Expert Systems with Applications 37 (2010) 2629–2636

Contents lists available at ScienceDirect

Expert Systems with Applications

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

An efficient job-shop scheduling algorithm based on particle swarm optimization ☆

Tsung-Lieh Lin [a], Shi-Jinn Horng [a,b,c,e,*], Tzong-Wann Kao [d], Yuan-Hsin Chen [c], Ray-Shine Run [c], Rong-Jian Chen [c], Jui-Lin Lai [c], I-Hong Kuo [a]

[a] Department of Electrical Engineering, National Taiwan University of Science and Technology, 106 Taipei, Taiwan
[b] Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, 106 Taipei, Taiwan
[c] Department of Electronic Engineering, National United University, 36003 Miao-Li, Taiwan
[d] Department of Electronic Engineering, Technology and Science Institute of Northern Taiwan, Taipei, Taiwan
[e] Department of Computer Science, Georgia State University, United States

ARTICLE INFO

Keywords:
Job-shop scheduling problem
Particle swarm optimization
Multi-type individual enhancement scheme
Random-key encoding scheme
Simulated annealing

ABSTRACT

The job-shop scheduling problem has attracted many researchers' attention in the past few decades, and many algorithms based on heuristic algorithms, genetic algorithms, and particle swarm optimization algorithms have been presented to solve it, respectively. Unfortunately, their results have not been satisfied at all yet. In this paper, a new hybrid swarm intelligence algorithm consists of particle swarm optimization, simulated annealing technique and multi-type individual enhancement scheme is presented to solve the job-shop scheduling problem. The experimental results show that the new proposed job-shop scheduling algorithm is more robust and efficient than the existing algorithms.

1. Introduction

The job-shop scheduling problem (JSSP) is one of the existing combinatorial optimization problems and it has been demonstrated to be an NP-hard problem (Garey, Johnson, & Sethi, 1976; Lawer, Lenstra, Rinooy Kan, & Shmoys, 1993). In the job-shop scheduling problem, each one of n jobs ($n \geqslant 1$) must be processed passing through m machines ($m \geqslant 1$) in a given sequence. The sequence of m machines is different for each different job and cannot be changed during the processing. When one job was processed on a machine, it can be considered as one operation, each job j ($1 \leqslant j \leqslant n$) needs a combination of m operations $(o_{j1}, o_{j2}, \ldots, o_{jm})$ to complete the work. One operation is processed on one of m machines, and just only one operation can be processed at a time. Any job cannot interrupt the machine that is processing one operation of another job. Each machine can process at most one operation at the same time. The main objective of the job-shop scheduling problem is to find a schedule of operations that can minimize the maximum completion time (called makespan) that is the completed time of carrying total operations out in the schedule for n jobs and m machines.

JSSP can be applied to the manufacture processing and effects really the production time and the cost of production for a plant. During the past few decades, JSSP has attracted many researchers to develop algorithms. Because JSSP is an NP-hard problem, it is difficult to develop a perfect algorithm to find a solution within a reasonable time especially for higher dimensions. Recently, many researchers made use of evolution algorithm to solve the problem, such as tabu search method (Nowicki & Smutnicki, 2005; Ponnambalam, Aravindan, & Rajesh, 2000), genetic algorithm (Goncalves, Mendes, & Resende, 2005; Park, Choi, & Kim, 2003; Wang & Zheng, 2001; Watanabe, Ida, & Gen, 2005), simulated annealing (Van Laarhoven, Aarts, & Lenstra, 1992; Steinhöel, Albrecht, & Wong, 1999; Suresh & Mohanasundaram, 2005), ant colony (Udomsakdigool & Kachitvichyanukul, 2008; Zhou, Li, & Zhang, 2004) and particle swarm optimization (Ge, Du, & Qian, 2007; Ge, Sun, Liang, & Qian, 2008; Lian, Gu, & Jiao, 2006). In this paper, we focus on exploiting particle swarm optimization algorithm to achieve the better solution for JSSP.

Particle swarm optimization (PSO) is developed by Kennedy and Eberhart (Kennedy & Eberhart, 1995). The position of one particle is corresponding to a solution of the solving problem. Liking a bird that flies to the food, one particle moves its position to a better solution according to the best particle's experience and its own experience. Every particle moves iteratively until the end of iterations. We call this process as evolution process. At the end of iterations, the position of best particle is the best solution of the solving problem. The original developed PSO is designed to search

☆ This work was supported in part by the National Science Council under Contract Number NSC 96-2918-I-011-002, 97-2221-E-239-022-, 95-2221-E-011-032-MY3.

* Corresponding author. Address: Department of Electrical Engineering, National Taiwan University of Science and Technology, 106 Taipei, Taiwan. Tel.: +886 2 27376700; fax: +886 2 27301081.

E-mail addresses: D8907305@mail.ntust.edu.tw (T.-L. Lin), horngsj@yahoo.com.tw (S.-J. Horng), tkao@ms6.hinet.net (T.-W. Kao), yschen@nuu.edu.tw (Y.-H. Chen), run5116@ms16.hinet.net (R.-S. Run), rjchen@nuu.edu.tw (R.-J. Chen), jllai@nuu.edu.tw (J.-L. Lai), yihonguo@gmail.com (I-Hong Kuo).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.08.015

solution in a continuous space. Because PSO's local search ability is weaker than global searching ability, in order to get better solution, some local search schemes should be integrated with the PSO. In this paper, we embedded a multi-type individual enhancement scheme (MIE) based on simulated annealing technique into particle swarm optimization (PSO). The proposed algorithm enhances the particle's searching ability and is suitable to solve the JSSP. The experimental results show that the proposed PSO with multi-type individual enhancement scheme outperforms the original PSO and is more efficient than those of existing meta-heuristics methods such as discrete particle swarm optimization with simulated annealing model (named HEA (Ge et al., 2007)), discrete particle swarm optimization with artificial immune system (named HIA (Ge, Sun, Liang, & Qian, 2008)) and genetic algorithm (named HGA (Goncalves et al., 2005)) for JSSP scheduling problem, respectively.

