Available online at www.sciencedirect.com
ScienceDirect
Computers & Industrial Engineering 51 (2006) 791–808
computers & industrial engineering
www.elsevier.com/locate/dsw

A hybrid particle swarm optimization for job shop scheduling problem

D.Y. Sha [a,b,*], Cheng-Yu Hsu [b]

[a] Department of Business Administration, Asia University, 500 Liufeng Road, Wufong, Taichung 413, Taiwan, ROC
[b] Department of Industrial Engineering and Management, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan, ROC

Received 14 January 2006; received in revised form 14 September 2006; accepted 14 September 2006
Available online 27 October 2006

Abstract

A hybrid particle swarm optimization (PSO) for the job shop problem (JSP) is proposed in this paper. In previous research, PSO particles search solutions in a continuous solution space. Since the solution space of the JSP is discrete, we modified the particle position representation, particle movement, and particle velocity to better suit PSO for the JSP. We modified the particle position based on preference list-based representation, particle movement based on swap operator, and particle velocity based on the tabu list concept in our algorithm. Giffler and Thompson's heuristic is used to decode a particle position into a schedule. Furthermore, we applied tabu search to improve the solution quality. The computational results show that the modified PSO performs better than the original design, and that the hybrid PSO is better than other traditional metaheuristics.

Keywords: Job shop problem; Scheduling; Particle swarm optimization

1. Introduction

The job shop scheduling problem (JSP) is one of the most difficult combinatorial optimization problems. The JSP can be briefly stated as follows (French, 1982; Gen & Cheng, 1997). There are n jobs to be processed through m machines. We shall suppose that each job must pass through each machine once and once only. Each job should be processed through the machines in a particular order, and there are no precedence constraints among different job operations. Each machine can process only one job at a time, and it cannot be interrupted. Furthermore, the processing time is fixed and known. The problem is to find a schedule to minimize the makespan (C_{max}), that is, the time required to complete all jobs.

Garey, Johnson, and Sethi (1976) demonstrated that JSP is NP-hard, so it cannot be exactly solved in a reasonable computation time. Many approximate methods have been developed in the last decade to solve

* Corresponding author. Tel.: +886 4 23323456x1936; fax: +886 4 2331 6699.
E-mail addresses: yjsha@mail.nctu.edu.tw (D.Y. Sha), cyhsu.iem92g@nctu.edu.tw (C.-Y. Hsu).

0360-8352/$ - see front matter
doi:10.1016/j.cie.2006.09.002

JSP, such as *simulated annealing* (SA) (Lourenço, 1995), *tabu search* (TS) (Nowicki & Smutnicki, 1996; Pezzella & Merelli, 2000; Sun, Batta, & Lin, 1995), and *genetic algorithm* (GA) (Bean, 1994; Gonçalves, Mendes, & Resende, 2005; Kobayashi, Ono, & Yamamura, 1995; Wang & Zheng, 2001). We applied a new evolutionary search technique – particle swarm optimization (PSO) – to solve the JSP in this paper.

The optimal JSP solution should be an active schedule. In an active schedule the processing sequence is such that no operation can be started any earlier without delaying some other operation (French, 1982). To reduce the search solution space, the tabu search proposed by Sun et al. (1995) searches solutions within the set of active schedules. In our algorithm, we applied Giffler and Thompson's (1960) heuristic to decode a particle position into a schedule. Furthermore, we applied a tabu search to improve the solution quality.

The background of particle swarm optimization (PSO) is introduced in the next section. In Section 3, we propose a hybrid PSO for the JSP. In Section 4, we test the hybrid PSO on Fisher and Thompson (1963) and Lawrence (1984) and Taillard (1993) test problems. Finally, conclusions and remarks for further works are given in Section 5.

2. The background of particle swarm optimization

Particle swarm optimization (PSO) was developed by Kennedy and Eberhart (1995). PSO is a population-based optimization algorithm. Each particle is an individual and the swarm is composed of particles. The problem solution space is formulated as a search space. Each position in the search space is a correlated solution of the problem. Particles cooperate to find out the best position (best solution) in the search space (solution space).

Particles move toward the pbest position and gbest position with each iteration. The pbest position is the best position found by each particle so far. Each particle has its own pbest position. The gbest position is the best position found by the swarm so far. The particle moves itself according to its velocity. The velocities are randomly generated toward pbest and gbest positions. For each particle k and dimension j, the velocity and position of particles can be updated by the following equations:

$$v_{kj} \leftarrow w \times v_{kj} + c_1 \times rand_1 \times (pbest_{kj} - x_{kj}) + c_2 \times rand_2 \times (gbest_j - x_{kj}) \tag{1}$$

$$x_{kj} \leftarrow x_{kj} + v_{kj} \tag{2}$$

In Eqs. (1) and (2), v_{kj} is the velocity of particle k on dimension j, and x_{kj} is the position of particle k on dimension j. The $pbest_{kj}$ is the pbest position of particle k on dimension j, and $gbest_j$ is the gbest position of the swarm on dimension j. The inertia weight w was first proposed by Shi and Eberhart (1998a, 1998b), and is used to control exploration and exploitation. The particles maintain high velocities with a larger w, and low velocities with a smaller w. A larger w can prevent particles from becoming trapped in local optima, and a smaller w encourages particles exploiting the same search space area. The constants c_1 and c_2 are used to decide whether particles prefer moving toward a pbest position or gbest position. The $rand_1$ and $rand_2$ are random variables between 0 and 1. The process for PSO is as follows:

Step 1: Initialize a population of particles with random positions and velocities on d dimensions in the search space.
Step 2: Update the velocity of each particle, according to Eq. (1).
Step 3: Update the position of each particle, according to Eq. (2).
Step 4: Map the position of each particle into solution space and evaluate its fitness value according to the desired optimization fitness function. At the same time, update pbest and gbest position if necessary.
Step 5: Loop to step 2 until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations.

The original PSO design is suited to a continuous solution space. For better suiting to combinatorial optimization problems, we have to modify PSO position representation, particle velocity, and particle movement. Zhang, Li, Li, and Huang (2005) proposed a PSO for resource-constrained project scheduling, and compared two kinds of position representation: (1) priority-based representation (particle position represented by priority values), and (2) permutation-based representation (particle position represented by a sequential order of activities). Zhang's results (2005) showed that permutation-based representation is better than priority-based representation.

