RESEARCH ARTICLE

ScienceAsia 35 (2009): **89–94**

doi: 10.2306/scienceasia1513-1874.2009.35.089

Particle swarm optimization algorithm applied to scheduling problems

Pisut Pongchairerks

Industrial Engineering Programme, School of Manufacturing Systems and Mechanical Engineering, Sirindhorn International Institute of Technology, Thammasat University, Pathum Thani 12121, Thailand

e-mail: pisut@siit.tu.ac.th

Received 30 Sep 2008
Accepted 2 Feb 2009

ABSTRACT: This research introduces three heuristic algorithms for solving job-shop scheduling problems, job-shop scheduling problems with multi-purpose machines, and open-shop scheduling problems. All these algorithms are based on the particle swarm optimization algorithm, and generate solutions that belong to the class of parameterized active schedules via their specific decoding procedures. Comparison of the benchmark test results of the proposed algorithms with those of existent algorithms reveal that the proposed algorithms perform better in some instances.

KEYWORDS: job-shop, open-shop

INTRODUCTION

Scheduling involves the allocation of resources to activities over time. Most organizations must schedule resources on a recurrent basis and this creates considerable demand for good scheduling techniques. Since the mid 1950s, researchers have been advocating the use of formal optimization algorithms to find solutions to scheduling problems. Unfortunately, after fifty years of work, these methods can still only guarantee optimality for a very limited set of problems.

There are two main reasons for the limited success of traditional optimization algorithms to scheduling. First, most scheduling problems belong to the class of NP-hard problems. This class of problems is distinguished by rapid growth in the number of potential solutions with modest growth in the number of resources to be scheduled. The growth is so quick that even the fastest computer could not search through every potential solution to large-scale problems in a reasonable amount of time. Second, for many practical scheduling problems, it is difficult to capture the problem formulation in a closed-form mathematical expression. This difficulty is perhaps the reason why most scheduling is still done in an ad hoc manner. Because of these two difficulties, many researchers have thus turned their attention to population-based stochastic search methods, e.g., genetic algorithms [1–4], ant colony optimizations [5–8], and particle swarm optimizations [9], which are able to find near-optimal solutions within an acceptable computation time.

The main purpose of this research is to apply particle swarm optimization (PSO) to three types of scheduling problem. There are many variants of the PSO algorithm in the literature [9–12]. We will use the GLN-PSO$_c$ algorithm [10,11] since it enables the swarm to explore different parts of the search spaces simultaneously. This algorithm is an extension of the GLN-PSO algorithm, some solutions of which are generated via the crossover operator.

PSO is a generic algorithm. It must therefore be combined with a specific decoding procedure in order to generate solutions for a given problem. We introduce the GLN-PSO$_c$ with three decoding procedures, based on the random keys representation [13], to solve each scheduling problem. The proposed decoding procedure enables GLN-PSO$_c$ to generate the solutions which belong to the class of parameterized active schedules [14]. This class of schedules has played an important role as the efficient solution space for several genetic algorithms [14–17].

DESCRIPTION OF PROBLEMS

The job-shop scheduling problem (JSP) can be stated as follows. There are a set of n jobs J_i ($i = 1, \ldots, n$) and a set of m machines M_j ($j = 1, \ldots, m$). Each job J_i consists of an ordered set of m operations $O_{i1}, O_{i2}, \ldots, O_{im}$. The order of operations cannot be changed. Furthermore, operation O_{ij} must be processed by exactly one given machine during τ_{ij} time units without preemption, i.e., when the operation has been already started, it cannot be interrupted or temporarily stopped until finished. Also, each machine can handle only one job in a given time

period, and each job can be processed on only one machine in a given time period. The problem requires finding a feasible allocation of all operations to time intervals on the given machines such that some objective is optimized. Allocation of all operations to time intervals is known as scheduling. Here our objective is to minimize the completion time of the last given job.

The job-shop scheduling problem with multi-purpose machines (MPMJSP)[18] arises in connection with flexible manufacturing systems where the machines are equipped with different tools. It is possible to define the MPMJSP as an extension of the JSP where operation O_{ij} has to be processed by exactly one machine in a set of machines $A_{ij} \subseteq \{M_1, \ldots, M_m\}$ during τ_{ij} time units without preemption.