The remainder of the paper is organized as follows: an introduction for the job-shop scheduling problem and particle swarm optimization are given in Sections 2 and 3, respectively. Section 4 gives a detailed description of the new proposed job-shop scheduling algorithm. Section 5 discusses the experimental results. Finally, Section 6 summarizes the contribution of this paper.

2. The job-shop scheduling problem

The job-shop scheduling problem (JSSP) consists of n jobs and m machines. Each job must go through m machines to complete its work. We consider one job consists of m operations. Each operation uses one of m machines to complete one job's work for a fixed time interval. Once one operation is processed on a given machine, it cannot be interrupted before it finishes the job's work. The sequence of operations of one job should be predefined and maybe different for any job. In general, one job being processed on one machine is considered as one operation noted as $o_{ji'}$ (means jth job being processed on i'th machine, $1 \leqslant j \leqslant n, 1 \leqslant i' \leqslant m$), then every job has a sequence of m operations. Each machine can process only one operation during the time interval. The objective of JSSP is to find an appropriate operation permutation for all jobs that can minimize the makespan C_{max}, i.e., the maximum completion time of the final operation in the schedule of $n \times m$ operations.

For an $n \times m$ JSSP, the problem can be modeled by a set of m machines, denoted by $\mathrm{M} = \{1, 2, \ldots, m\}$, to process a set of $n \times m$ operations, denoted by $o = \{0, 1, 2, \ldots, n \times m + 1\}$. The operations 0 and $n \times m + 1$, which are dummy operations, represent the initial and the last operations, respectively. Dummy operation is used to model the JSSP problem and need not any processing time. A precedence constraint is used to let operation i to be scheduled after all predecessor operations included in P_i are finished. Further, one operation can be scheduled on an appointed machine that is free. For the conceptual model, the notations are defined in the following:

n	number of jobs
m	number of operations for one job
O_i	completed time of operation i ($i = 0, 1, 2, \ldots, n \times m + 1$)
t_i	processing time of operation i on a given machine
ω_{im}	the flag of operation i initiates machine m
P_i	all predecessor operations of operation i
$A(t)$	the set of operations being processed at time t
$o_{ji'}$	i'th operation of job j
C_{max}	makespan

According to the description listed above, the conceptual model of the JSSP can be defined as follows (Goncalves et al., 2005):

$$\text{minimize} \quad O_{n \times m+1} \ (C_{\max}) \tag{1}$$

$$O_q \leqslant O_i - t_i, \quad i = 0, 1, 2, \ldots, n \times m + 1; \ q \in P_i \tag{2}$$

$$\sum_{i \in A(t)} \omega_{im} \leqslant 1, \quad m \in M, \quad t \geqslant 0 \tag{3}$$

$$O_i \geqslant 0, \quad i = 0, 1, 2, \ldots, n \times m + 1 \tag{4}$$

The objective fitness function in Eq. (1) is to minimize makespan that is the completion time of the last operation. The constraint of precedence relationship is defined by Eq. (2). In Eq. (3), it indicates that one machine can process at most one operation at a time. The finish time must be positive by the constraint stated in Eq. (4).

The following example illustrates the JSSP problem.

Suppose there are three jobs and two machines. The processing time and the initiated machine order of each operation are given in Table 1. The operation o_{j1} must be processed before o_{j2} for a job j. In Table 1b, operation o_{21} is processed on machine 2 for 3-unit time interval and operation o_{22} is processed on machine 1 for 1-unit time interval and the operation order of o_{21} should be preceded before that of o_{22}. An operation permutation, $(o_{11}, o_{21}, o_{22}, o_{31}, o_{32}, o_{12})$, is feasible because it satisfied with the operation ordering constraint as stated in Eqs. (1)–(4). O_1 is 2-unit time interval that is the finish time of operation o_{11} on machine 1. Then O_3 is 4-unit time interval as it is the summation of its own operation time and the maximal finished time of its predecessors. According to this permutation, the makespan of $(o_{11}, o_{21}, o_{22}, o_{31}, o_{32}, o_{12})$ is turned out to be 6-unit time interval. The resulting Gantt chart for operation permutation $(o_{11}, o_{21}, o_{22}, o_{31}, o_{32}, o_{12})$ is depicted in Fig. 1.

3. Particle swarm optimization

Particle swarm optimization(PSO) is a novel evolutionary algorithm that was inspired by the motion of a flock of birds searching for foods and was proposed by Kennedy and Eberhart for optimization of continuous non-linear problems (Kennedy & Eberhart, 1995). At the beginning of the evolutionary process, a set of parti-

Table 1
A 3×2 JSSP problem.

Job	Operations	
(*a*) *Operation index*		
Job1	o_{11}	o_{12}
Job2	o_{21}	o_{22}
Job3	o_{31}	o_{32}
(b) *Machine and time*		
Operation	Machine	Time
o_{11}	1	2
o_{12}	2	2
o_{21}	2	3
o_{22}	1	1
o_{31}	2	1
o_{32}	1	1

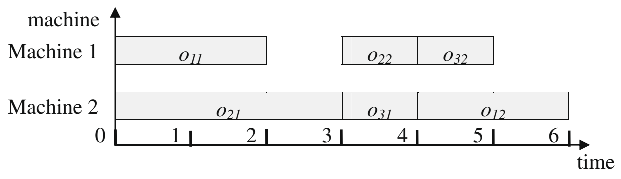

Fig. 1. Gann chart of $(o_{11}, o_{21}, o_{22}, o_{31}, o_{32}, o_{12})$.