We modified the particle position based on preference list-based representation (Davis, 1985) and the particle movement based on a swap operator in this paper. These will be discussed in Section 3.

3. A hybrid particle swarm optimization

In this section, we will first describe how to associate a particle position into a schedule with two different position representations, respectively, the priority-based representation and preference list-based representation. If we implement the priority-based representation, the particle position consists of continuous variables, and it is suited to the original PSO design, as described in Section 2. When we implement the preference list-based representation, we have to modify the particle velocity and particle movement, as described in Sections 3.2 and 3.3. Besides, we propose a diversification strategy and a local search procedure for better performance.

3.1. Position representation

3.1.1. Priority-based representation

When we implement the original PSO design, as described in Section 2 (i.e., the particles search solutions in a continuous solution space), each value of a particle position represents the associated operation priority. For an n-job m-machine problem, we can represent the particle k position by an $m \times n$ matrix, i.e.

$$X^k = \begin{bmatrix} x_{11}^k & x_{12}^k & \cdots & x_{1n}^k \\ x_{21}^k & x_{22}^k & \cdots & x_{2n}^k \\ & \vdots & & \\ x_{m1}^k & x_{m2}^k & \cdots & x_{mn}^k \end{bmatrix}$$

where x_{ij}^k denotes the priority of operation o_{ij} and o_{ij} is the operation of job j that needs to be processed on machine i. We can map (or decode) a particle position into an active schedule using Giffler and Thompson's (1960) heuristic. We briefly describe the G&T algorithm as follows:

Notation:

(i,j): the operation of job j that needs to be processed on machine i.
S: the partial schedule that contains scheduled operations.
Ω: the set of schedulable operations.
$s_{(i,j)}$: the earliest time at which operation $(i,j) \in \Omega$ could be started.
$p_{(i,j)}$: the processing time of operation (i,j).
$f_{(i,j)}$: the earliest time at which operation $(i,j) \in \Omega$ could be finished, $f_{(i,j)} = s_{(i,j)} + p_{(i,j)}$.

G&T algorithm:

Step 1: Initialize $S = \phi$; Ω is initialized 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^* could 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 (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. Else, delete (i,j) from Ω and include its immediate successor in Ω, then go to Step 2.

For example, there are two jobs and two machines, as shown on Table 1, and the position of particle k is

$$X^k = \begin{bmatrix} 0.6 & 1.3 \\ 0.8 & 0.5 \end{bmatrix}.$$

We can use the G&T algorithm to decode X^k into a schedule in the following steps:

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 $(2,2)$, which has the largest priority, and add it into schedule S, as illustrated in Fig. 1(a).
Step 4: Update $\Omega = \{(1, 1), (1, 2)\}$.

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 $(1, 2)$, which has the largest priority, and add it into schedule S, as illustrated in Fig. 1(b).
Step 4: Update $\Omega = \{(1, 1)\}$.

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 $(1, 1)$, which has the largest priority, and add it into schedule S, as illustrated in Fig. 1(c).
Step 4: Update $\Omega = \{(2, 1)\}$

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 $(2, 1)$, which has the largest priority, and add it into schedule S, as illustrated in Fig. 1(d).
Step 4: A complete schedule has been generated, and then stops.

However, there is a shortcoming of priority-based representation. The schedules of two particles may be quite different even though their positions are very close to each other. For example, if there are six operations to be sorted on a machine, and there are two positions of two particles as follows:

$$\text{position 1}: \quad [0.25, 0.27, 0.21, 0.24, 0.26, 0.23]$$

Table 1
A 2 × 2 example

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

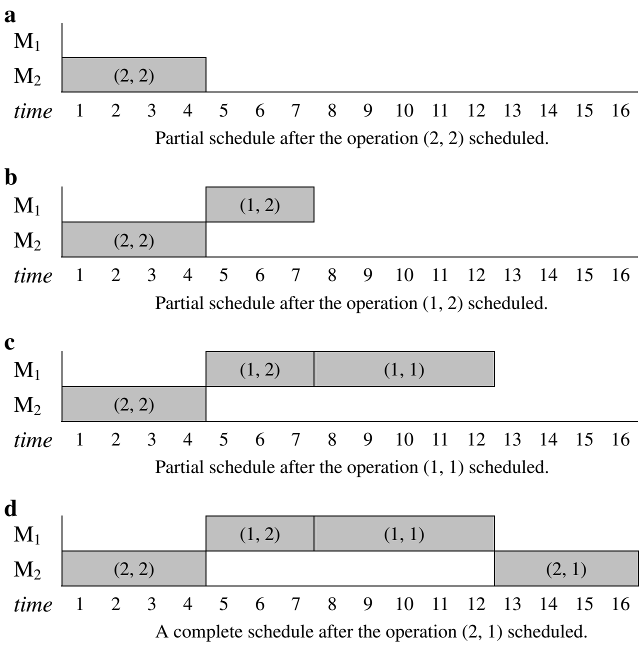

Fig. 1. An illustration of decoding a particle position into a schedule.

$$\text{position 2}: \quad [0.22, 0.25, 0.23, 0.26, 0.24, 0.21]$$

then we sort the operations according to the decreasing order of their position values as follows:

$$\text{permutation 1}: \quad [2 \quad 5 \quad 1 \quad 4 \quad 6 \quad 3]$$

$$\text{permutation 2}: \quad [4 \quad 2 \quad 5 \quad 3 \quad 1 \quad 6]$$

We can find that these two permutations are quite different even though the particle positions are very close to each other. This is because the location in the permutation of one operation depends on the position values of other operations.

3.1.2. Preference list-based representation

In the preference list-based representation, there is a preference list for each machine. For an n-job m-machine problem, we can also represent the particle k position by an $m \times n$ matrix, and the ith row is the preference list of machine i, i.e.

$$X^k = \begin{bmatrix} x_{11}^k & x_{12}^k & \cdots & x_{1n}^k \\ x_{21}^k & x_{22}^k & \cdots & x_{2n}^k \\ & \vdots & & \\ x_{m1}^k & x_{m2}^k & \cdots & x_{mn}^k \end{bmatrix}$$

where $x_{ij}^k \in \{1, 2, \ldots, n\}$ denotes the job on location j in the preference list of machine i. We can also use Giffler and Thompson's (1960) heuristic to map a particle position into an active schedule. The same example, as shown in Table 1, and 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), which is ahead of others in the preference list of machine 2, and add it into schedule S, as illustrated in Fig. 1(a).
Step 4: Update $\Omega = \{(1, 1), (1, 2)\}$.

