Expert Systems with Applications 37 (2010) 1065–1070

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A multi-objective PSO for job-shop scheduling problems

D.Y. Sha [a], Hsing-Hung Lin [b,*]

[a] Department of Industrial Engineering and System Management, Chung Hua University, Hsin Chu, Taiwan, ROC
[b] Department of Industrial Engineering and Management, National Chiao Tung University, Hsin Chu, Taiwan, ROC

ARTICLE INFO

Keywords:
Job-shop scheduling
Particle swarm optimization
Multiple objectives

ABSTRACT

Most previous research into the job-shop scheduling problem has concentrated on finding a single optimal solution (e.g., makespan), even though the actual requirement of most production systems requires multi-objective optimization. The aim of this paper is to construct a particle swarm optimization (PSO) for an elaborate multi-objective job-shop scheduling problem. The original PSO was used to solve continuous optimization problems. Due to the discrete solution spaces of scheduling optimization problems, the authors modified the particle position representation, particle movement, and particle velocity in this study. The modified PSO was used to solve various benchmark problems. Test results demonstrated that the modified PSO performed better in search quality and efficiency than traditional evolutionary heuristics.

1. Introduction

The job-shop scheduling problem (JSP) has been studied for more than 50 years in both academic and industrial environments. Jain and Meeran (1999) provided a concise overview of JSPs over the last few decades and highlighted the main techniques. The JSP is the most difficult class of combinational optimization. Garey, Johnson, and Sethi (1976) demonstrated that JSPs are non-deterministic polynomial-time hard (NP-hard); hence we cannot find an exact solution in a reasonable computation time. The single-objective JSP has attracted wide research attention. Most studies of single-objective JSPs result in a schedule to minimize the time required to complete all jobs, i.e., to minimize the makespan (C_{max}). Many approximate methods have been developed to overcome the limitations of exact enumeration techniques. These approximate approaches include simulated annealing (SA) (Lourenço, 1995), tabu search (Nowicki & Smutnicki, 1996; Pezzella & Merelli, 2000; Sun, Batta, & Lin, 1995) and genetic algorithms (GA) (Bean, 1994; Gonçalves, Mendes, & Resende, 2005; Kobayashi, Ono, & Yamamura, 1995; Wang & Zheng, 2001). However, real-world production systems require simultaneous achievement of multiple objective requirements. This means that the academic concentration of objectives in the JSP must been extended from single to multiple. Recent related JSP research with multiple objectives is summarized as below.

Ponnambalam, Ramkumar, and Jawahar (2001) has offered a multi-objective GA to derive optimal machine-wise priority dispatching rules for resolving job-shop problems with objective functions that consider minimization of makespan, total tardiness, and total machine idle time. Ponnambalam's multi-objective genetic algorithm (MOGA) has been tested with various published benchmarks, and is capable of providing optimal or near-optimal solutions. A Pareto front provides a set of best solutions to determine the tradeoffs between the various objects, and good parameter settings and appropriate representations can enhance the behavior of an evolution algorithm. Esquivel, Ferrero, and Gallard (2002) studied the influence of distinct parameter combinations as well as different chromosome representations. Initial results showed that:

(i) larger numbers of generations favor the building of a Pareto front because the search process does not stagnate, even though it may be rather slow,
(ii) Multi-recombination helps to speed the search and to find a larger set size when seeking the Pareto optimal set, and
(iii) operation-based representation is better than priority-list and job-based representation selected for contrast under recombination methods.

The Pareto archived simulated annealing (PASA) method, a meta-heuristic procedure based on the SA algorithm, was developed by Suresh and Mohanasndaram (2006) to find non-dominated solution sets for the JSP with the objectives of minimizing the makespan and the mean flow time of jobs. The superior performance of the PASA can be attributed to the mechanism it uses to accept the candidate solution. Candido, Khator, and Barcia (1998) addressed JSPs with numbers of more realistic constraints, such as jobs with several subassembly levels, alternative processing plans for parts and alternative resources of operations, and the

* Corresponding author. Tel.: +886 937808216.
E-mail address: hsinhung@gmail.com (H.-H. Lin).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.06.041

requirement for multiple resources to process an operation. The robust procedure worked well in all problem instances and proved to be a promising tool for solving more realistic JSPs. Lei and Wu (2006) first designed a crowding-measure-based multi-objective evolutionary algorithm (CMOEA) makes use of the crowding-measure to adjust the external population and assign different fitness for individuals. Compared to the strength Pareto evolutionary algorithm, CMOEA performs well in job-shop scheduling with two objectives including minimization of makespan and total tardiness.