The open-shop scheduling problem (OSP) is a special case of the JSP where the operations constituting a job can be processed in any order. For the OSP let O_k, where $k = m(i-1)+j$, represent the operation of job J_i that must be processed by machine M_j during τ_k time units without preemption.

NOTATION AND TERMINOLOGY OF PSO ALGORITHM

A population or swarm is a set of K particles located in D-dimensional space. At iteration t (where $t = 1, \ldots, T$), particle i (where $i = 1, \ldots, K$) has a position $X_i(t) \equiv (x_{i1}(t), \ldots, x_{id}(t), \ldots, x_{iD}(t))$ and a velocity $V_i(t) \equiv (v_{i1}(t), \ldots, v_{id}(t), \ldots, v_{iD}(t))$. The position and velocity components satisfy $X_{\min} \leqslant x_{id}(t) \leqslant X_{\max}$ and $|v_{id}(t)| \leqslant V_{\max}$, respectively. The velocity $V_i(t+1)$ is the rate at which particle i moves from position $X_i(t)$ to position $X_i(t+1)$.

Each position $X_i(t)$ may directly or indirectly represent a solution of a specific problem. The objective function value of the solution of a specific problem decoded from the position $X_i(t)$ is denoted by $f(X_i(t))$. This is also known as the fitness value. In a minimization problem, X_i is 'better' than X_j if $f(X_i) < f(X_j)$. Here we only consider minimization problems since a maximization problem can be turned into a minimization problem by negating the fitness value.

The personal best position of particle i is the position at which $f(X_i)$ achieved its lowest value so far and is denoted by $P_i \equiv (p_{i1}, \ldots, p_{id}, \ldots, p_{iD})$. The global best position, $P_{\mathrm{g}} \equiv (p_{\mathrm{g}1}, \ldots, p_{\mathrm{g}d}, \ldots, p_{\mathrm{g}D})$, is the best position found by the swarm so far. The local best position for particle i is the best position found by particles in the n_{a}-adjacent neighbourhood of particle i and is denoted by $P_{\mathrm{l}i} \equiv (p_{\mathrm{l}i1}, \ldots, p_{\mathrm{l}id}, \ldots, p_{\mathrm{l}iD})$. For odd n_{a}, the adjacent neighbourhood of particle i contains the particles $i - (n_{\mathrm{a}}-1)/2, \ldots, i, \ldots, i + (n_{\mathrm{a}}-1)/2$ where K is added or subtracted from the particle index if it lies outside $1, \ldots, K$.

The near neighbour best position[19] is denoted by $P_{\mathrm{n}i} = (p_{\mathrm{n}i1}, \ldots, p_{\mathrm{n}id}, \ldots, p_{\mathrm{n}iD})$ in which $p_{\mathrm{n}id} = p_{jd}$ where, for each d, the value of $j \neq i$ is chosen so as to maximize the value of the fitness-distance ratio,

$$\gamma(j,i,d) = \frac{f(X_i) - f(P_j)}{|p_{jd} - x_{id}|}. \tag{1}$$

Hence, unlike the other best positions defined above, $P_{\mathrm{n}i}$ will in general not be a position of an existing particle.

DESCRIPTION OF GLN-PSO$_{\mathrm{c}}$ ALGORITHM

The GLN-PSO algorithm is an extension of the standard PSO algorithm using the global, local, and near neighbour best positions simultaneously in order to update the particle velocities[10,11]. It outperforms the standard version in terms of solution quality. One advantage of GLN-PSO over the standard PSO is that GLN-PSO uses multiple social learning structures which can reduce the rate of swarm clustering, and thus enable the swarm to explore different parts of the search spaces simultaneously.

In order to enhance the search performance of GLN-PSO in the search space based on a random keys representation[13], this study combines GLN-PSO with a crossover operator. The GLN-PSO combined with this crossover operator is henceforth called GLN-PSO$_{\mathrm{c}}$. The purpose of this modification is to maintain the diversity of the swarm to keep it from premature convergence to a local optimum.