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), which is ahead of others in the preference list of machine 1, and add it into schedule S, as illustrated in Fig. 1(b).
Step 4: Update $\Omega = \{(1, 1)\}$.

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), which is ahead of others in the preference list of machine 1, and add it into schedule S, as illustrated in Fig. 1(c).
Step 4: Update $\Omega = \{(2, 1)\}$

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), which is ahead of others in the preference list of machine 2, and add it into schedule S, as illustrated in Fig. 1(d).
Step 4: A complete schedule has been generated, and then stops.

The preference list-based PSO we proposed differs from the original PSO design in that the pbest solutions and gbest solution do not record the best positions found so far, but rather the best schedules generated by the G&T algorithm. For the above example, we do not record the particle position X^k, but record the schedule

$$S^k = \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}$$

into pbest and gbest solutions if necessary. Because the particle position representation differs from the original design, we also modified the movement based on the swap operator. This will be discussed in Section 3.3.

3.2. Modified particle velocity

When a particle moves in a continuous solution space, due to inertia, the particle velocity not only moves the particle to a better position, but also prevents the particle from moving back to the current position. The velocity can be controlled by inertia weight w in Eq. (1). The larger the inertia weight, the harder the particle backs to the current position.

If we implement preference list-based representation, the velocity of operation o_{ij} of particle k is denoted by v^k_{ij}, $v^k_{ij} \in \{0, 1\}$, where o_{ij} is the operation of job j that needs to be processed on machine i. When v^k_{ij} equals 1, it means that operation o_{ij} in the preference list of particle k (the position matrix, X^k) has just been moved to the current location, and we should not move it in this iteration. On the contrary, if operation o_{ij} is moved to a new location in this iteration, we set $v^k_{ij} \leftarrow 1$, indicating that o_{ij} has been moved in this iteration and should not been moved in the next few iterations. The particle velocity can prevent recently moved operations from moving back to the original location in the next iterations.

Just as the original PSO is applied to a continuous solution space, inertia weight w is used to control particle velocities. We randomly update velocities at the beginning of the iteration. For each particle k and operation o_{ij}, if v_{ij}^k equals 1, v_{ij}^k will be set to 0 with probability $(1 - w)$. This means that if operation o_{ij} is fixed on the current location in the preference list of particle k, o_{ij} is allowed to move in this iteration with probability $(1 - w)$. The newly moved operations will then be fixed for more iterations with larger inertia weight, and fixed for less iterations with smaller inertia weight. The pseudo code for updating velocities is given in Fig. 2.

3.3. Modified particle movement

The modified particle movement is based on the swap operator. If $v_{ij}^k = 0$, the job j on x_i^k will be moved to the corresponding location of $pbest_i^k$ with probability c_1, and will be moved to the corresponding location of $gbest_i$ with probability c_2. Where x_i^k is the preference list of machine i of particle k, $pbest_i^k$ is the preference list of machine i of kth *pbest* solution, $gbest_i$ is the preference list of machine i of *gbest* solution, c_1 and c_2 are constant between 0 and 1, and $c_1 + c_2 \leqslant 1$. The process is described as follows:

Step 1: Randomly choose a location l in x_i^k.
Step 2: Denote the job on location l in x_i^k by J_1.
Step 3: Find out the location of J_1 in $pbest_i^k$ with probability c_1, or find out the location of J_1 in $gbest_i$ with probability c_2. Denote the location that has been found in $pbest_i^k$ or $gbest_i$ by l', and denote the job in location l' in x_i^k by J_2.
Step 4: If J_2 has been denoted, $v_{iJ_1}^k = 0$, and $v_{iJ_2}^k = 0$, then swap J_1 and J_2 in x_i^k, and set $v_{iJ_1}^k \leftarrow 1$.
Step 5: If all the locations in x_i^k have been considered, then stop. Otherwise, if $l < n$, then set $l \leftarrow l + 1$, else $l \leftarrow 1$, and go to Step 2, where n is the number of jobs.

For example, there is a five-job problem, and x_i^k, $pbest_i^k$, $gbest_i$, and v_i^k are shown in Fig. 3(a). We set $c_1 = 0.5$ and $c_2 = 0.3$ in this instance.

In Step 1, we randomly choose a location $l = 3$. In Step 2, the job in the 3rd location in x_i^k is job 4, i.e. $J_1 = 4$. In Step 3, we generate a random variable *rand* between 0 and 1, and the generated random variable *rand* is 0.6. Since $c_1 < rand \leqslant c_1 + c_2$, we find out the location of J_1 in $gbest_i$. The location $l' = 5$, and the job in the 5th location in x_i^k is job 5, i.e., $J_2 = 5$. Steps 1–3 are shown in Fig. 3(b). In Step 4, since $v_{i4}^k = 0$ and $v_{i5}^k = 0$, swap jobs 4 and 5 in x_i^k and set $v_{i4}^k \leftarrow 1$ are shown in Fig. 3(c). In Step 5, set $l \leftarrow 4$, and go to Step 2. Repeat the procedure until all the locations in x_i^k have been considered.

We also adopt a mutation operator in our algorithm. After a particle moves to a new position, we randomly choose a machine and two jobs on the machine, and then swap these two jobs, disregarding v_{ij}^k. The particle movement pseudo code is given in Fig. 4.

3.4. The diversification strategy

If all the particles have the same pbest solutions, they will be trapped into local optima. To prevent such a situation, we proposed a diversification strategy to keep the pbest solutions different (i.e., keeps the makespans

```
for each particle k and operation o_ij do
  rand ~ U(0,1)
  if (v_ij^k = 1) and (rand ≥ w) then
      v_ij^k ← 0
  end if
end for
```

Fig. 2. Pseudo code of updating velocities.