One of the latest evolutionary techniques for unconstrained continuous optimization is particle swarm optimization (PSO) proposed by Kennedy and Eberhart (1995). PSO has been successfully used in different fields due to its ease of implementation and computational efficiency. Even so, application of PSO to the combination optimization problem is rare. Coello, Plido, and Lechga (2004) provided an approach in which Pareto dominance is incorporated into PSO to allow the heuristic to handle problems with several object functions. The algorithm uses a secondary repository of particles to guide particle flight. That approach was validated using several test functions and metrics drawn from the standard literature on evolutionary multi-objective optimization. The results show that the approach is highly competitive. Liang, Ge, Zho, and Guo (2005) invented a novel PSO-based algorithm for JSPs. That algorithm effectively exploits the capability of distributed and parallel computing systems, with simulation results showing the possibility of high-quality solutions for typical benchmark problems. Lei (2008) presented a PSO for the multi-objective JSP to minimize makespan and total job tardiness simultaneously. Job-shop scheduling can be converted into a continuous optimization problem by constructing the corresponding relationship between a real vector and a chromosome obtained using the priority rule-based representation method. The global best position selection is combined with crowding-measure-based archive maintenance to design a Pareto archive PSO. That algorithm is capable of producing a number of high-quality Pareto optimal scheduling plans.

Hybrid algorithms that combine different approaches to build on their strengths have led to another branch of research. Wang and Zheng (2001) combined GA with SA in a hybrid framework, in which the GA was introduced to present a parallel search architecture, and SA was used to increase the probability of escape from local optima at high temperatures. Computer simulation results showed that the hybrid strategy was very effective and robust, and could find optima for almost all benchmark instances. Xia and Wu (2005) developed an easily implemented approach for the multi-objective flexible JSP based on the combination of PSO and SA. They demonstrated that their proposed algorithm was a viable and effective approach to the multi-objective flexible JSP, especially for large-scale problems. Ripon (2007) extended the idea in the jumping genes genetic algorithm, a hybrid approach capable of searching for near-optimal and non-dominated solutions with better convergence by simultaneously optimizing criteria.

Previous literature indicates that there has been little study of the JSP with multiple objectives. In this study, we use a new evolutionary PSO technique to solve the JSP with multiple objectives.

2. Job-shop scheduling problem

A typical JSP can be formulated as follows. There are n jobs to be processed through m machines. Each job must pass through each machine once and only once. Each job should be processed through the machines in a particular order, and there are no precedence constraints among the different job operations. Each machine can perform only one job at a time, and it cannot be interrupted. In addition, the operation time is fixed and known in advance. The objective of the JSP is to find a schedule to minimize the time required to complete all jobs, that is, to minimize the makespan C_{max}. In this study, we attempt to attain the three objectives (i.e., minimizing makespan, machine idle time, and total tardiness) simultaneously. We formulate the multi-objective JSP using the following notation:

n is the total number of jobs to be scheduled,
m is the total number of machines in the process,
$t(i,j)$ is the processing time for job i on machine j $(i = 1,2,\ldots n),\ (j = 1,2,\ldots m)$,
L_i is the lateness of job i,
$\{\pi_1, \pi_2, \ldots, \pi_n\}$ is the permutation of jobs.