cles we called it as a swarm must be initiated randomly. Each particle can change its position in the search space just like a flying bird searching the food in the sky. During the evolutionary process, a particle *id* of a swarm adjusts its newer moving velocity according to its best experience, the best experience of all particles in the swarm and the previous moving velocity. Then, the particle *id* moves to a new position according to newly generated velocity and its previous position. The following mathematical formula is used to describe how a particle keeps on moving and finding the optimal solution:

$$v_{id} = \omega \times v_{id} + C_1 \times Rand() \times (p_{id}^{best} - p_{id}) + C_2 \times Rand() \times \left(p_{gd}^{best} - p_{id}\right). \tag{5}$$

$$p_{id} = p_{id} + v_{id}. \tag{6}$$

In Eq. (5), v_{id} means the moving distance of *id*th particle on one iteration and is limited to $[-\mathrm{V}_{\max}, \mathrm{V}_{\max}]$ in which $\mathrm{V}_{\max}$ is the maximum moving distance in one particle's step. The variable ω called inertial weight is used to define a one step movement distance for a particle. C_1 is the self learning factor which means how much one particle will believe in its own best experience p_{id}^{best}. C_2 is the social learning factor which means one particle will believe in the global best experience p_{gd}^{best} of all particles of a swarm. *Rand*() is a function to generate a random number uniformly distributed in U(0,1). p_{id} means the position of *id*th particle. So, p_{id}^{best} is the personal best position of *id*th particle and p_{gd}^{best} is the global best position of all particles of a swarm. Based on Eq. (5), the newly generated velocity can be obtained for the *id*th particle, the position of the *id*th particle can be updated by Eq. (6).

The PSO algorithm is described as follows:

Algorithm 1. Particle swarm optimization (PSO) algorithm

```
1:  Initialize a population of N particles with random
    positions and random velocities with D dimensions in a
    given searching space.
2:  while a specified stop condition (the optimal solution is
    found or the maximal number of iterations is reached) is
    not met do
3:    Evaluate the fitness of each particle in the populations
    according to the objective function of the problem.
      for each particle id do
4:      Update the personal best position ($p_{id}^{best}$) for each
    particle and the global best position ($p_{gd}^{best}$) for all
    particles.
5:      By Eq. (5), update the velocity of each particle.
6:      By Eq. (6), update the position of each particle.
7:    end for
8:  end while
```

In order to improve the solution quality, the inertial weight ω, the cognition learning factor C_1 and the social learning factor C_2 are varying with time, not using the fixed coefficients. PSO algorithm can get better solution by letting ω be varying from higher to lower by the increasing of time (Kuo et al., 2009; Xia & Wu, 2005). Let ω be varying with time by the following linear decreasing function.

$$\omega = \omega_{\max} - Iter \times \frac{\omega_{\max} - \omega_{\min}}{MaxIter}. \tag{7}$$

where *MaxIter* is the maximum iterations during the evolutionary process, *Iter* is the current iteration number, ω_{max} is the initial value of ω and ω_{min} is the final value of ω, respectively.

4. The proposed algorithm for JSSP

Some issues are in applying PSO algorithm to solve JSSP. The original PSO design is developed to solve continuous function. But, JSSP is a combinatorial problem, the solution space is discrete. The first issue is to find a suitable representation which the particles of PSO can simulate an operation permutation schedule of JSSP. In this paper, based on an operation permutation, the continuous PSO combined with a random-key (RK) encoding scheme is used to solve the first issue. The detailed description will be discussed in Section 4.1. The second issue is how to enhance PSO's local search ability by applying PSO to solve the combinatorial problems. No matter applying CPSO (continuous PSO) or DPSO (discrete PSO) to the combinatorial problem, embedding the local search ability in PSO algorithm is an effective way to get a better solution (Ge et al., 2008; Kuo et al., 2009; Liao, Tseng, & Luarn, 2007; Xia & Wu, 2005). A multiple-type individual enhancement scheme based on SA (simulated annealing algorithm) is applied to enhance the local search ability of PSO. The detailed description is stated in Section 4.3. The complete algorithm named MPSO is shown in Section 4.5, which consists of random-key (RK) encoding scheme, multiple-type individual enhancement scheme based on SA and particle swarm optimization.

4.1. Representation of a particle

The searching space is created in an $n \times m$ dimensions space for n jobs on m machines JSSP. The position of a particle consists of $n \times m$ dimensions and is represented with $n \times m$ real numbers. In order to simulate an operation permutation sequence of JSSP, the $n \times m$ real numbers are transformed into an integer series from 1 to $n \times m$ by the random-key encoding scheme. Each integer number represents one operation index $(o_{ji'}, 1 \leqslant j \leqslant n, 1 \leqslant i' \leqslant m)$ according to its ordering in all $n \times m$ real numbers. Fig. 2 illustrates an example of a representation of a particle for a 3×2 JSSP.

4.2. Random key encoding scheme

The random-key (RK) encoding scheme can be used to transform a position in RK continuous space to a discrete space. A vector in RK space consists of real numbers. According to RK scheme, one particle represented by real numbers can simulate an operation permutation that consists of discrete numbers. For *n* jobs on *m* machines JSSP, the RK virtual space is created in $n \times m$ dimensions, that is, one particle is represented as $\{R_j | R_j$ is a real number, $1 \leqslant j \leqslant n \times m\}$, R_j is the corresponding weight of an operation order. During the process of the random key encoding scheme, a real vector is sorted in an ascending order with an integer series $(\pi_1, \pi_2, \ldots, \pi_k)$ that each integer π_k is between 1 and $n \times m$, $1 \leqslant k \leqslant n \times m$. Each integer π_k indirectly represents an operation order of a job. Because each job must go through *m* machines to complete its work, a job must have *m* operations that are scheduled in a predecessor constraint. According to this constraint, we can easily make further transformation from an integer series $(\pi_1, \pi_2, \ldots \pi_k, \ldots, \pi_{nm})$ to the job index by $(\pi_k \bmod n) + 1$, where *n* is the number of jobs. For $\pi_k \in (n, 2n, \ldots, n \times m)$, it means the listed operations belonging to job1. Similarly, for $\pi_k \in (1, n+1, \ldots, (m-1) \times n + 1)$, it means the listed operations belonging to job 2, and so on. Through this transformation, the integer series $(\pi_1, \pi_2, \ldots \pi_k, \ldots, \pi_{nm})$ can be transformed into an operation order sequence, $(\lambda_1, \lambda_2, \ldots, \lambda_k, \ldots, \lambda_{nm})$, where λ_k represents a job index, $1 \leqslant \lambda_k \leqslant n$. We scan this permutation

particle	1.3	0.7	2.4	1.1	5.3	3.4

Fig. 2. A representation of a particle for a 3×2 JSSP.