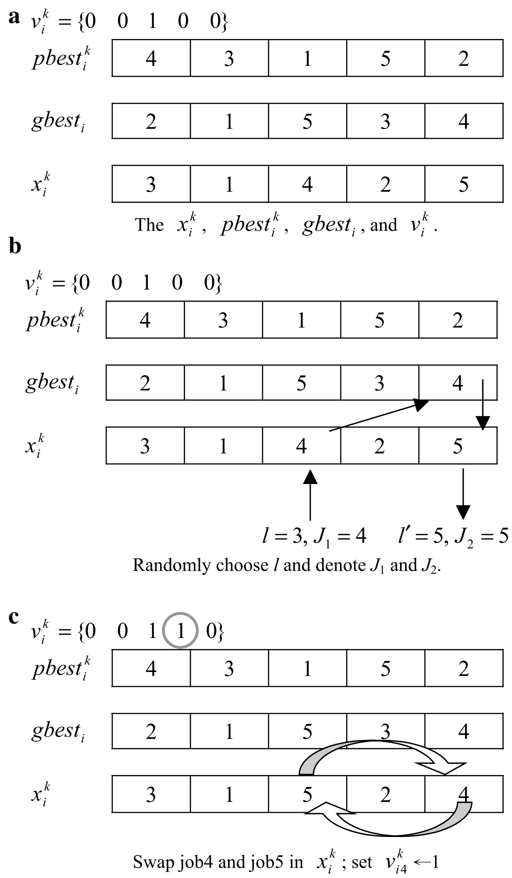

Fig. 3. An instance of particle movement.

of pbest solutions different). In the diversification strategy, the pbest solution of each particle is not the best solution found by the particle itself, but one of the best N solutions found by the swarm so far where N is the size of the swarm. Once any particle generates a new solution, the pbest and gbest solutions will be updated in these three situations:

1. If the particle's fitness value is better than the fitness value of the gbest solution, set the worst pbest solution equal to the current gbest solution, and set the gbest solution equal to the particle solution.
2. If the particle's fitness value is worse than the gbest solution, but better then the worst pbest solution and not equal to any gbest or pbest solution, set the worst pbest solution equal to the particle solution.
3. If the particle's fitness value is equal to any pbest or gbest solution, replace the pbest or gbest solution (whose fitness value is equal to the particle fitness value) with the particle solution.

The pseudo code for updating the pbest solution and gbest solution with diversification strategy is given in Fig. 5.

3.5. Local search

The tabu search is a metaheuristic approach and a strong local search mechanism. In the tabu search, the algorithm starts from an initial solution and improves it iteratively to find a near-optimal solution. This method was proposed and formalized primarily by Glover (1986, 1989, 1990). We applied the tabu search proposed by Nowicki and Smutnicki (1996) but without back jump tracking. We briefly describe Nowicki and Smutnicki's method as follows:

```
for i←1 to m do //for machine 1 to machine m
    l_start ← an integer random number between 1 to n
    l ← l_start
    for j←1 to n do //for all location
        rand ~ U(0,1)
        if (rand ≤ c1) then
            J1 ← x^k_il
            l′ ← the location of J1 in pbest^k_i
            J2 ← x^k_il′
            if (v^k_iJ1 = 0) and (v^k_iJ2 = 0) and (J1 ≠ J2) then
                x^k_il ← J2;  x^k_il′ ← J1;  v^k_iJ1 ← 1
            end if
        end if
        if (c1 < rand ≤ c1 + c2) then
            J1 ← x^k_il
            l′ ← the location of J1 in gbest_i
            J2 ← x^k_il′
            if (v^k_iJ1 = 0) and (v^k_iJ2 = 0) and (J1 ≠ J2) then
                x^k_il ← J2;  x^k_il′ ← J1;  v^k_iJ1 ← 1
            end if
        end if
        l ← l_start + j
        if (l > n) then
            l ← l − n
        end if
    end for
end for
//mutation operator
M ← randomly choose a machine between 1 to m
l ← randomly choose a location between 1 to n
l′ ← randomly choose a location between 1 to n
J1 ← x^k_Ml;  J2 ← x^k_Ml′
x^k_Ml ← J2;  x^k_Ml′ ← J1
v^k_MJ1 ← 1;  v^k_MJ2 ← 1
//mutation operator
```

Fig. 4. Pseudo code of particle movement.

3.5.1. The neighborhood structure

Nowicki and Smutnicki's method randomly chooses a critical path in the current schedule, and then represents the critical path in terms of blocks. The neighborhood exchanges the first two and the last two operations in every block, but excludes the first and last operations in the critical path. The research of Jain, Rangaswamy, and Meeran (2000) shows that the strategy used to generate the critical path does not materially affect the final solution. Therefore, in this paper, we randomly choose one critical path if there is more than one critical path. For example, there is a schedule for a four-job, three-machine problem, as shown in Fig. 6(a). We can find that there are two critical paths: $CP_1 = \{o_{31}, o_{11}, o_{13}, o_{33}\}$ and $CP_2 = \{o_{31}, o_{32}, o_{22}, o_{21}, o_{24}, o_{14}\}$, where o_{ij} is the operation of job j that needs to be processed on machine i. If we randomly choose CP_2, we can represent CP_2 in terms of blocks: $\{o_{31}, o_{32}\}$, $\{o_{22}, o_{21}, o_{24}\}$, and $\{o_{14}\}$. The possible moves in this schedule are exchanging $\{o_{22}, o_{21}\}$ or $\{o_{21}, o_{24}\}$ (see Fig. 6(b)).

N: the size of the swarm
S^k : the schedule generated by particle k
$pbest^{worst}$: the worst solution of pbest solutions
$C_{\max}(S^k)$: the makespan of S^k
// ↓ situation 1 as described in 3.4
if $(C_{\max}(S^k) < C_{\max}(gbest))$ ***then***
    $pbest^{worst} \leftarrow gbest$; $gbest \leftarrow S^k$
// ↑ situation 1 as described in 3.4
else if $(C_{\max}(S^k) \le C_{\max}(pbest^{worst}))$ ***then***
    // ↓ situation 3 as described in 3.4
    $the\_same = 0$
    if $(C_{\max}(S^k) = C_{\max}(gbest))$ ***then***
        $gbest \leftarrow S^k$; $the\_same = 1$
    else
        for $k' \leftarrow 1$ ***to*** N ***do***
            if $(C_{\max}(S^k) = C_{\max}(pbest^{k'}))$ ***then***
                $pbest^{k'} \leftarrow S^k$; $the\_same = 1$
                break
            end if
        end for
    end if
    // ↑ situation 3 as described in 3.4
    // ↓ situation 2 as described in 3.4
    if $(the\_same = 0)$ ***then***
        $pbest^{worst} \leftarrow S^k$
    end if
    // ↑ situation 2 as described in 3.4
end if

Fig. 5. Pseudo code of updating pbest solution and gbest solution with diversification strategy.