The objectives considered in this paper are formulated as follows:

Completion time (makespan) $C(\pi, j)$

$$C(\pi_1, 1) = t(\pi_1, 1) \tag{1}$$

$$C(\pi_i, 1) = C(\pi_{i-1}, 1) + t(\pi_i, 1) \quad i = 2, \ldots, n \tag{2}$$

$$C(\pi_1, j) = C(\pi_1, j-1) + t(\pi, j) \quad j = 2, \ldots, m \tag{3}$$

$$C(\pi_i, j) = \max C(\pi_{i-1}, j), C(\pi_i, j-1)\} + t(\pi_i, j) \quad i = 2, \ldots, n; \quad j = 2, \ldots, m \tag{4}$$

$$\text{Makespan}, f_{C\max} = C(\pi_n, m) \tag{5}$$

$$\text{Total tardiness}, f_{\text{total tardiness}} = \sum_{i=1}^{n} \max[0, L_i] \tag{6}$$

$$\text{Total idle time}, f_{\text{total idle time}} = \{C(\pi_1, j-1) + \sum_{i=2}^{n} \{\max\{C(\pi_i, j-1) - C(\pi_{i-1}, j), 0\}\} | j = 2 \ldots m\} \tag{7}$$

3. PSO background

PSO is based on observations of the social behavior of animals, such as birds in flocks or fish in schools, as well as on swarm theory. The population consisting of individuals or particles is initialized randomly. Each particle is assigned with a randomized velocity according to its own movement experience and that of the rest of the population. The relationship between the swarm and particles in PSO is similar to the relationship between the population and chromosomes in a GA.

In PSO, the problem solution space is formulated as a search space. Each particle position in the search space is a correlated solution to the problem. Particles cooperate to determine the best position (solution) in the search space (solution space).

Suppose that the search space is D-dimensional and there are ρ particles in the swarm. Particle i is located at position $X^i = \{x_1^i, x_2^i, \ldots, x_D^i\}$ and has velocity $V^i = \{v_1^i, v_2^i, \ldots, v_D^i\}$, where $i = 1, 2, \ldots, \rho$. Based on the PSO algorithm, each particle move towards its own best position (*pbest*), denoted as $Pbest^i = \{pbest_1^i, pbest_2^i, \ldots, pbest_n^i\}$, and the best position of the whole swarm (*gbest*) is denoted as $Gbest = \{gbest_1, gbest_2, \ldots, gbest_n\}$ with each iteration. Each particle changes its position according to its velocity, which is randomly generated toward the *pbest* and *gbest* positions. For each particle r and dimension s, the new velocity v_s^r and position x_s^r of particles can be calculated by the following equations:

$$v_s^r(\tau) = w \times v_s^r(\tau - 1) + c_1 \times rand_1 \times [pbest_s^r(\tau - 1) - x_s^r(\tau - 1)] + c_2 \times rand_2 \times [gbest_s^r(\tau - 1) - x_s^r(\tau - 1)] \tag{8}$$

$$x_s^r(\tau) = x_s^r(\tau - 1) + v_s^r(\tau - 1) \tag{9}$$

In Eqs. (8) and (9), τ is the iteration number. The inertial weight w is used to control exploration and exploitation. A large w value keeps the particles moving at high velocity and prevents them from becoming trapped in local optima. A small w value ensures a low particle velocity and encourages particles to exploit the same search area. The constants c_1 and c_2 are acceleration coefficients to determine whether particles prefer to move closer to the *pbest* or *gbest* positions. The $rand_1$ and $rand_2$ are two independent random numbers uniformly distributed between 0 and 1. The termination criterion of the PSO algorithm includes a maximum number of generations, a designated value of *pbest*, and lack of further improvement in *pbest*. The standard PSO process is outlined as follows:

Step 1: Initialize a population of particles with random positions and velocities in a D-dimensional search space.
Step 2: Update the velocity of each particle using Eq. (8).
Step 3: Update the position of each particle using Eq. (9).
Step 4: Map the position of each particle into the solution space and evaluate its fitness value according to the desired optimization fitness function. Simultaneously update the *pbest* and *gbest* positions if necessary.
Step 5: Loop to Step 2 until the termination criterion is met, usually after a sufficient good fitness or a maximum number of iterations.

The original PSO was designed for a continuous solution space. We must modify the PSO position representation, particle velocity, and particle movement so they work better with combinational optimization problems. These changes are described in next section.