$(\lambda_1, \lambda_2, \ldots, \lambda_k, \ldots, \lambda_{nm})$ from left to right, then each job index has m occurrences. By scanning, the ith occurrence of a job index is corresponding to the ith operation in the m operations of a job. The operation permutation is always feasible because the operation order is satisfied with the predecessor constraint.

We illustrate a complete example to show the processing from RK virtual space to a feasible operation permutation in the JSSP solution space as shown in Fig. 3. For a 3×2 JSSP, suppose that the position of a particle in RK virtual space is (1.3, 0.7, 2.4, 1.1, 3.4, 5.3). It can be encoded to an integer series (3, 1, 4, 2, 5, 6) by sorting the 3×2 real numbers in an ascending order (For example, 0.7 is the smallest number, it is then ranked to 1). In this integer series, the integers 3 and 6 indicate the operations belonging to job 1 because (3 mod 3) + 1 = 1, (6 mod 3) + 1 = 1. The integers 1 and 4 indicate the operations belonging to job 2, because (1 mod 3) + 1 = 2, (4 mod 3) + 1 = 2. The integers 2 and 5 indicate the operations belonging to job3 by (2 mod 3) + 1 = 3 and (5 mod 3) + 1 = 3, respectively. Then, an operation permutation (1, 2, 2, 3, 3, 1) corresponding to job indexes is obtained. By scanning (1, 2, 2, 3, 3, 1) from left to right, the first 1 means the first operation of job 1, corresponding to o_{11}, the second 1 means the second operation of job 1, corresponding to o_{12}. According to this scanning process, the partial series (2, 2) is corresponding to $(o_{21},\ o_{22})$ and (3, 3) is corresponding to $(o_{31},\ o_{32})$. After scanning the job index series from left to right, the permutation (1, 2, 2, 3, 3, 1) is corresponding to an operation sequence $(o_{11},\ o_{21},\ o_{22},\ o_{31},\ o_{32},\ o_{12})$. The operation sequence represented by this encoding scheme is always a feasible solution of JSSP and the representation of a particle is easily to proceed the local searching which will be explained on the multi-type individual enhancement scheme. An example from the RK virtual space to an operation permutation is shown in Fig. 3. It is obvious that a vector in RK space is corresponding to a particle of PSO.

4.3. Multiple-type individual enhancement scheme

In order to enhance the local search ability and get a better solution, Kuo et al. (2009) proposed an individual enhancement scheme which is to exchange the job order of two jobs for solving the FSSP problem. In this paper, we developed a new multiple-type individual enhancement scheme for JSSP problem. A multiple-type individual enhancement scheme is composed of swapping operation, insertion operation, inversion operation and long-distance movement operation which can be used to search an individual's neighborhood to get a better solution. Swapping operation scheme is to swap two weighting numbers that indirectly represent two operations in the pth and qth dimension $(p \neq q)$ of an individual in RK virtual space. Insertion operation is to remove the one in the pth dimension and insert it into the qth dimension $(p \neq q)$ of an individual. In general, it is enough to get a better solution for most problems by using these two types of enhancement scheme. By the experimental experience, it needs a scheme to jump away from the local optimal for some hard problems which have higher dimensions. So, we incorporated another two types of enhancement scheme to the proposed algorithm. The inversion operation scheme is to pick two dimensions p and q $(p \neq q)$ first and invert the weighting numbers between them.

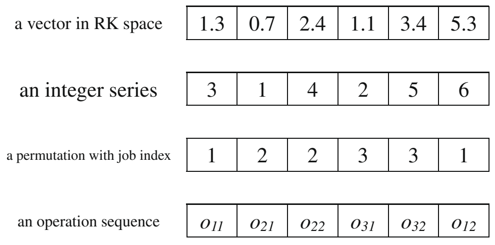

Fig. 3. An example for the RK encoding scheme.

The last enhancement scheme is the long distance movement operation. At first, pick two dimensions p and q $(p \neq q)$ of an individual, remove all weighting numbers between them and insert these removed weighting numbers to the place where it begins at the rth dimension $(r \neq p \neq q, r \notin [p, q])$.

The process of multiple-type individual enhancement scheme is to select an operation scheme from multiple-type individual enhancement scheme, to operate on an individual, and to compare the makespan obtained before the selected scheme and that obtained after the selected scheme. If the latter is better than the former, update the real vector of the individual by the selected operation scheme. If not, the new real vector can be accepted and updated according to a threshold that is generated by the simulated annealing algorithm (SA). If a random probability is less than a threshold, the new real vector can be accepted and updated; otherwise, drop the new real vector and keep the previous real vector as a next position to proceed the local search operation. After finishing one scheme, continue to select another scheme to operate on the individual until it meets the stop condition.

An example is given to explain the operation scheme based on the RK encoding scheme in Figs. 4–7. In Fig. 4, the individual (0.7, 1.3, 2.4, 1.1, 3.4, 5.3) is obtained by swapping two items located at the second dimension and the third dimension from the individual (0.7, 2.4, 1.3, 1.1, 3.4, 5.3). According to the RK encoding scheme, the individual can be transformed to an operation permutation with job indexes. Like Fig. 3, the makespan of (0.7,**2.4**, **1.3**, 1.1, 3.4, 5.3) is 8 and that of (0.7, **1.3**, **2.4**, 1.1, 3.4, 5.3) is changed to 6. For the Makespan, the latter is better than the former. Hence, after swapping, the position of this individual is updated from (0.7,**2.4**, **1.3**, 1.1, 3.4, 5.3) to (0.7, **1.3**, **2.4**, 1.1, 3.4, 5.3). In Fig. 5, the individual ($\underline{1.1}$, $\underline{2.3}$, $\underline{4.6}$, **3.7**, 6.5, 5.1) is obtained by inserting the item 3.7 in the first dimension to the 4th dimension in the individual (**3.7**, $\underline{1.1}$, $\underline{2.3}$, $\underline{4.6}$, 6.5, 5.1). The makespan of (1.1, 2.3, 4.6, 3.7, 6.5, 5.1) equals to 8. The makespan of (3.7, 1.1, 2.3, 4.6, 6.5, 5.1) equals to 6. Assume the probability exceeds the threshold evaluated by SA, we drop the position (1.1, 2.3, 4.6, 3.7, 6.5, 5.1) and keep the individual on position (3.7, 1.1, 2.3, 4.6, 6.5, 5.1). In Fig. 6, it illustrates the inversion operation scheme. By inversing the weighting numbers between the 3rd dimension and the 6th dimension of the individual (1.8, 0.9, **2.7, 1.6, 5.5, 3.3**), we get the new position of this individual as (1.8, 0.9, **3.3, 5.5, 1.6, 2.7**). In Fig. 7, for the long distance movement operation scheme, the individual (**5.8, 3.2**, 1.9, 0.5, 2.9, 1.4) is obtained by moving the segment between dimensions 5 and 6 in individual (1.9, 0.5, 2.9, 1.4,