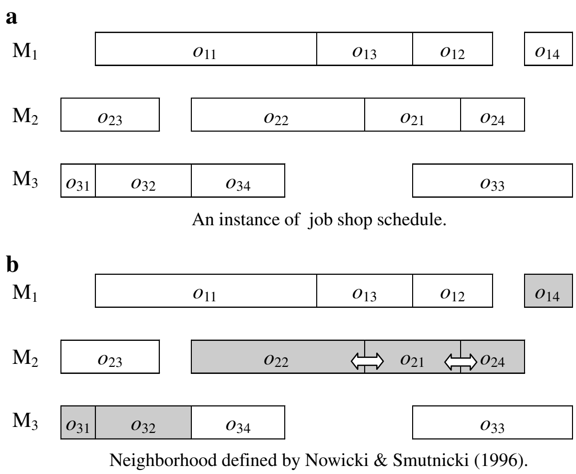

Fig. 6. An illustration of neighborhoods in tabu search.

3.5.2. Tabu list

The tabu list consists of *maxt* operation pairs that have been moved in the last *maxt* moves in the tabu search. If a move $\{o_{iJ_1}, o_{iJ_2}\}$ has been performed, this move replaces the oldest move in the tabu list, and moving these same two operations is not permitted while the move is recorded in the tabu list.

Table 2
Computational result of FT and LA test problems

Problem	Size ($n \times m$)	Best Known Solution (BKS)	Shifting bottleneck				Tabu Search		
			SBI	SBII	SB-RGLS1	SB-RGLS2	ACM	TSAB	TSSB
			Adams et al. (1988)		Balas and Vazacopoulos (1998)		Sun et al. (1995)	Nowicki and Smutnicki (1996)	Pezzella and Merelli (2000)
FT06	6×6	55	55	55	–	–	–	55	55
FT10	10×10	930	1015	930	930	930	930	930	930
FT20	20×5	1165	1290	1178	–	–	–	1165	1165
LA01	10×5	666	666	666	–	–	–	666	666
LA02	10×5	655	720	669	655	655	–	655	655
LA03	10×5	597	623	605	–	–	–	597	597
LA04	10×5	590	597	593	–	–	–	590	590
LA05	10×5	593	593	593	–	–	–	593	593
LA06	15×5	926	926	926	–	–	–	926	926
LA07	15×5	890	890	890	–	–	–	890	890
LA08	15×5	863	868	863	–	–	–	863	863
LA09	15×5	951	951	951	–	–	–	951	951
LA10	15×5	958	959	959	–	–	–	958	958
LA11	20×5	1222	1222	1222	–	–	–	1222	1222
LA12	20×5	1039	1039	1039	–	–	–	1039	1039
LA13	20×5	1150	1150	1150	–	–	–	1150	1150
LA14	20×5	1292	1292	1292	–	–	–	1292	1292
LA15	20×5	1207	1207	1207	–	–	–	1207	1207
LA16	10×10	945	1021	978	–	–	975	945	945
LA17	10×10	784	796	787	–	–	784	784	784
LA18	10×10	848	891	859	–	–	848	848	848
LA19	10×10	842	875	860	842	842	842	842	842
LA20	10×10	902	924	914	–	–	902	902	902
LA21	15×10	1046	1172	1084	1048	1046	1074	1047	1046
LA22	15×10	927	1040	944	–	–	941	927	927
LA23	15×10	1032	1061	1032	–	–	1032	1032	1032
LA24	15×10	935	1000	976	937	935	954	939	938
LA25	15×10	977	1048	1017	977	977	1010	977	979
LA26	20×10	1218	1304	1224	–	–	1218	1218	1218
LA27	20×10	1235	1325	1291	1235	1235	1277	1236	1235
LA28	20×10	1216	1256	1250	–	–	1245	1216	1216
LA29	20×10	1157	1294	1239	1164	1164	1234	1160	1168
LA30	20×10	1355	1403	1355	–	–	1355	1355	1355
LA31	30×10	1784	1784	1784	–	–	1784	1784	1784
LA32	30×10	1850	1850	1850	–	–	1850	1850	1850
LA33	30×10	1719	1719	1719	–	–	1719	1719	1719
LA34	30×10	1721	1721	1721	–	–	1721	1721	1721
LA35	30×10	1888	1888	1888	–	–	1888	1888	1888
LA36	15×15	1268	1351	1305	1268	1268	1303	1268	1268
LA37	15×15	1397	1485	1423	1397	1397	1422	1407	1411
LA38	15×15	1196	1280	1255	1198	1196	1245	1196	1201
LA39	15×15	1233	1321	1273	1233	1233	1269	1233	1240
LA40	15×15	1222	1326	1269	1226	1224	1255	1229	1233
Average gap			3.8796%	1.3838%	0.1157%	0.0591%	1.5184%	0.0501%	0.1015%
No. of instance			43	43	13	13	26	43	43
No. of BKS obtained			16	20	8	11	13	37	36

(*continued on next page*)

Table 2 (*continued*)