4. Proposed method

There are four types of feasible schedules in JSPs, including inadmissible, semi-active, active, and non-delay. The optimal schedule is guaranteed to be an active schedule. We can decode a particle position into an active schedule employing Giffler and Thompson (1960) heuristic. There are two different representations of particle position associated with a schedule. The results of Zhang, Li, Li, and Hang (2005) demonstrated that permutation-based position representation outperforms priority-based representation. While choosing to implement permutation-based position presentation, we must also adjust the particle velocity and particle movement. In addition, we also propose the maintenance of Pareto optima and a diversification procedure to achieve better performance.

4.1. Position representation

In this study, we randomly generated a group of particles (solutions) represented by a permutation sequence that is an ordered list of operations. For an n-job m-machine problem, the position of particle k can be represented by an $m \times n$ matrix, i.e., $X^k = \begin{bmatrix} x^k_{11} & x^k_{12} & \dots & x^k_{1n} \\ x^k_{21} & x^k_{22} & \dots & x^k_{2n} \\ \vdots & \vdots & & \vdots \\ x^k_{m1} & x^k_{m2} & \dots & x^k_{mn} \end{bmatrix}$, where x^k_{ij} denotes the priority of operation o_{ij}, which means the operation of job j that must be processed on machine i.

The Giffler and Thompson (G&T) algorithm is briefly described below.

Notation:

(i,j) is the operation of job j that must be processed on machine i.
S is the partial schedule that contains scheduled operations.
Ω is the set of operations that can be scheduled.
$s_{(i,j)}$ is the earliest time at which operation (i,j) belonging to Ω can be started.
$p_{(i,j)}$ is the processing time of operation (i,j).
$f_{(i,j)}$ is the earliest time at which operation (i,j) belonging to Ω can be finished, $f_{(i,j)} = s_{(i,j)} + p_{(i,j)}$.

G&T algorithm:

Step 1: Initialize $S = \phi$; Ω to contain all operations without predecessors.
Step 2: Determine $f^* = \min_{(i,j)\in\Omega}\{f_{(i,j)}\}$ and the machine m^* on which f^* can be realized.
Step 3:
(1) Identify the operation set $(i',j') \in \Omega$ such that (i',j') requires machine m^*, and $s_{(i',j')} < f^*$
(2) Choose (i,j) from the operation set identified in Step 3(1) with the largest priority.
(3) Add (i,j) to S.
(4) Assign $s_{(i,j)}$ as the starting time of (i,j).
Step 4: If a complete schedule has been generated, stop. Otherwise, delete (i,j) from Ω, include its immediate successor in Ω, and then go to Step 2.

Tables 1 and 2 shows the mechanism of the G&T algorithm using a 2×2 example. The position of particle k is $X^k = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

Initialization

Step 1: $S = \phi$; $\Omega = \{(1,1),(2,2)\}$.

Iteration 1

Step 2: $s_{(1,1)} = 0, s_{(2,2)} = 0, f_{(1,1)} = 5, f_{(2,2)} = 4; f^* = min\{f_{(1,1)}, f_{(2,2)}\} = 4, m^* = 2$.
Step 3: Identify the operation set {(2,2)}; choose operation (2,2) that has the largest priority, and add it into schedule S.
Step 4: Update $\Omega = \{(1,1),(1,2)\}$; go to Step 2.

Iteration 2

Step 2: $s_{(1,1)} = 0, s_{(1,2)} = 4, f_{(1,1)} = 5, f_{(1,2)} = 7; f^* = min\{f_{(1,1)}, f_{(1,2)}\} = 5, m^* = 1$.
Step 3: Identify the operation set {(1,1), (1,2)}; choose operation (1,2) that has the largest priority, and add it into schedule S.
Step 4: Update $\Omega = \{(1,1)\}$; go to Step 2.

Iteration 3

Step 2: $s_{(1,1)} = 7, f_{(1,1)} = 12; f^* = min\{f_{(1,1)}\} = 12, m^* = 1$.
Step 3: Identify the operation set {(1,1)}; choose operation (1,1) that has the largest priority, and add it into schedule S.
Step 4: Update $\Omega = \{(2,1)\}$; go to Step 2.

Table 1
2×2 Example.