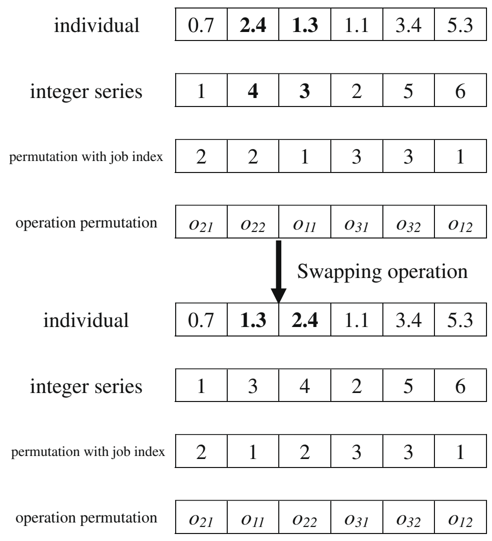

Fig. 4. Swapping operation scheme, $p = 2$ and $q = 3$.

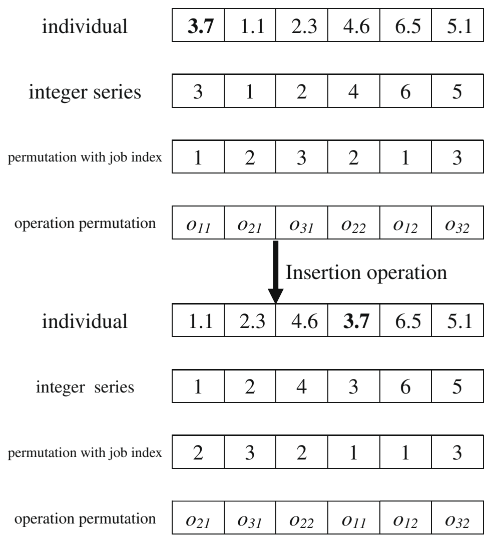
individual 3.7 1.1 2.3 4.6 6.5 5.1
integer series 3 1 2 4 6 5
permutation with job index 1 2 3 2 1 3
operation permutation o11 o21 o31 o22 o12 o32
Insertion operation
individual 1.1 2.3 4.6 3.7 6.5 5.1
integer series 1 2 4 3 6 5
permutation with job index 2 3 2 1 1 3
operation permutation o21 o31 o22 o11 o12 o32

Fig. 5. Insertion operation scheme, $p = 1$ and $q = 4$.

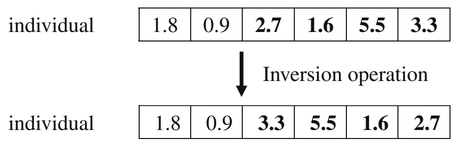
individual 1.8 0.9 2.7 1.6 5.5 3.3
Inversion operation
individual 1.8 0.9 3.3 5.5 1.6 2.7

Fig. 6. Inverse operation scheme between $p = 3$ and $q = 6$.

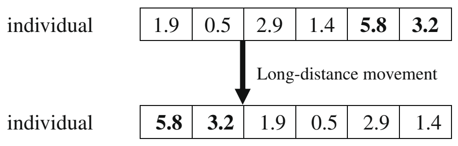
individual 1.9 0.5 2.9 1.4 5.8 3.2
Long-distance movement
individual 5.8 3.2 1.9 0.5 2.9 1.4

Fig. 7. Long distance movement operation scheme, $p = 5$, $q = 6$ and $r = 1$.

5.8, 3.2) to the first dimension of this individual. Like before, the makespan of the new position of an individual in Figs. 6 and 7 can be evaluated as in Figs. 4 and 5.

A partial algorithm about 4 type operations is listed in Algorithm 2. $Prob_s$ means the probability of executing the swapping scheme; $Prob_i$ means the probability of executing the insertion scheme; $Prob_{inv}$ means the probability of executing the inversion scheme; $Prob_{long}$ means the probability of executing the long distance movement scheme, respectively.

Algorithm 2. the operation of multi-type individual enhancement

```
Input: p, the individual to be enhanced
Output: p', one individual after executing multi-type
  individual enhancement
1:    $q \leftarrow rand()$
2:    if $(0 \leqslant q \leqslant Prob_s)$ then execute swapping scheme for
      individual p
3:    else if $(Prob_s < q \leqslant Prob_s + Prob_i)$ then execute inserting
      scheme for individual p
4:    else if $(Prob_s + Prob_i < q \leqslant Prob_s + Prob_i + Prob_{inv})$ then
      execute inversion scheme for individual p
5:        //Finally, (q will match with $Prob_{long}$)
          else execute long distance movement scheme for
      individual p
6:    end if
```

For example, suppose that $Prob_s = 0.4$, $Prob_i = 0.4$, $Prob_{inv} = 0.1$, and $Prob_{long} = 0.1$, if rand() = 0.33, the individual $p$will be enhanced by swapping scheme; if rand() = 0.66, the individual p will be enhanced by inserting scheme; if rand() = 0.88, the individual p will be enhanced by inversion scheme; if rand() = 0.98,the individual p will be enhanced by long distance movement scheme.