At each iteration in the GLN-PSO$_{\mathrm{c}}$ algorithm, the position of each particle is updated by using either the traditional GLN-PSO equations or by performing a crossover between its current position and the global best position. For each particle, the type of update is determined probabilistically; the probability for performing a crossover is q_{c}.

The traditional GLN-PSO equations (applied with probability $1 - q_{\mathrm{c}}$) are

$$v_{id}(t+1) = \begin{cases} -V_{\max}, & a(t) < -V_{\max}, \\ a(t), & -V_{\max} \leqslant a(t) \leqslant V_{\max}, \\ V_{\max}, & a(t) > V_{\max}, \end{cases} \tag{2}$$

$$x_{id}(t+1) = \begin{cases} -X_{\max}, & b(t) < -X_{\max}, \\ b(t), & -X_{\max} \leqslant b(t) \leqslant X_{\max}, \\ X_{\max}, & b(t) > X_{\max}, \end{cases} \tag{3}$$

where

$$a(t) = w(t)v_{id}(t) + c_{\mathrm{p}}u(p_{id} - x_{id}(t)) + c_{\mathrm{g}}u(p_{\mathrm{g}d} - x_{id}(t)) + c_{\mathrm{l}}u(p_{\mathrm{l}id} - x_{id}(t)) + c_{\mathrm{n}}u(p_{\mathrm{n}id} - x_{id}(t)),$$

$$b(t) = x(t) + v_{id}(t+1),$$

in which $w(t)$ is known as the inertia weight, c_{p}, c_{g}, c_{l}, and c_{n} are constants (known as acceleration constants), and u represents a random number from the uniform distribution on the interval $[0, 1]$.

For the crossover procedure (applied with probability q_{c}),

$$v_{id}(t+1) = v_{id}(t) \tag{4}$$

$$x_{id}(t+1) = \begin{cases} x_{id}(t) & \text{if } u < q_{\mathrm{u}}, \\ p_{\mathrm{g}d} & \text{otherwise.} \end{cases} \tag{5}$$

where q_{u} is a constant satisfying $0 < q_{\mathrm{u}} < 1$.

Algorithm 1 GLN-PSO$_{\mathrm{c}}$ procedure
Step 1: Set $t = 1$. Initialize $X_i(1)$ and $V_i(1)$.
Step 2: Evaluate $f(X_i)$, then update P_i, P_{g}, $P_{\mathrm{l}i}$, $P_{\mathrm{n}i}$, and then obtain $X_i(t+1)$ and $V_i(t+1)$.
Step 3: If the stopping criterion is satisfied, stop. Otherwise, increase t by 1 and go to Step 2.

APPLICATION OF GLN-PSO$_{\mathrm{c}}$ TO SCHEDULING PROBLEMS

The GLN-PSO$_{\mathrm{c}}$ algorithm is a generic optimizer and is therefore applicable to many types of optimization problem. The problem-specific part is how to decode a particle position into a solution. Thus, in order to apply GLN-PSO$_{\mathrm{c}}$ to the three problems JSP, MPMJSP, and OSP, the specific solution decoding procedures are proposed to deal with each problem. To simplify the terms often used in this paper, let JSP-DECODE, MPM-DECODE, and OSP-DECODE represent the solution decoding procedures for JSP, MPMJSP, and OSP, respectively. Then, let JSP-PSO, MPM-PSO, and OSP-PSO represent the GLN-PSO$_{\mathrm{c}}$ using JSP-DECODE, MPM-DECODE, and OSP-DECODE, respectively.

In order to decode a particle position into a solution, the three mentioned decoding procedures use a random keys representation similar to that used in the genetic algorithm of Bean[13]. The random keys representation encodes a solution with random numbers where these numbers are used as sort keys to decode the solution. The important feature of the random keys representation is that all particle positions can represent feasible solutions without requiring any repair method.