Jobs	Machine sequence	Processing times
1	1, 2	$p_{(1,2)}$=5; $p_{(2,1)}$=4
2	2, 1	$p_{(2,2)}$=4; $p_{(1,2)}$=3

Table 2
Comparison of MOGA and MOPSO with three objectives.

Benchmark	N	m	Makespan (MOGA)	Makespan (MOPSO)	% Deviation	Total idle time (MOGA)	Total idle time (MOPSO)	% Deviation	Total tardiness (MOGA)	Total tardiness (MOPSO)	% Deviation
abz5	10	10	1587	1338	0	8097	3978	0	1948	611	0
abz6	10	10	1369	1046	0	7744	2937	0	1882	339	0
ft10	10	10	1496	1045	0	9851	1999	0	3459	1534	0
la16	10	10	1452	1040	0	9169	2718	0	1127	1417	0.25732
la17	10	10	1172	889	0	7044	3365	0	1779	53	0
la19	10	10	1251	938	0	7164	2796	0	1581	733	0
la20	10	10	1419	985	0	8745	2883	0	1451	407	0
orb01	10	10	1704	1181	0	11631	3909	0	3052	191	0
orb02	10	10	1284	1029	0	7585	3539	0	1565	137	0
orb03	10	10	1643	1114	0	11138	3788	0	4140	247	0
orb04	10	10	1543	1122	0	9802	3921	0	4951	221	0
orb05	10	10	1323	1013	0	8322	3727	0	2195	30	0
orb06	10	10	1645	1144	0	10836	3478	0	2601	0	0
orb07	10	10	583	302	0	3423	1381	0	699	0	0
orb08	10	10	1340	1000	0	8840	3542	0	3498	253	0
orb09	10	10	1462	1044	0	9439	4224	0	2029	0	0
orb10	10	10	1382	1077	0	8271	4177	0	1806	0	0
la01	10	5	1256	709	0	3431	571	0	3324	721	0
la02	10	5	1066	713	0	2687	573	0	2081	425	0
la03	10	5	821	671	0	1722	633	0	1926	373	0
la04	10	5	861	631	0	1798	557	0	3194	673	0
la05	10	5	893	593	0	2182	473	0	1716	736	0
ft06	6	6	76	56	0	259	100	0	31	3	0

Iteration 4

Step 2: $s_{(2,1)} = 12, f_{(2,1)} = 16; f^* = min\{f_{(2,1)}\} = 16, m^* = 2$.
Step 3: Identify the operation set $\{(2,1)\}$; choose operation $(2,1)$ that has the largest priority, and add it into schedule S.
Step 4: A complete schedule has been generated, so stop the process.

The proposed PSO differs from the original PSO in the information stored in the *pbest* and *gbest* solutions. While the original PSO keeps the best positions found so far, the proposed PSO maintains the best schedule generated by the G&T algorithm. In the previous example, the schedule S^k rather than the position X^k is retained in the *pbest* and *gbest* solutions, where S^k is $\begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}$. The movement of particles is modified in accordance with the representation of particle position based on the insertion operator.

4.2. Particle velocity

The original PSO velocity concept assumes that each particle moves according to the velocity determined by the distance between the previous position of the particle and the *gbest* (*pbest*) solution. The two major purposes of the particle velocity are to keep the particle moving toward the *gbest* and *pbest* solutions, and to maintain inertia to prevent particles from becoming trapped in local optima.

In the proposed PSO, we concentrate on preventing particles from becoming trapped in local optima rather than moving them toward the *gbest* (*pbest*) solution. If the priority value is increased or decreased by the present velocity in the current iteration, we keep the priority value increasing or decreasing at the beginning of the next iteration with probability w, which is the inertial weight in PSO. The larger the value of w, the more the iteration priority value keeps increasing or decreasing, and the more the difficult it is for the particle to return to its current position. For an n-job problem, the velocity of particle k can be represented as