4.4. Simulated annealing

An algorithm simulates ideas and mechanism in the annealing of solids is named simulated annealing (SA). Since its introduction by Kirkpatrick, Gelatt and Vecchi in 1984, the simulated annealing algorithm(SA) has been successfully applied to many combinatorial optimization problems (Kirkpatrick, Gelatt, & Vecchi, 1984). The key function of SA is to allow occasional alternations to accept worsened solutions in order to increase the probability of jumping away from a local optimum and getting a better solution. SA algorithm starts from an initial state s, the system is perturbed randomly to a neighboring state s' by applying a suitable operation on state s. Two objective functions $f(s)$ and $f(s')$ are evaluated, respectively. For a minimization problem, s' is accepted as a new state if the difference of two objective functions, $\Delta = f(s') - f(s)$, is < 0. If $\Delta = f(s') - f(s) \geqslant 0$, the new state s' is accepted with probability given by $\min\left\{1, \exp^{\frac{-\Delta}{T}}\right\}$, where T is a control parameter referred as temperature. The temperature T is defined by user, and T is decreased iteration by iteration according to a referred cooling schedule from high to low. The SA algorithm is executed from high temperature until T is lower than a user-defined final temperature T_f which is a value near to zero. By SA algorithm, we can decide whether to accept an individual that is enhanced by Algorithm 2 but its makespan is not better than the individual not being enhanced by Algorithm 2 or not. For an enhanced individual that did not make improvement for makespan, if one random probability is smaller than $\min\left\{1, \exp^{\frac{-\Delta}{T}}\right\}$, the individual's position can be accepted as a new position of the individual; otherwise, we drop the position and keep the previous position for the individual. A complete multi-type individual enhancement scheme based on simulated annealing algorithm (SA) is listed in the following.

Algorithm 3. Multi-type individual enhancement scheme

```
Input: P, the individual to be enhanced; a starting
  temperature T; a final temperature $T_f$; a cooling rate $\beta$
Output: an enhanced individual
1:      Makespan(P) $\leftarrow$ makespan of an operation
        permutation represented by P
2:      while $(T > T_f)$ do
3:          Randomly select an operation from the multi-type
        individual enhancement scheme (MIE) by Algorithm 2,
        and generate a new individual P' by the selected
        operation.
4:          Makespan(P') $\leftarrow$ makespan of an operation
        permutation represented by P'
5:          $\Delta \leftarrow$ Makespan(P') – Makespan(P)
6:          if $(\Delta > 0)$ then   //P' is worse than P
              // randomly generate a probability rand() to accept
        the worse P' with a probability $\exp^{-\frac{\Delta}{T}}$
7:            if $\left(R = \text{rand}() < \min\left\{1, \exp^{-\frac{\Delta}{T}}\right\}\right)$ then
8:            $P \leftarrow P'$   //update P to be a enhanced P'
9:            Makespan(P) $\leftarrow$ Makespan(P')
10:         end if
11:     else
12:         $P \leftarrow P'$   // to accept a better P'
13:           Makespan(P) $\leftarrow$ Makespan(P')
14:           $T \leftarrow \beta \times T$
15:         end if
16:     end while
```

4.5. The MPSO algorithm

In this paper, we integrated random-key (RK) encoding scheme, multi-type individual enhancement scheme (MIE) into particle swarm optimization (PSO), named it as MPSO to solve the job-shop scheduling problem. In MPSO, a particle is represented by a real vector as shown in Fig. 2. Every particle moves its position in the RK virtual space by Eqs. (5 and 6), and the objective function of one particle corresponding to the solution space of JSSP can be evaluated by the transformation from RK space to a solution space of JSSP. For increasing the local search ability of PSO, MIE is used as an effective way to search the local neighborhood of one particle in the solution space of JSSP. The RK encoding scheme provides a search space for the continuous particle swarm optimization (PSO) and an easy way to encode the representation of PSO. According to the RK encoding scheme, we enhance the particle by MIE scheme that is corresponding to make a local search for the particle. One particle is selected with a probability $Prob_{MIE}$ as an individual to be enhanced in the MIE algorithm. After MIE algorithm, the selected particles can be in a better position than the previous one. Then, each particle of the swarm moves to a new position according to Eqs. (5) and (6). The process of MIE scheme and PSO Algroithm is executed until it gets the optimal solution or the maximum iteration number.

Algorithm 4. MPSO Algorithm

```
Input: $Prob_{MIE}$, the probability to execute the multi-type
   individual enhancement scheme; set $c1,\ c2,\ \omega$, $MaxIter$
Output: one best operation permutation schedule represented
   by the global best
1:   Initialize the position and velocity for all particles of a
     swarm
2:   while the stop condition(the optimal solution is found
     or the maximal number of iteration is reached) is not
     met do
3:     for each particle $id$ do
4:       if $S = rand() \leqslant Prob_{MIE}$ then
5:         Execute the multi-type individual enhancement
     scheme shown in Algorithm 3 for particle $id$
6:       end if
7:     Update the local best of each particle
8:     end for
9:     Update the global best of the swarm
10:    Update the $\omega$ according to Eq. (7)
11:    for each particle $id$ do
12:      Move particle$id$ to the next position according to
     Eqs. (5 and 6) with new $\omega$
13:    end for
14:  end while
```

5. Experimental results

In this paper, we use 43 instances that are taken from the OR-Library (Beasley, 1990) as test benchmarks to test our new proposed algorithm, named MPSO. In the 43 instances, FT06, FT10, and FT20 were designed by Fisher and Thompson (Fisher & Thompson, 1963) and instances LA01–LA40 that were designed by Lawerence (Lawrence, 1984). We coded the MPSO algorithm in ANSI C language with the environment of Microsoft Visual C++ 6.0, and simulated it with a 1.73GHz Intel Pentium M PC. The parameters used during the experimental process in Eqs. (5 and 6) are defined in the following. $C_1 = 2.0, C_2 = 2.0, \omega$ (the inertia weight) is decreased linearly from (ω_{max}) 1.4 to (ω_{min}) 0.4 for a run. The maximum of velocity V_{max} is $n \times m \times 0.1$, the maximum of position is limited at $n \times m$, and the population size of the swarm is set to 30. The probability to run the multi-type individual enhancement scheme (MIE) shown in Algorithm 3 during the whole running procedure of MPSO algorithm is set to be 0.01. The initial temperature T is set to be the difference between the makespan of selected particle and the best known solution, T_f is set to be 0.1, and β is set to be 0.97. Every instance is executed by MPSO for 10 runs. Most of the 43 instances only need 300 iterations in each run, but a few of them need 500 iterations in each run.

