Int J Adv Manuf Technol (2006) 29: 360–366
DOI 10.1007/s00170-005-2513-4

ORIGINAL ARTICLE

Wei-jun Xia · Zhi-ming Wu

A hybrid particle swarm optimization approach for the job-shop scheduling problem

Received: 28 April 2004 / Accepted: 3 December 2004 / Published online: 23 November 2005

Abstract A new approximation algorithm is proposed for the problem of finding the minimum makespan in the job-shop scheduling environment. The new algorithm is based on the principle of particle swarm optimization (PSO). PSO combines local search (by self-experience) and global search (by neighboring experience), and possesses high search efficiency. Simulated annealing (SA) employs certain probability to avoid becoming trapped in a local optimum and the search process can be controlled by the cooling schedule. By reasonably combining these two different search algorithms, we develop a general, fast and easily implemented hybrid optimization algorithm; we called the HPSO. The effectiveness and efficiency of the proposed PSO-based algorithm are demonstrated by applying it to some benchmark job-shop scheduling problems. Comparison with other results in the literature indicates that the PSO-based algorithm is a viable and effective approach for the job-shop scheduling problem.

Keywords Hybrid optimization · Job-shop scheduling · Particle swarm optimization · Simulated annealing

1 Introduction

Scheduling is concerned with allocating limited resources to tasks to optimize certain objective functions. In the last four decades, many papers have been published in the scheduling area. One of the most popular models in scheduling is that of the job-shop. The classic job-shop scheduling problem (JSP) can be described as follows: a set of m machines and a set of n jobs are given. Each job consists of a sequence of operations that must be executed in a specified order. Each operation has to be performed on a given machine for a given time. A schedule is an allocation of operations on machines in time, i.e. a sequence of operations on machines. The problem is to find the schedule that the makespan (the maximum of job completion times) or other cost function is minimal, subject to the following constraints: (i) the operation precedence constraints are respected for every job; (ii) each machine can process at most one operation at a time; and (iii) an operation cannot be interrupted if it initiates processing on a given machine.

It is well-known that the JSP is NP-hard [1] and belongs to the most intractable problems considered. The minimum makespan problem of job-shop scheduling is a classical combinatorial optimization problem that has received considerable attention in the literature. Historically, JSP was treated via classical math optimization and approximation methods. The math optimization methods are based chiefly on the branch and bound (BB) method. BB algorithms use a dynamically-constructed tree structure as a means of representing the solution space of all feasible sequences. The principle of BB is the enumeration of all feasible solutions of JSP. Early work was performed by Brooks and White [2], followed by Greenberg [3]. Further research included Balas [4], Florian et al. [5], Lageweg et al. [6], and Carlier and Pinson [7]. In the last decade, job-shop researchers considering the math optimization methods include Applegate and Cook [8], Brucker et al. [9], Perregaard and Clausen [10], and Martin [11]. Although the computational study indicates that improvements have been achieved by BB methods, these methods cannot be applied to large problems and their execution necessitates a very good understanding of the JSP. As such, the algorithm lost its attraction to practitioners. Many researchers have turned their attention to approximation methods.

Although approximation methods do not guarantee achieving optimal solutions, they are able to attain near-optimal solutions, even for difficult-to-solve problems, with moderate computing times. Therefore, approximation methods are more suitable for larger problems.

Approximation procedures applied to JSP were first developed on the basis of priority dispatching rules (PDRs). PDRs are probably the most frequently applied heuristics for solving JSP because of their ease of implementation and their substan-

W.-J. Xia (✉) · Z.-M. Wu
Department of Automation,
Shanghai Jiao Tong University,
Shanghai 200030, P.R. China
E-mail: xwjemail@yahoo.com
Tel.: +86-21-62933428

tially reduced computational requirement. But because they only consider the current state of the machine and its immediate surroundings, in general the solution quality is very poor and degrades as problem size increases.

Adams, Balas and Zawack [12] developed a shifting bottleneck procedure (SB) that employs an ingenious combination of scheduling construction and iterative improvement, and is guided by solutions to one-machine scheduling problems. SB procedure is characterized by the following tasks: subproblem identification, bottleneck selection, subproblem solution, and schedule re-optimization. The main contribution of this approach is the way one-machine relaxation is used to decide the order in which machines should be scheduled. A general weakness of the SB approaches is a certain level of programmer technique is required.