$$V^k = [v_1^k v_2^k \dots v_n^k],\ v_i^k \in \{-1, 0, 1\},$$

where v_i^k is the velocity of j_i of particle k

The initial velocity of particles is generated randomly. Instead of considering the distance from x_i^k to $pbest_i^k(gbest_i)$, our PSO considers whether the value of x_i^k is larger or smaller than $pbest_i^k(gbest_i)$. If x_i^k decreases in the present iteration, this mean that $pbest_i^k(gbest_i)$ is smaller than x_i^k and x_i^k is set moving toward $pbest_i^k(gbest_i)$ by letting $v_i^k \leftarrow -1$. Therefore, in the next iteration, $x_i^k$is kept decreasing by one (i.e., $x_i^k \leftarrow x_i^k - 1$) with probability w. Conversely, if x_i^k increases in this iteration, then $pbest_i^k(gbest_i)$ is larger than x_i^k, and x_i^k is set moving toward $pbest_i^k(gbest_i)$ by setting $v_i^k \leftarrow 1$. Therefore, in the next iteration, x_i^k is kept increasing by one (i.e., $x_i^k \leftarrow x_i^k + 1$) with probability w.

The inertial weight w influences the velocity of the particles in the PSO. We randomly update velocities at the beginning of the iteration. For each particle k and operation j_i, if v_i^k does not equal to 0, v_i^k will be set to 0 with probability $(1 - w)$. This forces x_i^k to stop increasing or decreasing continuously in this iteration with probability $(1 - w)$ while x_i^k keeps increasing or decreasing.

4.3. Particle movement

The particle movement is based on the swap operator proposed by Sha and Hsu (2006) and Sha and Hsu (2008).

Notation:

x_i^k is the schedule list at machine i of particle k.
$pbest_i^k$ is the schedule list at machine i of the kth *pbest* solution.
$gbest_i$ is the schedule list at machine i of the *gbest* solution.
c_1 and c_2 are constants between 0 and 1 such that $c_1 + c_2 \leqslant 1$.

The swap procedure occurs as shown below.

Step 1: Randomly choose a position ζ from x_i^k.
Step 2: Mark the job on position ζ of x_i^k by Λ_1.
Step 3: If the random number $rand < c_1$ then seek the position of Λ_1 in $pbest_i^k$; otherwise, seek the position of Λ_1 in $gbest_i$. Denote the position that has been found in $pbest_i^k$ or $gbest_i$ by ζ', and the job in position ζ' of x_i^k by Λ_2.
Step 4: If Λ_2 has been denoted, $v_{ij_1}^k = 0$, and $v_{ij_2}^k = 0$, then swap Λ_1 and Λ_2 in x_i^k, $v_{ij_1}^k \leftarrow 1$.
Step 5: If all the positions of x_i^k have been considered, then stop. If not, and if $\zeta < n$, then $\zeta \leftarrow \zeta + 1$; otherwise, $\zeta \leftarrow 1$. Go to Step 2.

For example, consider the 6-job problem where $x_i^k = [4\ 2\ 1\ 3\ 6\ 5], pbest_i^k = [1\ 5\ 4\ 2\ 6\ 3], gbest_i = [3\ 2\ 6\ 4\ 5\ 1], v_i^k = [0\ 0\ 1\ 0\ 0\ 0], c_1 = 0.6$, and $c_2 = 0.2$.

Step 1: The position of x_i^k is randomly chosen: $\zeta = 3$.
Step 2: The job in the 3rd position of x_i^k is job 1, i.e., $\Lambda_1 = 1$.
Step 3: A random number *rand* is generated; assume *rand*=0.7. Since $rand > c_1$, we compare each position of $gbest_i$ with Λ_1 and the matched position $\zeta' = 6$. The job in the 6th position of x_i^k is job 5, i.e., $\Lambda_2 = 5$.
Step 4: Since $v_{i4}^k = 0$ and $v_{i5}^k = 0$, swap jobs 1 and 5 in x_i^k so $x_i^k = [4\ 2\ 5\ 3\ 6\ 1]$. Then let $v_{i4}^k \leftarrow 1$ and $v_i^k = [0\ 0\ 1\ 1\ 0\ 0]$.
Step 5: Let $\zeta \leftarrow 4$ and go to Step 2. Repeat the process until all positions of x_i^k have been considered.