The evaluated experimental results compared with the results in HIA (Ge et al., 2008), HEA (Ge et al., 2007) and HGA (Goncalves et al., 2005) are listed in Table 2. In Table 2, instance means the problem name, size means the problem size n jobs on m machines, BKS means the best known solution for the instance, Best means the best solution found by each algorithm, and RD means the percentage of the deviation with respect to the best known solution for MPSO and HIA algorithms, respectively. In Table 2, the boldface represents the better solution for one instance that at least one of the four algorithms cannot obtain the best known solution. According to Table 2, MPSO can find the best known solution with 35 instances that is much better than HIA, HEA and HGA. For instances LA24, LA25, LA27 to LA29 and LA36 to LA40, the deviations between the best minimum founded solution (Best) and the best known solution (BKS) are all less than the deviation results of HIA. Except for LA 37, MPSO can obtain better solution for instances LA24, LA25, LA27 to LA29, LA36, LA38 to LA40 than HGA. Obviously, the experimental results show that MPSO is more efficient than other existing discrete particle swarm optimization and genetic algorithms, respectively.

We use the same parameters in Eqs. (5 and 6) to test the original PSO with random key encoding scheme and the proposed MPSO algorithm. We select instances FT06, FT10, FT20 and the first instance of other type instance set as test benchmark. Each instance is executed for 10 runs. The original PSO is executed 10^5 iterations for each run. The results are shown in Table 3. BKS and Best are the same meaning as those in Table 2. Max means the maximum founded solution by PSO and MPSO for 10 runs, Avg means the average of results for 10 runs, respectively. The first fact in Table 3 shows that MIE can provide a better searching ability than the original PSO. In general, the PSO is easy to be trapped in a local optimal and cannot find a better solution. From the results of Table 3, MIE effectively increases the local searching ability of the original PSO for the JSSP scheduling problems. Observing Table 3, the difference between the MPSO's Max and the BKS, and the difference between the MPSO's Avg and the BKS are within 2%. This fact shows that the solution of MPSO is quite stable.

In the above experimental results, we can see that MPSO can obtain the optimal area in the search space with smaller population size, and can get better solution by making use of the better individual enhancement ability.

6. Conclusions

In this paper, an algorithm called MPSO that combining random-key (RK) encoding scheme, multi-type individual enhancement scheme (MIE), and particle swarm optimization (PSO) is proposed to solve the NP-hard JSSP problem. For combinatorial optimization problems such as JSSP, the search space is a discrete space that used by HIA (Ge et al., 2008) and HEA (Ge et al., 2007). But, MPSO adopts the real space as the search space called random-key (RK) space. In RK space, a position of a particle composed of $n \times m$ real numbers can represent the permutation of all operations of all jobs by the encoding scheme. It is very different to most of other proposed algorithms for solving the job shop

Table 2
Computational Results of FT and LA test instances.

Instance	Size ($n \times m$)	BKS	MPSO in this paper		Ge et al. (2008)		Ge et al. (2007)	Goncalves et al. (2005)
					HIA	HIA	HEA	HGA-Param
			Best	RD	Best	RD	Best	Best
FT06	6×6	55	55	0.00	55	0.00	55	55
FT10	10×10	930	930	0.00	930	0.00	930	930
FT20	20×5	1165	1165	0.00	1165	0.00	1169	1165
LA01	10×5	666	666	0.00	666	0.00	666	666
LA02	10×5	655	655	0.00	655	0.00	655	655
LA03	10×5	597	597	0.00	597	0.00	597	597
LA04	10×5	590	590	0.00	590	0.00	590	590
LA05	10×5	593	593	0.00	593	0.00	593	593
LA06	15×5	926	926	0.00	926	0.00	926	926
LA07	15×5	890	890	0.00	890	0.00	890	890
LA08	15×5	863	863	0.00	863	0.00	863	863
LA09	15×5	951	951	0.00	951	0.00	951	951
LA10	15×5	958	958	0.00	958	0.00	958	958
LA11	20×5	1222	1222	0.00	1222	0.00	1222	1222
LA12	20×5	1039	1039	0.00	1039	0.00	1039	1039
LA13	20×5	1150	1150	0.00	1150	0.00	1150	1150
LA14	20×5	1292	1292	0.00	1292	0.00	1292	1292
LA15	20×5	1207	1207	0.00	1207	0.00	1207	1207
LA16	10×10	945	945	0.00	945	0.00	945	945
LA17	10×10	784	784	0.00	784	0.00	784	784
LA18	10×10	848	848	0.00	848	0.00	848	848
LA19	10×10	842	842	0.00	842	0.00	–	842
LA20	10×10	902	902	0.00	902	0.00	–	907
LA21	15×10	1046	1046	0.00	1046	0.00	1046	1046
LA22	15×10	927	932	0.54	932	0.54	935	935
LA23	15×10	1032	1032	0.00	1032	0.00	1032	1032
LA24	15×10	935	**941**	**0.64**	950	1.60	–	953
LA25	15×10	977	**977**	**0.00**	979	0.20	–	986
LA26	20×10	1218	1218	0.00	1218	0.00	1218	1218
LA27	20×10	1235	**1239**	**0.32**	1256	1.70	1272	1256
LA28	20×10	1216	**1216**	**0.00**	1227	0.90	1227	1232
LA29	20×10	1152	**1173**	**1.82**	1184	2.78	1192	1196
LA30	20×10	1355	1355	0.00	1355	0.00	1355	1355
LA31	30×10	1784	1784	0.00	1784	0.00	1784	1784
LA32	30×10	1850	1850	0.00	1850	0.00	1850	1850
LA33	30×10	1719	1719	0.00	1719	0.00	1719	1719
LA34	30×10	1721	1721	0.00	1721	0.00	1721	1721
LA35	30×10	1888	1888	0.00	1888	0.00	1888	1888
LA36	15×15	1268	**1278**	**0.79**	1281	1.03	1287	1279
LA37	15×15	1397	1411	1.00	1415	1.72	1415	**1408**
LA38	15×15	1196	**1208**	**1.00**	1213	1.42	1213	1219
LA39	15×15	1233	**1233**	**0.00**	1246	1.05	1245	1246
LA40	15×15	1222	**1225**	**0.25**	1240	1.47	1242	1241