Genetic algorithm		Particle Swarm Optimization					
GASA	HGA-Param	PSO-priority based		PSO-permutation based		HPSO	
Wang and Zheng (2001)	Gonçalves et al. (2005)	Best solution	Average	Best solution	Average	Best solution	Average
55	55	55	58.9	55	55.0	55	55.0
930	930	1007	1086.0	937	965.2	930	932.0
1165	1165	1242	1296.7	1165	1178.8	1165	1165.0
666	666	681	705.0	666	666.0	666	666.0
–	655	694	729.7	655	662.1	655	655.0
–	597	633	657.5	597	602.3	597	597.0
–	590	611	648.1	590	592.9	590	590.0
–	593	593	601.1	593	593.0	593	593.0
926	926	926	940.2	926	926.0	926	926.0
–	890	890	941.0	890	890.0	890	890.0
–	863	863	896.6	863	863.0	863	863.0
–	951	953	991.8	951	951.0	951	951.0
–	958	958	976.1	958	958.0	958	958.0
1222	1222	1222	1235.3	1222	1222.0	1222	1222.0
–	1039	1039	1058.4	1039	1039.0	1039	1039.0
–	1150	1150	1179.0	1150	1150.0	1150	1150.0
–	1292	1292	1292.2	1292	1292.0	1292	1292.0
–	1207	1232	1271.7	1207	1207.0	1207	1207.0
945	945	1006	1033.5	945	969.8	945	945.2
–	784	833	883.5	784	787.1	784	784.0
–	848	901	959.9	848	856.8	848	848.0
–	842	895	945.8	842	851.5	842	842.0
–	907	963	1014.0	907	913.3	902	902.3
1058	1046	1201	1247.5	1055	1085.5	1046	1049.8
–	935	1046	1142.5	935	950.5	927	927.0
–	1032	1146	1205.1	1032	1032.0	1032	1032.0
–	953	1082	1140.9	937	967.8	935	937.9
–	986	1107	1176.6	983	1005.9	977	978.2
1218	1218	1409	1468.0	1218	1219.7	1218	1218.0
–	1256	1437	1495.4	1252	1269.1	1235	1251.4
–	1232	1434	1487.4	1216	1241.7	1216	1216.0
–	1196	1359	1429.8	1179	1215.8	1163	1168.8
–	1355	1517	1557.0	1355	1355.0	1355	1355.0
1784	1784	1886	1942.5	1784	1784.0	1784	1784.0
–	1850	2000	2065.6	1850	1850.0	1850	1850.0
–	1719	1832	1896.8	1719	1719.0	1719	1719.0
–	1721	1876	1953.5	1721	1721.0	1721	1721.0
–	1888	2027	2074.5	1888	1888.0	1888	1888.0
1292	1279	1437	1541.0	1291	1317.5	1268	1271.3
–	1408	1539	1628.0	1442	1475.1	1397	1401.6
–	1219	1370	1445.1	1228	1251.1	1196	1200.5
–	1246	1436	1499.4	1233	1285.6	1233	1233.0
–	1241	1380	1457.4	1236	1258.0	1224	1226.2
0.2764%	0.3916%	7.4021%	12.0940%	0.3719%	1.3491%	0.0159%	0.1091%
11	43	43		43		43	
9	31	10		31		41	

3.5.3. *Back jump tracking*

When finding a new best solution, store the current state (the new best solution, set of moves, and tabu list) in a list L. After the tabu search algorithm performs *maxiter_tabu* iterations, restart the tabu search algorithm from the latest recorded state, and repeat it until the list L is empty. We did not implement the back jump tracking in our algorithm to reduce computation time.

We implement a tabu search procedure after a particle generates a new solution for further improved solution quality. The tabu search will be stopped after 100 moves that do not improve the solution. The research of Jain et al. (2000) shows that the solution quality of tabu search (Nowicki & Smutnicki, 1996) is mainly affected by its initial solution. Therefore, in the hybrid PSO, the purpose of the PSO process is to provide good and diverse initial solutions to the tabu search.

4. Computational results

There are three PSOs we tested: (1) priority-based PSO, of which the particle position is represented by the priorities of operations, and implements the original PSO design as described in Section 2; (2) preference list-based PSO, of which the particle position is represented by a preference list of machines; (3) hybrid PSO (HPSO), which is the preference list-based PSO with a local search mechanism. The PSOs were tested on Fisher and Thompson (1963) (FT06, FT10, and FT20), Lawrence (1984) (LA01 to LA40) and Taillard (1993) (TA01 to TA80) test problems. These problems are available on the OR-Library web site (Beasley, 1990) (URL: http://people.brunel.ac.uk/~mastjjb/jeb/info.html) and Taillard's web site (URL: http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html).

In the preliminary experiment, four swarm sizes N (10, 20, 30, 50) were tested, where $N = 30$ was superior and used for all further studies. The other parameters of the priority-based PSO were set to the same common settings as most of the previous research: $c_1 = 2.0$, $c_2 = 2.0$, the inertia weight w is decreased linearly from 0.9 to 0.4 during a run, and the maximum value of $|x_{ij}|$ and $|v_{ij}|$, $X_{\max}$ and $V_{\max}$ are equal to the number of jobs n and $n/5$, respectively.

The parameters of the preference list-based PSO are determined experimentally. The parameters c_1 and c_2 were tested between 0.1 and 0.5 in increments of 0.1, and the parameter w was tested between 0 and 0.9 in increments of 0.1. The settings $c_1 = 0.5$, $c_2 = 0.3$ and $w = 0.5$ were superior. The length of the tabu list *maxt* was set to 8 where the value is derived from Nowicki and Smutnicki (1996). The tabu search will be stopped after 100 moves that do not improve the solution. The priority-based PSO and the preference list-based PSO will be terminated after 10^5 iterations, and HPSO will be terminated after 10^3 iterations. The number of iterations is determined by the computation time compared with Pezzella and Merelli (2000) and Gonçalves et al. (2005).

The program was coded in Visual C++, optimized by speed, and run on an AMD Athlon 1700+ PC 20 times for each of the 123 problems. The proposed algorithm is compared with Shifting Bottleneck (Adams,

Table 3
Computation time of FT and LA test problems (in CPU seconds)

Problem	Size ($n \times m$)	HGA-Param Gonçalves et al. (2005)[a]	Particle swarm optimization[b]					
			PSO-priority based		PSO-permutation based		HPSO	
			Best solution time	Total time	Best solution time	Total time	Best solution time	Total time
FT06	6 × 6	13	0.0	34	0.0	32	0.0	28
FT10	10 × 10	292	1.0	112	21.7	91	4.1	157
FT20	20 × 5	204	3.0	180	19.2	138	19.8	219
LA01-05	10 × 5	40	0.4	60	5.3	50	0.5	38
LA06-10	15 × 5	94	1.0	114	0.1	92	0.1	61
LA11-15	20 × 5	192	3.5	177	0.5	143	0.1	100
LA16-20	10 × 10	227	0.6	109	15.5	90	19.9	139
LA21-25	15 × 10	602	4.8	208	37.2	164	59.6	295
LA26-30	20 × 10	1303	12.6	325	103.1	259	90.5	579
LA31-35	30 × 10	3691	46.9	652	31.4	520	3.0	1462
LA36-40	15 × 15	1920	7.4	331	68.4	254	105.2	471

[a] Run on an AMD Thunderbird 1.333 GHz PC.
[b] Run on an AMD Athlon 1700+ PC.