4.4. Diversification strategy

If all the particles have the same non-dominated solutions, they will be trapped in local optima. To prevent this from happening, a diversification strategy is proposed to keep the non-dominated solutions different. Once any new solution is generated by particles, the non-dominating solution set will be updated in these three situations:

(i) If the solution of the particle dominates the *gbest* solution, assign the particle solution to the *gbest*.
(ii) If the solution of the particle equals to any solution in the non-dominated solution set, replace the non-dominated solution with the particle solution.
(iii) If the solution of the particle is dominated by the worst non-dominated solution and not equal to any non-dominated solution, set the worst non-dominated solution equal to the particle solution.

5. Computational results

The proposed multi-objective PSO (MOPSO) algorithm was tested on benchmark problems obtained from the OR-Library (Beasley, 1990; Taillard, 1993). The program was coded in Visual C++ and run 40 times on each problem on a Pentium 4 3.0-GHz computer with 1 GB of RAM running Windows XP. During the pilot experiment, we used four swarm sizes N (10, 30, 60, and 80) to test the algorithm. The outcome of $N = 80$ was best, so that value was used in all further tests. Parameters c_1 and c_2 were tested at various values in the range 0.1–0.7 in increments of 0.2. The inertial weight w was reduced from $w_{max}$to w_{min} during iterations, where w_{max} was set to 0.5, 0.7, and 0.9, and w_{min} was set to 0.1, 0.3, and 0.5. The combination of $c_1 = 0.7, c_2 = 0.1, w_{max} = 0.7$ and $w_{min} = 0.3$ gave the best results. The maximum iteration limit was set to 60 and the maximum archive size was set to 80.

The MOGA proposed by (Pezzella & Merelli, 2000) was chosen as a baseline against which to compare the performance of our PSO algorithm. The objectives considered in the MOGA algorithm are minimization of makespan, minimization of total tardiness, and minimization of machine idle time. The MOGA methodology is based on the machine-wise priority dispatching rule (pdr) and the G&T procedure (Giffler & Thompson, 1960). The each gene represents a pdr code. The G&T procedure was used to generate an active feasible schedule. The MOGA fitness function is the weighted sum of makespan, total tardiness, and total idle time of machines with random weights.

The computation results showed that the relative error of the solution for C_{max} and total idle time determined by the proposed MOPSO was better in 23 out of 23 problems than the MOGA. In 22 of the 23 problems, the proposed PSO performed better for the solution considering total tardiness. Overall, the proposed MOPSO was superior to the MOGA in solving the JSP with multiple objectives.

6. Conclusion

While there has been a large amount of research into the JSP, most of this has focused on minimizing the maximum completion time (i.e., makespan). There exist other objectives in the real-world, such as the minimization of machine idle time that might help improve efficiency and reduce production costs. PSO, inspired by the behavior of birds in flocks and fish in schools, has the advantages of simple structure, easy implementation, immediate accessibility, short search time, and robustness. However, few applications of PSO to multi-objective JSPs can be found in the literature. Therefore, we presented a MOPSO method for solving the JSP with multiple objectives, including minimization of makespan, total tardiness, and total machine idle time.

The original PSO was proposed for continuous optimization problems. To make it suitable for job-shop scheduling (i.e., a combinational problem), we modified the representation of particle position, particle movement, and particle velocity. We also introduced a mutation operator and used a diversification strategy. The results demonstrated that the proposed MOPSO could obtain more optimal solutions than the MOGA. The relative error ratios of each problem scenario in our MOPSO algorithm were less than in the MOGA. The performance measure results also revealed that the proposed MOPSO algorithm outperformed MOGA in simultaneously minimizing makespan, total tardiness, and total machine idle time.

We will attempt to apply MOPSO to other shop scheduling problems with multiple objectives in future research. Other possible topics for further study include the modification of the particle position representation, particle movement, and particle velocity. In addition, issues related to Pareto optimization, such as solution maintenance strategy and performance measurement, merit future investigation.

Acknowledgement

This study was supported by a grant from the National Science Council of Taiwan (NSC-96-2221-E-216-052MY3).