The three decoding procedures JSP-DECODE, MPM-DECODE, and OSP-DECODE convert a particle position into a parameterized active schedule. All of these decoding procedures consist of two steps: (1) decoding a particle position into operation priorities, and (2) constructing a parameterized active schedule by using the predefined operation priorities. Each decoding procedure has its own distinct method for implementing these steps.

Decoding a particle position into operation priorities

In both JSP-DECODE and MPM-DECODE, a particle position is decoded into an operation-based permutation which is a representation of operation priorities. For an n-job, m-machine instance, the operation-based permutation[3] is a sequence of mn integers, where each job index appears in the permutation for exactly m times. The method for decoding a particle position, i.e., $X = (x_1, x_2, x_3, \ldots, x_{mn})$, into an operation-based permutation, i.e., $\pi = (\pi_1, \pi_2, \pi_3, \ldots, \pi_{mn})$, is presented in Algorithm 2.

Algorithm 2 To decode a particle position into an operation-based permutation.
Step 1: Sort the components of X into ascending order.
Step 2: Arrange the components of π in the same order as the X components sorted in Step 1.
Step 3: Let the first m reordered components of π equal 1, the second m components equal 2, and so on.

For instance, suppose in a 2-job, 2-machine problem $X = (0.2, 0.7, 0.8, 0.4)$. Then using Algorithm 2 we would obtain $\pi = (1, 2, 2, 1)$.

The operation-based permutation represents the operation priorities for constructing a parameterized active schedule. A component of π with a value i stands for job J_i. The jth occurrence of i in π refers to the jth operation of job J_i, that is, to O_{ij}. The priority of the operation is determined simply by the order of the components of π. The operation corresponding to the kth component of π has a higher priority than that corresponding to the $(k+1)$th component. Thus, referring to the above example, the interpretation of $\pi = (1, 2, 2, 1)$ is that the (decreasing) order of priority of the operations is O_{11}, O_{21}, O_{22}, O_{12}.

In OSP-DECODE, the operation priorities are determined from the particle position using the condition that if $x_d < x_b$ then the priority of O_d will be larger than that of O_b. For example, if $X = (0.1, 0.3, 0.7, 0.4)$ then the (descending) order of priorities is O_1, O_2, O_4, O_3.

Constructing a parameterized active schedule by using predefined operation priorities

All decoding procedures will schedule one operation at a time. An operation is schedulable if it has not yet been scheduled and all the operations which precede it have already been scheduled. For OSP-DECODE, the latter condition is not required, as OSP does not have operation precedence constraints. We let S_t denote the set of all the schedulable operations at stage t, and let Φ_t denote the partial schedule of the $(t-1)$ scheduled operations. We also introduce the input parameter $\delta \in [0,1]$ which is the bound on the length of time a machine is allowed to remain idle. A decoding procedure with $\delta = 0$ generates non-delay schedules whereas one with $\delta \to 1$ generates active schedules.

All decoding procedures will iterate for mn stages since there are mn operations. Note that in the following algorithms we use O_d to denote either O_k or O_{ij}.

Algorithm 3 To construct a parameterized active schedule of JSP-DECODE and OSP-DECODE.
Step 1: Let $t = 1$ and make the set Φ_1 empty.
Step 2: Find $\sigma^* = \min_{O_d \in S_t}\{\sigma_d\}$ and $\phi^* = \min_{O_d \in S_t}\{\phi_d\}$ where σ_d is the earliest time that operation O_d in S_t could be started and ϕ_d is the earliest time that it could be completed. Hence $\phi_d = \sigma_d + \tau_d$, where τ_d is the processing time of O_d.
Step 3: Select the highest priority operation $O^* \in S_t$ such that $\sigma_d \leqslant \sigma^* + \delta(\phi^* - \sigma^*)$.
Step 4: Create Φ_{t+1} by adding O^* to Φ_t.
Step 5: If a complete schedule is constructed, stop. Otherwise, increase t by 1 and go to Step 2.