Table 4
Computational result of TA test problems

Problem	Size ($n \times m$)	Optimal solution (or upper bound)	TSAB	TSSB	HPSO	
			Nowicki and Smutnicki (1996)	Pezzella and Merelli (2000)	Best solution	Average
TA01	15 × 15	1231		1241	1231	1236
TA02	15 × 15	1244	1244	1244	1244	1245
TA03	15 × 15	1218	1222	1222	1218	1224
TA04	15 × 15	1175		1175	1175	1180
TA05	15 × 15	1224	1233	1229	1224	1233
TA06	15 × 15	1238		1245	1238	1248
TA07	15 × 15	1227		1228	1228	1229
TA08	15 × 15	1217	1220	1220	1217	1220
TA09	15 × 15	1274	1282	1291	1274	1283
TA10	15 × 15	1241	1259	1250	1249	1264
TA11	20 × 15	(1359)		1371	1366	1386
TA12	20 × 15	(1367)	1377	1379	1370	1380
TA13	20 × 15	(1342)		1362	1350	1364
TA14	20 × 15	1345	1345	1345	1345	1350
TA15	20 × 15	(1339)		1360	1350	1364
TA16	20 × 15	(1360)		1370	1368	1377
TA17	20 × 15	1462		1481	1473	1480
TA18	20 × 15	(1396)	1413	1426	1407	1425
TA19	20 × 15	(1335)	1352	1351	1335	1353
TA20	20 × 15	(1348)	1362	1366	1358	1373
TA21	20 × 20	(1644)		1659	1658	1679
TA22	20 × 20	(1600)		1623	1614	1625
TA23	20 × 20	(1557)		1573	1559	1578
TA24	20 × 20	(1646)		1659	1654	1664
TA25	20 × 20	(1595)		1606	1616	1632
TA26	20 × 20	(1645)	1657	1666	1662	1679
TA27	20 × 20	(1680)		1697	1690	1712
TA28	20 × 20	(1603)		1622	1617	1627
TA29	20 × 20	(1625)	1629	1635	1634	1645
TA30	20 × 20	(1584)		1614	1589	1613
TA31	30 × 15	1764	1766	1771	1766	1772
TA32	30 × 15	(1795)	1841	1840	1823	1848
TA33	30 × 15	(1791)	1832	1833	1818	1834
TA34	30 × 15	(1829)		1846	1844	1879
TA35	30 × 15	2007		2007	2007	2010
TA36	30 × 15	1819		1825	1825	1843
TA37	30 × 15	1771	1815	1813	1795	1808
TA38	30 × 15	1673	1700	1697	1681	1701
TA39	30 × 15	1795	1811	1815	1796	1810
TA40	30 × 15	(1674)	1720	1725	1698	1714
TA41	30 × 20	(2018)		2045	2047	2071
TA42	30 × 20	(1949)		1979	1970	1984
TA43	30 × 20	(1858)		1898	1899	1928
TA44	30 × 20	(1983)		2036	2019	2039
TA45	30 × 20	(2000)		2021	2010	2032
TA46	30 × 20	(2015)		2047	2041	2070
TA47	30 × 20	(1903)		1938	1935	1958
TA48	30 × 20	(1949)	2001	1996	1994	2022
TA49	30 × 20	(1967)		2013	1992	2015
TA50	30 × 20	(1926)		1975	1975	1998
TA51	50 × 15	2760		2760	2760	2760

Table 4 (*continued*)

Problem	Size ($n \times m$)	Optimal solution (or upper bound)	TSAB	TSSB	HPSO	
			Nowicki and Smutnicki (1996)	Pezzella and Merelli (2000)	Best solution	Average
TA52	50 × 15	2756		2756	2756	2758
TA53	50 × 15	2717		2717	2717	2717
TA54	50 × 15	2839		2839	2839	2840
TA55	50 × 15	2679	2679	2684	2679	2694
TA56	50 × 15	2781		2781	2781	2785
TA57	50 × 15	2943		2943	2943	2943
TA58	50 × 15	2885		2885	2885	2885
TA59	50 × 15	2655		2655	2655	2666
TA60	50 × 15	2723		2723	2723	2732
TA61	50 × 20	2868	2868	2868	2868	2896
TA62	50 × 20	2869	2902	2942	2930	2958
TA63	50 × 20	2755	2755	2755	2755	2774
TA64	50 × 20	2702	2702	2702	2702	2718
TA65	50 × 20	2725	2725	2725	2735	2759
TA66	50 × 20	2845	2845	2845	2848	2869
TA67	50 × 20	2825	2841	2865	2840	2861
TA68	50 × 20	2784	2784	2784	2784	2802
TA69	50 × 20	3071	3071	3071	3071	3096
TA70	50 × 20	2995	2995	2995	3005	3041
TA71	100 × 20	5464		5464	5519	5595
TA72	100 × 20	5181		5181	5211	5305
TA73	100 × 20	5568		5568	5581	5655
TA74	100 × 20	5339		5339	5355	5412
TA75	100 × 20	5392		5392	5466	5563
TA76	100 × 20	5342		5342	5396	5504
TA77	100 × 20	5436		5436	5444	5493
TA78	100 × 20	5394		5394	5394	5476
TA79	100 × 20	5358		5358	5363	5434
TA80	100 × 20	5183	5183	5183	5209	5364
Average Gap			0.7792%	0.8122%	0.5659%	1.4651%
# of instance			33	80	80	
# of BKS obtained			12	31	27	

Balas, & Zawack, 1988; Balas & Vazacopoulos, 1998), Tabu Search (Nowicki & Smutnicki, 1996; Pezzella & Merelli, 2000; Sun et al., 1995), and Genetic Algorithm (Gonçalves et al., 2005; Wang & Zheng, 2001).

The computational results of FT and LA test problems are shown in Table 2. The results show that the preference list-based PSO we proposed is much better than the original design, the priority-based PSO. Since the number of instances tested by each method is different, we cannot compare the result by average gap directly. Nevertheless, the result obtained by HPSO is better then other algorithms that tested all of the 43 instances, and the HPSO obtained the best-known solution for 41 of the 43 instances.

Table 3 shows the average computation time on FT and LA test problems in CPU seconds. The 'best-solution time' is the average time that the algorithm takes to first reach the final best solution, and the 'total time' is the average total computation time that the algorithm takes during a run. In HPSO, there is about 99% computation time spent on local search process. As mentioned in Section 3.5, the solution quality of tabu search (Nowicki & Smutnicki, 1996) is mainly affected by its initial solution, and the main purpose of the PSO process is to provide good and diverse initial solutions to tabu search. Therefore, the computational results show that the hybrid method, HPSO, performs better than both TSAB and PSO, and its average gap is 0.356% less than PSO.