Appendix A

Pseudo-code of the PSO for the multi-objective JSP is as follows.
Initialize a population of particles with random positions.
for each particle k ***do***

Evaluate X^k(the position of particle k)
Save the $pbest^k$ to optimal solution set S

end for
Set *gbest* solution equal to the best $pbest^k$
repeat

Updates particles velocities
for each particle k ***do***
Move particle k

Evaluate X^k
Update *gbest*, *pbest*, and S

end for

until maximum iteration limit is reached

References

Bean, J. (1994). Genetic algorithms and random keys for sequencing and optimization. *Operations Research Society of America (ORSA) Journal on Computing, 6*, 154–160.

Beasley, J. E. (1990). OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society, 41*(11), 1069–1072.

Candido, M. A. B., Khator, S. K., & Barcia, R. M. (1998). A genetic algorithm based procedure for more realistic job shop scheduling problems. *International Journal of Production Research, 36*(12), 3437–3457.

Coello, C. A., Plido, G. T., & Lechga, M. S. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation, 8*(3), 256–278.

Esquivel, S. C., Ferrero, S. W., & Gallard, R. H. (2002). Parameter settings and representations in Pareto-based optimization for job shop scheduling. *Cybernetics and Systems: An international Journal, 33*, 559–578.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*, 117–129.

Giffler, J., & Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research, 8*, 487–503.

Gonçalves, J. F., Mendes, J. J. M., & Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research, 167*(1), 77–95.

Jain, A. S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research, 113*, 390–434.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks 1995* (pp. 1942–1948).

Kobayashi, S., Ono, I. & Yamamura, M. (1995). An efficient genetic algorithm for job shop scheduling problems. In L.J. Eshelman (Ed.), *Proceedings of the sixth international conference on genetic algorithms* (pp. 506–511). San Francisco, CA: Morgan Kaufman Publishers.

Lei, D., & Wu, Z. (2006). Crowding-measure-based multi-objective evolutionary algorithm for job shop scheduling. *International Journal of Advanced Manufacturing Technology, 30*, 112–117.

Lei, D. (2008). A Pareto archive particle swarm optimization for multi-objective job shop scheduling. *Computers and Industrial Engineering, 54*(4), 960–971.

Liang, Y. C., Ge, H. W., Zho, Y., Guo, X. C., et al. (2005). A particle swarm optimization-based algorithm for job-shop scheduling problems. *International Journal of Computational Methods, 2*(3), 419–430.

Lourenço, H. R. (1995). Local optimization and the job-shop scheduling problem. *European Journal of Operational Research, 83*, 347–364.

Nowicki, E., & Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science, 42*(6), 797–813.

Pezzella, F., & Merelli, E. (2000). A tabu search method guided by shifting bottleneck for the job shop scheduling problem. *European Journal of Operational Research, 120*(2), 297–310.

Ponnambalam, S. G., Ramkumar, V., & Jawahar, N. (2001). A multi-objective genetic algorithm for job shop scheduling. *Production Planning and Control, 12*(8), 764–774.

Ripon, K. S. N. (2007). Hybrid evolutionary approach for multi-objective job-shop scheduling problem. *Malaysian Journal of Computer Science, 20*(2), 183–198.

Sha, D. Y., & Hsu, C.-Y. (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers and Industrial Engineering, 51*(4), 791–808.

Sha, D. Y., & Hsu, C.-Y. (2008). A new particle swarm optimization for the open shop scheduling problem. *Computers and Operations Research, 35*, 3243–3261.

Sun, D., Batta, R., & Lin, L. (1995). Effective job shop scheduling through active chain manipulation. *Computers and Operations Research, 22*(2), 159–172.

Suresh, R. K., & Mohanasndaram, K. M. (2006). Pareto archived simulated annealing for job shop scheduling with multiple objectives. *International Journal of Advanced Manufacturing Technology, 29*, 184–196.

Taillard, E. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*, 278–285.

Wang, L., & Zheng, D.-Z. (2001). An effective hybrid optimization strategy for job-shop scheduling problems. *Computers and Operations Research, 28*, 585–596.

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers and Industrial Engineering, 48*, 409–425.

Zhang, H., Li, X., Li, H., & Hang, F. (2005). Particle swarm optimization-based schemes for resource-constrained project scheduling. *Automation in Construction, 14*(3), 393–404.