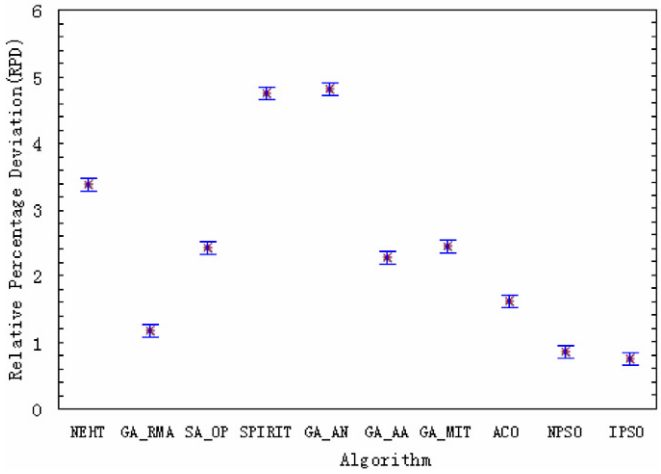

Fig. 1. Means plot of the relative percentage deviation (RPD) for the Taillard benchmark and the algorithms tested.

values for each algorithm. The response variable of the experiment is then calculated with the following expression:

$$RPD = \frac{Some_{sol} - Optimal}{Optimal} \times 100$$

where $Some_{sol}$ is the solution obtained by a given algorithm alternative on a given instance with the elapsed time as a stopping criterion. The response variable is, therefore, the average percentage increase over the lowest upper bound for each instance. The mean plot for the single factor is depicted in Fig. 1. From the results we see that our proposed IPSO algorithm produces statistically better results than all others. There are no statistically significant differences between SPIRIT and GA_AN. The GA_MIT and SA_OP algorithms are also almost equivalent.

In the previous evaluation we have considered many of the best known metaheuristics, including several GAs as well as other recent methods. However, the comparison does not consider some of the state-of-the-art metaheuristics with local search like such as and HGA_RMA [25]. There are several reasons for this. There exists no local search strategy in the proposed IPSO algorithm and a re-implementation of them will take long coding time. Based on the above experiments, we can obtained that, the proposed IPSO algorithm reaches and even surpasses the performance of the compared state-of the-art algorithms at a computation time penalty. Another advantage of the IPSO algorithm is that it is much simpler and only includes two parameters.

4. Conclusions

In this paper, the IPSO algorithm is proposed for the flow shop scheduling problem which combines the PSO with the crossover and mutation operators, and a fast fitness computation method based on matrix is devised to speed up it. The IPSO algorithm is tested on different scale problems and compared with the recently proposed algorithms. The results show that the IPSO algorithm has obtained better performance and possesses better convergence property than the compared algorithms. Another good quality of the IPSO algorithm is that it contains only two parameters, the swarm size and the max generation, avoiding the time consuming work of parameter turning. There are a number of research directions that can be considered as useful extensions of this research. The proposed algorithm in this paper using single crossover operator with single mutation operator, maybe

using several crossovers and various mutation operators is more effective for flow shop scheduling problem. Only the two point crossover operator is included in this paper, the other crossover operators should be considered and compared next. Furthermore, applying the proposed algorithm to solve other combinatorial optimization problems is also possible in further research.

References

[1] C. Dimopoulos, A.M.S. Zalza, Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and computations, IEEE Transactions on Evolutionary Computation 4 (2) (2000) 93–113.

[2] J.M.S. Valente, R.A.F.S. Alves, An exact approach to early/tardy scheduling with release dates, Computers & Operations Research 32 (11) (2005) 2905–2917.

[3] M. Garey, D. Johnson, R. Sethy, The complexity of flow shop and job shop scheduling, Mathematics of Operations Research 1 (2) (1976) 117–129.

[4] A. Sadegheih, Scheduling problem using genetic algorithm, simulated annealing and the effects of parameter values on GA performance, Applied Mathematical Modelling 30 (2) (2006) 147–154.

[5] C. Rajendran, H. Ziegler, Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, European Journal of Operational Research 155 (2004) 426–438.