Table 3
Computational comparison between PSO and MPSO.

Instance	Size	BKS	PSO			MPSO		
			Best	Max	Avg	Best	Max	Avg
FT06	6×6	55	55	59	56.1	55	55	55.0
FT10	10×10	930	985	1084	1035.6	930	937	930.7
FT20	20×5	1165	1208	1352	1266.9	1165	1169	1165.4
LA01	10×5	666	666	688	668.6	666	666	666.0
LA06	15×5	926	926	926	926.0	926	926	926.0
LA11	20×5	1222	1222	1222	1222.0	1222	1222	1222.0
LA16	10×10	945	956	1035	986.9	945	946	945.7
LA21	15×10	1046	1102	1147	1128.4	1046	1058	1051.3
LA26	20×10	1218	1263	1351	1312.6	1218	1218	1218.0
LA31	30×10	1784	1789	1897	1830.4	1784	1784	1784.0
LA36	15×15	1268	1373	1436	1409.2	1278	1293	1287.5

scheduling problems and is easy to escape the limit of each type of combinatorial optimization problems.

Many pre-proposed evolutionary algorithms for solving JSSP need a heuristic algorithm to initialize a population in order to speed the convergence rate. It has a drawback of increasing the computational load. It need not use any one heuristic algorithm in the proposed MPSO algorithm and can still achieve a better solution. MPSO is tested and approved with 43 instances that are a standard benchmark taken from the OR-Library. According to the experimental results, MPSO can reach the optimal area in the search space with smaller population size and iterations than other existing algorithms achieved. Of course, for another important achievement, combining a multi-type individual enhancement scheme into the PSO can achieve a superior result. For other variant job scheduling, we believe MPSO can easily be applied to solve these problems in the future research.

References

Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society, 14*, 1069–1072.

Fisher, H., & Thompson, G. L. (1963). *Probabilistic learning combinations of local job shop scheduling rules* (pp. 225–251). Prentice-Hall, NJ: Englewood Cliffs.

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

Ge, H., Du, W., & Qian, F. (2007). A hybrid algorithm based on particle swarm optimization and simulated annealing for job shop scheduling. In *Proceedings of the third international conference on natural computation* (Vol. 3, pp. 715–719).

Ge, H. W., Sun, L., Liang, Y. C., & Qian, F. (2008). An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling. *IEEE Transactions on Systems, Man and Cybernetics, Part A, 38*, 358–368.

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

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

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1984). Optimization by simulated annealing. *Science, 220*, 671–680.

Kuo, I. H., Horng, S. J., Kao, T. W., Lin, T. L., Lee, C. L., Terano, T., et al. (2009). An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. *Expert Systems with Applications, 36*, 7027–7032.

Lawer, E. L., Lenstra, J. K., Rinooy Kan, A. H. G., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. Rinnooy Kan, & P. H. Zipkin (Eds.), *Logistics of production and inventory*. *Handbooks in operations research and management science* (Vol. 4). Amsterdam: Elsevier.

Lawrence, S. (1984). *An experimental investigation of heuristic scheduling techniques. In Supplement to resource constrained project scheduling*, GSIA. Pittsburgh, PA: Carnegie Mellon University.

Lian, Z., Gu, X., & Jiao, B. (2006). A dual similar particle swarm optimization algorithm for job-shop scheduling with penalty. In *Proceedings of the sixth world congress on intelligent control and automation* (Vol. 2, pp. 7312–7316).

Liao, C. J., Tseng, C. T., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers and Operations Research, 34*, 3099–3111.

Nowicki, E., & Smutnicki, C. (2005). An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling, 8*, 145–159.

Park, B. J., Choi, H. R., & Kim, H. S. (2003). A hybrid genetic algorithm for the job shop scheduling problems. *Computers and Industrial Engineering, 45*, 597–613.

Ponnambalam, S. G., Aravindan, P., & Rajesh, S. V. (2000). A tabu search algorithm for job shop scheduling. *The International Journal of Advanced Manufacturing Technology, 16*, 765–771.

Steinhöel, K., Albrecht, A., & Wong, C. K. (1999). Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research, 118*, 524–548.

Suresh, R. K., & Mohanasundaram, K. M. (2005). Pareto archived simulated annealing for job shop scheduling with multiple objectives. *The International Journal of Advanced Manufacturing Technology, 29*, 184–196.

Udomsakdigool, A., & Kachitvichyanukul, V. (2008). Multiple colony ant algorithm for job-shop scheduling problem. *International Journal of Production Research, 46*, 4155–4175.

Van Laarhoven, P. J. M., Aarts, E. H. L., & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operaons Research, 40*(1), 113–125.

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

Watanabe, M., Ida, K., & Gen, M. (2005). A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem. *Computers and Industrial Engineering, 48*, 743–752.

Xia, W., & Wu, Z. (2005). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 29*(3–4), 360–366.

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.

Zhou, P., Li, X. P., & Zhang, H. F. (2004). An ant colony algorithm for job shop scheduling problem. In *Proceedings of the fifth world congress on intelligent control and automation* (Vol. 4, pp. 2899–2903).