Table 5
Comparison with TSSB (Pezzella & Merelli, 2000) on TA test problems

Problem	Size ($n \times m$)	TSSB[a]		HPSO[b]		
		Average gap (%)	Total time	Average gap (%)	Time to get best solution	Total time
TA01-10	15×15	0.4502	2175	0.0726	99	514
TA11-20	20×15	1.1537	2526	0.5023	345	855
TA21-30	20×20	1.0840	34910	0.7029	401	1238
TA31-40	30×15	1.4475	14133	0.7654	1185	2026
TA41-50	30×20	1.9474	11512	1.6133	1734	2769
TA51-60	50×15	0.0187	421	0.0000	565	2909[c]
TA61-70	50×20	0.3960	6342	0.3463	2322	2862[c]
TA71-80	100×20	0.0000	231	0.5244	2797	3137[c]
Total average gap		0.8122		0.5659		

[a] Run on a Pentium 133 MHz PC.
[b] Run on an AMD Athlon 1700 + PC.
[c] Decreasing the percentage to perform local search procedure reduces the computation time.

We further tested HPSO on TA test problems (Taillard, 1993). The computational results are shown in Table 4, and we particularly compared HPSO with TSSB (Pezzella & Merelli, 2000) in Table 5. Since the maximum computation time of TSSB is about 3×10^4 s and our machine is about ten times faster then TSSB (Pezzella & Merelli, 2000), we limited the maximum computation time of HPSO in 3×10^3 s. As mentioned above, 99% of the computation time is spent on the local search process in HPSO. Therefore, we do not reduce the computation time by decreasing the number of iterations, but decreasing the percentage of particles that perform a local search procedure. The HPSO will also be terminated after 10^3 iterations, but there are only 34.6% of particles randomly chosen to perform the local search procedure in each iteration on TA51 to TA60 test problems, 26.6% on TA61 to TA70 test problems, and 6.4% on TA71 to TA80 test problems.

Table 5 shows the comparison with TSSB (Pezzella & Merelli, 2000). The HPSO performs better than TSSB on 7 of 8 problem sizes, and only worse than TSSB on the 100×20 problem size. In the 100×20 problem sizes, the final best solutions are obtained after 890 iterations of the average (so the best-solution time is very close to the total time). Since the HPSO only performs 10^3 iterations for each run, it shows that the particles of HPSO did not converge in 10^3 iterations, and can further improve the solutions by increasing the maximum iteration. However, since we want to compare HPSO with TSSB, we do not consider increasing the maximum iteration because it takes too much computation time.

5. Conclusions

We have presented a hybrid particle swarm optimization (HPSO) for job shop scheduling problems in this paper. We modified the representation of particle position, particle movement, and particle velocity to better suit it for JSP. We also applied Tabu Search to improve solution quality. The computational results show that HPSO can obtain better solutions than other methods.

For further research, if the HPSO we proposed is implemented to other sequential ordering problems, there are two aspects for discussion: (1) Modify particle position representation for better suitability to the problem. In the original PSO design, the particles search solutions in a continuous solution space. Although most sequential ordering problems can be represented by the priority-based representation, it may not suit the sequential ordering problems that we illustrated in Section 3.1.1. Preference list-based representation or other representations will better suit the algorithm for sequential ordering problems. (2) Design other particle movement methods and particle velocity for the modified particle position representation. Besides, which particle movement method or particle velocity is better could be a further research topic.

Appendix A

A pseudo code of the HPSO for JSP is given below:

initialize a population of particles with random positions.
for each particle k ***do***
  apply G&T algorithm to decode X^k (the position of particle k) into a schedule S^k.
  set the kth pbest solution ($pbest_k$) equal to S^k, $pbest^k \leftarrow S^k$.
end for
set gbest solution equal to the best $pbest^k$.
repeat
  update velocities according to Fig. 2.
  for each particle k ***do***
    move particle k according to Fig. 4.
    apply G&T algorithm to decode x^k into S^k.
    update pbest solutions and gbest solution according to Fig. 5.
    apply tabu search on S^{k^*}.
    update pbest solutions and gbest solution according to Fig. 5.
  end for
until maximum iterations is attained.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science, 34*(3), 391–401.

Balas, E., & Vazacopoulos, A. (1998). Guided local search with shifting bottleneck for job shop scheduling. *Management Science, 44*(2), 262–275.

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, 14*, 1069–1072.

Davis, L. (1985). Job shop scheduling with genetic algorithm. In J. J. Grefenstette (Ed.), *Proceedings of the first international conference on genetic algorithms* (pp. 140–163). Hillsdale, NJ: Lawrence Erlbaum Associates.

Fisher, H., & Thompson, G. L. (1963). *Industrial scheduling*. Englewood Cliffs, NJ: Prentice-Hall.

French, S. (1982). *Sequencing and scheduling: An introduction to the mathematics of the job-shop*. UK: Horwood.

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

Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York: Wiley.

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

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*, 533–549.

Glover, F. (1989). Tabu search: Part I. *ORSA Journal on Computing, 1*, 190–206.

Glover, F. (1990). Tabu search: Part II. *ORSA Journal on Computing, 2*, 4–32.

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., Rangaswamy, B., & Meeran, S. (2000). New and "stronger" job-shop neighbourhoods: A focus on the method of Nowicki and Smutnicki (1996). *Journal of Heuristics, 6*, 457–480.

Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. *Proceedings of the 1995 IEEE international conference on neural networks* (Vol. 4, pp. 1942–1948). Piscataway, NJ: IEEE Press.

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.

Lawrence, S., (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration (GSIA), Carnegie Mellon University, Pittsburgh, PA.

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.

Shi, Y., & Eberhart, R. C. (1998a). Parameter selection in particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, & A. E. Eiben (Eds.), *Proceedings of the 7th international conference on evolutionary programming* (pp. 591–600). New York: Springer.

Shi, Y., & Eberhart, R. C. (1998b). A modified particle swarm optimizer. In D. Fogel (Ed.), *Proceedings of the 1998 IEEE international conference on evolutionary computation* (pp. 69–73). Piscataway, NJ: IEEE Press.

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

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

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

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