Algorithm 4 To construct a parameterized active schedule of MPM-DECODE.
Step 1: Let $t = 1$ and make the set Φ_1 empty.
Step 2: Find σ^*, ϕ^*, and also the lowest index machine M^* on which ϕ^* occurs.
Step 3: Select the highest priority operation $O^* \in S_t$ such that O^* is in M^*, and $\sigma_d \leqslant \sigma^* + \delta(\phi^* - \sigma^*)$.
Step 4: Create Φ_{t+1} by adding O^* to Φ_t.
Step 5: If a complete schedule is constructed, stop. Otherwise, increase t by 1 and go to Step 2.

PERFORMANCE EVALUATION

We used the same parameter values for testing the performance of all three algorithms. The following parameter values were the same as those used in Ref. 11: $K = 40$, $n_\mathrm{a} = 7$, $c_\mathrm{p} = 0.5$, $c_\mathrm{g} = 0.5$, $c_\mathrm{l} = 1.5$, $c_\mathrm{n} = 1.5$, $V_\mathrm{max} = 0.25$, $X_\mathrm{max} = \infty$, and

$$w(t) = \begin{cases} 0.9 - 0.5(t-1)/999, & 1 \geqslant t \geqslant 1000, \\ 0.4, & t \geqslant 1000. \end{cases}$$

After experimenting with 25 combinations of values of q_c and δ (both ranging from 0.0 to 0.8 in steps of 0.2) we chose $q_\mathrm{c} = 0.2$ and $\delta = 0.4$ as these gave the best results. As in Ref. 17, we used $q_\mathrm{u} = 0.7$. It was also checked that this value was optimal when compared with nearby values. The initial population was given by $v_{id}(1) = 0$ and $x_{id}(1) = u$. The stopping criterion (see Algorithm 1) was if $t = T$ or a lower bound solution is met. We chose $T = 2000$ since the solutions rarely improve for t larger than this.

The three proposed algorithms were tested on benchmarks using a C# program under Windows on a 1.60 GHz Pentium M processor. For each benchmark, the result from the best of ten runs was taken as the final result.

Performance of JSP-PSO

To evaluate the performance of JSP-PSO, we used 36 well-known benchmark instances; ft06, ft10, ft20 from Fisher and Thompson[20], and the rest from Lawrence[21]. The results of the execution of JSP-PSO are compared to the optimal solutions and the best results taken from an efficient ant colony optimization[8] in Table 1. JSP-PSO can find the optimal solutions to 17 out of 33 instances, and it outperforms the ant colony optimization in 13 instances, and does worse than it in only 9 instances.

For an $m \times n$ instance, the average computation times in seconds (in parentheses) are as follows: 6×6 (0.1), 5×10 (10.5), 5×15 (0.1), 5×20 (7.4), 10×10 (39.2), 10×15 (71.1), 10×20 (126.2), 15×15 (157.2).

Performance of MPM-PSO

To test the performance of MPM-PSO, this study used 23 benchmark instances taken from the RDATA set of Jurisch[22]. These MPMJSP benchmark instances were modified versions of the standard JSP benchmark instances[21,22]. The results of MPM-PSO were compared to the best found solutions of a tabu search algorithm, namely NB1-1000, taken from Jurisch[22] (Table 2).

Table 2 demonstrates that MPM-PSO can generate the optimal solutions of only 5 out of 23 instances. Also, none of the optimal solutions found are of instances of large problems. However, it can be observed that the generated solution values are very close to the optimal solution values for almost all problem sizes, in particular 5×15 and 5×20. MPM-PSO performs better than the tabu search algorithm in 6 instances, and worse in 11 instances.

The average computation times are 6×6 (0.1), 5×10 (40.7), 5×15 (84.3), 5×20 (144.9), 10×10 (127.3).

Table 1 Performance evaluation of JSP-PSO on benchmark problems.