With the advent of fast and inexpensive computing power, researchers have begun to experiment with neighborhood search techniques during the last decade. Neighborhood search techniques are based on the concept of local improvement and are easier to implement than the classical operations research techniques. In the seventies, researchers started to apply random swaps search techniques in scheduling problems. The subsequent research in neighborhood search has focused mainly on three techniques: simulated annealing (SA), taboo search (TS), and genetic algorithms (GA). Their general idea is to modify current solutions in a certain sense, where the modifications are defined by a neighborhood operator, such that new feasible solutions are generated, called neighbors, which hopefully have an improved or at most limited the deterioration of their objective function values [13]. These algorithms are all based on a certain neighborhood structure and some rules defining how to obtain a new solution from existing ones. The first efforts to implement powerful general algorithms such as SA (Van Laarhoven et al. [14]), parallel TS (Taillard [15]), and GA (Yamada and Nakano [16], Aarts et al [17]) finally culminated in the excellent TS implementation of Dell'Amico and Trubian [18], Nowicki and Smutnicki [19] and Balas and Vazacopoulos [20]. In recent years, hybrid algorithm-based scheduling became very popular. Kolonko [21] presented a hybrid algorithm (SAGen) where SA was embedded into a GA framework. Pezzella and Merelli [22] described a hybrid optimization strategy TS-SB for JSP, and Aiex et al [23] proposed a parallel greedy randomized adaptive search procedure (GRASP).

In this paper, we introduce a very fast and easily implemented hybrid algorithm based on particle swarm optimization (PSO) and simulated annealing algorithm. Computational experiments demonstrate that the hybrid algorithm performs much better than many known hybrid algorithms.

The remainder of the paper is organized as follows: Sect. 2 describes the general PSO algorithm and how it is applied in JSP. Sect. 3 focuses on basic ingredients of SA for the JSP, and describes some rules of parameters selection in SA. The hybrid optimization algorithm is described in detail in Sect. 4. In Sect. 5, computational experiments are performed with the new optimization algorithm for some benchmark job-shop scheduling problems, results are reported and a comparison is made to other heuristic methods. Some concluding remarks are made in Sect. 6.

2 PSO algorithm

PSO is an evolutionary computation technique proposed by Kennedy and Eberhart [24, 25]. The particle swarm concept was based on the premise of social behavior. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. A PSO algorithm mimics the behavior of flying birds and their means of information exchange to solve optimization problems. PSO has been introduced as an optimization technique in real-number spaces. But many optimization problems are set in a space featuring discrete components. Typical examples include problems that require ordering and route planning, such as in scheduling and routing problems. In this paper, we introduce a method of converting the continuous domain to the discrete domain for PSO. PSO requires only primitive and simple mathematical operators, and is computationally inexpensive in terms of both memory requirements and time.

2.1 Standard PSO algorithm

PSO is similar to the evolutionary algorithm in that the system is initialized with a population ("swarm") of random solutions. Each individual or potential solution, called a particle, flies in the D-dimensional problem space with a velocity that is dynamically adjusted according to the flying experience of the individual and its colleagues. In past years, researchers have explored several models of PSO algorithm. In this paper, we use the global model equations, which are described as follows [26]:

$$\begin{aligned} V_{id} =& W \bullet V_{id} + C_1 \bullet \text{Rand}(\quad) \bullet (P_{id} - X_{id}) \\ &+ C_2 \bullet \text{rand}(\quad) \bullet (P_{gd} - X_{id}) \end{aligned} \tag{1a}$$

$$X_{id} = X_{id} + V_{id} \tag{1b}$$

where V_{id}, the velocity for particle i, represents the distance to be traveled by particle i from its current position, X_{id} represents the particle position, P_{id} – also called "pbest", the local best solution – represents ith particle's best previous position, and P_{gd} – also called qutgbest, the global best solution – represents the best position among all particles in the swarm. W is inertia weight. It regulates the trade-off between the global exploration and local exploitation abilities of the swarm. The acceleration constants C_1 and C_2 represent the weight of the stochastic acceleration terms that pull each particle toward "pbest" and "gbest" positions. Rand( ) and rand( ) are two random functions with range [0,1].

For Eq. 1a, the first part represents the inertia of previous velocity; the second part is the "cognition" part, which represents individuals thinking independently; and the third part is the "social" part, which represents cooperation among the particles [27]. The process of implementing the PSO algorithm is as follows:

1. Initialize a swarm of particles with random positions and velocities in the D-dimensional problem space.
2. For each particle, evaluate the desired optimization fitness function.

3. Individual particles' fitness value are with its pbest. If the current value is better than pbest, then set the pbest value to be equal to the current value, and the pbest position equal to the current position in D-dimensional space.
4. Compare the fitness evaluation value with the swarm's best fitness value obtained so far. If current value is better than gbest, then reset gbest to the current particle's fitness value.
5. Change the velocity and position of the particle according to Eqs. 1a and 1b, respectively.
6. Loop to step (2) until a termination criterion is met, usually a sufficiently good fitness or a specified number of generations.