[6] J. Grabowski, M. Wodecki, A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion, Computers & Operations Research 31 (11) (2004) 1891–1909.

[7] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, In: Proc of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, 1995, pp. 39–43.

[8] J. Kennedy, R. Eberhart, Particle swarm optimization. In: IEEE Int. Conf. on Neural Networks, Perth, Australia, 1995, pp. 1942–1498.

[9] S. He, Q.H. Wu, J.Y. Wen, J.R. Saunders, R.C. Paton, A particle swarm optimizer with passive congregation, BioSystems 78 (2004) 135–147.

[10] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Inform. Process. Lett. 85 (6) (2003) 317–325.

[11] R.C. Eberhart, Y. Shi, Comparison between genetic algorithms and particle swarm optimization, in: Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming, Springer-Verlag, Berlin, San Diego, CA, 1998, pp. 611–618.

[12] Bo Liu, Ling Wang, Yi-Hui Jin, Fang Tang, De-Xian Huang, Improved particle swarm optimization combined with chaos, Chaos, Solitons & Fractals 25 (2005) 1261–1271.

[13] M.J. Ji, H.W. Tang, Application of chaos in simulated annealing, Chaos, Solitons & Fractals 21 (2004) 933–941.

[14] Z. Lu, L.S. Shieh, G.R. Chen, On robust control of uncertain chaotic systems: a sliding-mode synthesis via chaotic optimization, Chaos, Solitons & Fractals 18 (2003) 819–827.

[15] K. Rameshkumar, R.K. Suresh, K.M. Mohanasundaram, Discrete particle swarm optimization (DPSO) algorithm for permutation flowshop scheduling to minimize makspan, in: Proc. ICNC 2005, LNCS, vol. 3612, 2005, pp. 572–581.

[16] Lian Zhigang, Xingsheng Gu, Bin Jiao, A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan, Applied Mathematics and Computation 175 (1) (2006) 773–785.

[17] Lian Zhigang, Xingsheng Gu, Bin Jiao, A novel particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan, Chaos, Solitons & Fractals 35 (5) (2008) 851–861.

[18] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[19] C. Koulamas, A new constructive heuristic for flow-shop scheduling problem, European Journal of Operational Research 105 (1998) 66–71.

[20] J.N. Gupta, A functional heuristic algorithm for flowshop scheduling problem, Operational Research Quarterly 22 (1) (1971) 39–47.

[21] M. Nawaz, E. Enscore Jr., I. Ham, A heuristics algorithm for the m-machine, n-job flowshop sequencing problem, Omega 11 (1) (1983) 91–95.

[22] F. van den Bergh, An analysis of particle swarm optimizers, PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[23] L. Lauriere, A language and a program for stating and solving combinatorial problems, Artificial Intelligence 10 (1) (1978) 29–127.

[24] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, European Journal of Operational Research 47 (1) (1990) 65–74.

[25] R. Ruiz, C. Maroto, J. Alcaraz, Two new robust genetic algorithms for the flowshop scheduling problem, OMEGA, the International Journal of Management Science 34 (2006) 461–476.

[26] I. Osman, C. Potts, Simulated annealing for permutation flow-shop scheduling, OMEGA, The International Journal of Management Science 17 (6) (1989) 551–557.

[27] M. Widmer, A. Hertz, A new heuristic method for the flow shop sequencing problem, European Journal of Operational Research 41 (2) (1989) 186–193.

[28] A.C. Nearchou, The effect of various operators on the genetic search for large scheduling problems, Int. J. Product. Economy 88 (2004) 191–203.

[29] T. Aldowaisan, A. Allahvedi, New heuristics for no-wait flowshops to minimize makespan, Computers and Operations Research 30 (8) (2003) 1219–1231.

[30] T. Murata, H. Ishibuchi, H. Tanaka, Genetic algorithms for flowshop scheduling problems, Computers and Industrial Engineering 30 (4) (1996) 1061–1071.

[31] B. Yagmahan, M.M. Yenisey, Ant colony optimization for multi-objective flow shop scheduling problem, Computers & Industrial Engineering 54 (3) (2008) 411–420.