Instance	$m \times n$	Optimal Solution Value	Result from Ant[8]	Result from JSP-PSO
ft06	6×6	55	55	55*
ft10	10×10	930	**944**[a]	951
ft20	5×20	1165	**1178**	1191
la01	5×10	666	666	666*
la02	5×10	655	**658**	663
la03	5×10	597	603	603
la04	5×10	590	**590**	611
la05	5×10	593	593	593*
la06	5×15	926	926	926*
la07	5×15	890	890	890*
la08	5×15	863	863	863*
la09	5×15	951	951	951*
la10	5×15	958	958	958*
la11	5×20	1222	1222	1222*
la12	5×20	1039	1039	1039*
la13	5×20	1150	1150	1150*
la14	5×20	1292	1292	1292*
la15	5×20	1207	1240	**1207***
la16	10×10	945	977	**959**
la17	10×10	784	793	**784***
la18	10×10	848	848	848*
la19	10×10	842	860	**857**
la20	10×10	902	925	**910**
la21	10×15	1046	**1063**	1074
la22	10×15	927	954	**944**
la23	10×15	1032	1055	**1032***
la24	10×15	935	**954**	971
la25	10×15	977	1003	**987**
la26	10×20	1218	1308	**1224**
la27	10×20	1235	**1269**	1280
la29	10×20	1152	**1162**	1228
la30	10×20	1355	1411	**1355***
la36	15×15	1268	1334	**1307**
la37	15×15	1397	1457	**1456**
la38	15×15	1196	**1224**	1263
la40	15×15	1222	1269	**1251**

* indicates optimum solution value
[a] bold indicates algorithm beats the other

Table 2 Performance evaluation of MPM-PSO on benchmark problems.

Instance	Optimal Solution Value	Result from Tabu Search[22]	Result from MPM-PSO
ft06_r	47	47	47*
ft10_r	(707)[a]	737	**736**
ft20_r	(1026)	1028	**1026***
la01_r	(571)	**574**	578
la02_r	(530)	535	**534**
la03_r	(478)	**481**	482
la04_r	502	509	**507**
la05_r	457	**460**	577
la06_r	(800)	**801**	803
la07_r	(750)	**752**	753
la08_r	(766)	767	**766***
la09_r	(854)	859	**856**
la10_r	(805)	**806**	808
la11_r	1071	**1073**	1074
la12_r	(937)	937	937*
la13_r	1038	1039	1039
la14_r	1070	1071	1071
la15_r	(1091)	1093	1093
la16_r	717	**717**	752
la17_r	646	**646**	648
la18_r	666	**674**	691
la19_r	(703)	**725**	733
la20_r	756	756	756*

[a] value in parentheses represents the best known solution – no provable optimal solution exists

Performance of OSP-PSO

The performance of OSP-PSO was evaluated through the set of standard OSP test instances given by Taillard[23]. Table 3 compares the best solution found over ten runs from OSP-PSO to the best solution found by a particular genetic algorithm[4] for each instance. The results of the performance evaluation of OSP-PSO, based on Taillard's instances, are presented in Table 3.

OSP-PSO returns optimal solutions for 17 instances from the total of 30 instances. OSP-PSO performs better than the genetic algorithm in 14 instances and worse in 5 instances.

The average computation times are 5×5 (6.5), 7×7 (24.8), 10×10 (90.7).

CONCLUSIONS

This paper presented three algorithms, using GLN-PSO$_c$ as their framework, for solving the job-shop scheduling problem, the job-shop scheduling problem with multi-purpose machines, and the open-shop scheduling problem. Based on the computational experiments, the proposed algorithms are better than the existent algorithms on many instances, thus demonstrating that GLN-PSO$_c$ is a valid method for solving various types of scheduling problems. It could be applied to other hard-to-solve scheduling problems, such as flow-shop scheduling problem, by developing the decoding procedures which match those problems.

Table 3 Performance evaluation of OSP-PSO on benchmark problems

Instance	Optimal Solution Value	Result from Genetic algorithm [4]	Result from OSP-PSO
5×5_1[a]	300	301	**300***
5×5_2	262	263	**262***
5×5_3	323	335	**326**
5×5_4	310	316	**310***
5×5_5	326	330	**329**
5×5_6	312	312	312*
5×5_7	303	308	**303***
5×5_8	300	304	**301**
5×5_9	353	358	**354**
5×5_10	326	328	**326***
7×7_1	435	436	**435***
7×7_2	443	447	**444**
7×7_3	468	472	472
7×7_4	463	463	463*
7×7_5	416	417	**416***
7×7_6	451	455	**453**
7×7_7	422	426	**425**
7×7_8	424	424	424*
7×7_9	458	458	458*
7×7_10	398	398	398*
10×10_1	637	**637**	639
10×10_2	588	588	588*
10×10_3	598	**598**	601
10×10_4	577	577	577*
10×10_5	640	**640**	641
10×10_6	538	538	538*
10×10_7	616	616	616*
10×10_8	595	595	595*
10×10_9	595	**595**	597
10×10_10	596	**596**	597

[a] $m\times n\_a$ is the ath instance of a m-machine, n-job task

REFERENCES

1. Croce F, Tadei R, Volta G (1995) A genetic algorithm for the job shop problem. *Comput Oper Res* **22**, 15–24.
2. Gen M, Cheng R (1997) *Genetic Algorithms and Engineering Design*, John Wiley & Sons, New York.
3. Gen M, Tsujimura Y, Kubota E (1994) Solving job-shop scheduling problem using genetic algorithms. In: The 16th International Conference on Computers and Industrial Engineering, Japan, pp 576–9.
4. Prins C (2000) Competitive genetic algorithms for the open-shop scheduling problem. *Math Meth Oper Res* **52**, 389–411.
5. Blum C (2005) Beam-ACO – hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Comput Oper Res* **32**, 1565–91.
6. Blum C, Roli A, Dorigo M (2001) The hyper-cube framework for ant colony optimization. In: Proceedings of the Fourth Meta-heuristics International Conference, Porto, pp 399–403.
7. Colorni A, Dorigo M, Maniezzo V, Trubian M (1994) Ant system for job-shop scheduling. *Belg J Oper Res Statist Comput Sci* **34**, 39–53.
8. Udomsakdigool A, Kachitvichyanukul V (2008) Multiple colony ant algorithm for job-shop scheduling problem. *Int J Prod Res* **46**, 4155–75.
9. Kennedy J, Eberhart RC, Shi Y (2001) *Swarm Intelligence*, Morgan Kaufmann, San Francisco.
10. Pongchairerks P, Kachitvichyanukul V (2005) A non-homogeneous particle swarm optimization with multiple social structures. In: Proceedings of International Conference on Simulation and Modeling, Thailand, A5-02.
11. Pongchairerks P, Kachitvichyanukul V (2006) Particle swarm optimization algorithm with multiple social learning structures. In: Proceedings of the 36th CIE Conference on Computers & Industrial Engineering, Taiwan, pp 1556–67.
12. Clerc M (2006) *Particle Swarm Optimization*. ISTE Ltd, California.
13. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. *ORSA J Comput* **6**, 154–60.
14. Bierwirth C, Mattfeld DC (1999) Production scheduling and rescheduling with genetic algorithms. *Evol Comput* **7**, 1–17.
15. Storer RH, Wu SD, Vaccari R (1992) New search spaces for sequencing problems with application to job shop scheduling. *Manag Sci* **38**, 1495–509.
16. Mattfeld DC, Bierwirth C (2004) An efficient genetic algorithm for job shop scheduling with tardiness objectives. *Eur J Oper Res* **155**, 616–30.
17. Gonçalves JF, Mendes JJM, Resende MGC (2005) A hybrid genetic algorithm for the job shop scheduling problem. *Eur J Oper Res* **167**, 77–95.
18. Brucker P, Schlie R (1990) Job-shop scheduling with multi-purpose machines. *Computing* **45**, 369–75.
19. Veeramachaneni K, Peram T, Mohan C, Osadciw LA (2003) Optimization using particle swarms with near neighbor interactions. In: *Genetic and Evolutionary Computation – GECCO 2003*, Springer, Berlin, pp 110–21.
20. Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) *Industrial Scheduling*, Prentice Hall, Englewood Cliffs, pp 225–51.
21. Lawrence S (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Tech Rep, GSIA, Carnegie Mellon Univ.
22. Jurisch B (1992) Scheduling jobs in shops with multi-purpose machines. PhD thesis, Universität Osnabrück.
23. Taillard ED (1993) Benchmarks for basic scheduling problems. *Eur J Oper Res* **64**, 278–85.