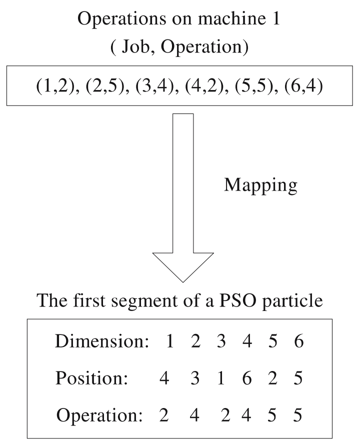
2.2 PSO for JSP

2.2.1 Encoding scheme and initial swarm

One of the key issues in applying PSO successfully to JSP is how to encode a schedule to a search solution, i.e. finding a suitable configuration between problem solution and PSO particle. In this paper, we set up a search space of $n \times m$ dimensions for a problem of n jobs on m machines. A particle consists of m segments, and every segment has n different job numbers, representing the processing orders of n jobs on m machines. For example, an easy problem with six jobs and six machines (FT06 [28]) is considered. Figure 1 shows an instance of a configuration from one possible order (on machine 1) to a particle's position coordinates in the PSO domain.

Generally, initial swarm and initial particle velocities are generated randomly. For reducing the iterative time of PSO, we introduce a new method to generate initial swarm values. Notice that most feasible solutions in JSP are arranged according to the increasing order of the operation, and only a few operations are reversed. Then, we arrange job order according to the increasing order of operation on the machine. If one job's operation order is the same as the other, the two jobs' orders are arranged randomly. For example, consider the jobs and operations on machine 1 in Fig. 1. Figure 2 shows two possible expressions of the first segment of an initial particle that can be generated according to the new way. If jobs on every machine are arranged in this manner, the probability that initial particle may be a feasible solution, i.e. a feasible schedule, increases greatly.

Fig. 1. Jobs assignment to PSO particle mapping



Dimension:	1	2	3	4	5	6
Position:	4	3	1	6	2	5
Operation:	2	4	2	4	5	5

2.2.2 Setting parameters

In Eq. 1a, inertia weight (W) is an important parameter for the search-ability of a PSO algorithm. A large inertia weight facilitates searching a new area, while a small inertia weight facilitates fine-searching in the current search area. Suitable selection of the inertia weight provides a balance between global exploration and local exploitation, and, on average, reduces the number of iterations to find an adequate solution. Therefore, by linearly decreasing the inertia weight from a relatively large value to a relatively small value over the course of operation, the PSO tends to have more global search-ability at the beginning of the run, while having more local search-ability near the end of the run. For all computational experiments in this paper, the inertia weight is set to the following equation:

$$W = W_{\max} - \frac{W_{\max} - W_{\min}}{\text{iter}_{\max}} \bullet \text{iter} \tag{2}$$

where $W_{\max}$ is the initial value of weighting coefficient, $W_{\min}$ is the final value of weighting coefficient, $\text{iter}_{\max}$ is the maximum number of iterations or generation, and "iter" is the current iteration or generation number.

In the following computational examples, the inertia weight is set starting with a value 1.2 and linearly decreases to 0.4 according to Eq. 2 over the course of the run.

The acceleration constants C_1 and C_2 in Eq. 1a adjust the amount of "tension" in PSO system. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement towards, or past, target regions [29]. According to experiences of other researchers, we set the acceleration constants C_1 and C_2 each equal to 2.0 for all the following examples.

Through Eqs. 1a and 1b, the absolute value of V_{id} and X_{id} may be great. Thus, the particle may overshoot the problem space. Therefore, V_{id} and X_{id} should be limited to a maximum velocity $V_{\max}$ and maximum position $X_{\max}$, which are two parameters specified by the user. $V_{\max}$ serves as a constraint to control the global exploration ability of a particle swarm. In this paper, the maximum velocity $V_{\max}$ is set to n (number of jobs), i.e. V_{id} is

Fig. 2. Two possible expressions

Initial particle 1 (the first segment)

Dimension:	1	2	3	4	5	6
Position:	4	1	6	3	2	5
Operation:	2	2	4	4	5	5

Initial particle 2 (the first segment)

Dimension:	1	2	3	4	5	6
Position:	1	4	6	3	5	2
Operation:	2	2	4	4	5	5

a value in the range $[-n, n]$. The maximum position $X_{\max}$ is also set to n. Because X_{id} represents job number in JSP, X_{id} must be a positive integer. Thus, X_{id} is an integer value in the range $[1, n]$.

2.2.3 Fitness function

Fitness is used as the performance evaluation of particles in the swarm. Fitness is usually represented with a function $f : S \to R^+$ (where S is the set of candidate schedules, and R^+ is the set of positive real values). Mapping an original objective function value to a fitness value that represents relative superiority of particles is a feature of the evaluation function. In JSP, the objective function is to minimize the maximum of complete-time on all machines, or other cost functions. Therefore, in our algorithm, we use the maximum complete-time among all machines as the fitness function of a candidate. A particle with the lowest fitness will be superior to other particles and should be reserved in the search process.

2.2.4 Modifying solutions

In the encoding scheme, only position coordinates are used in the practical computational process. Position coordinates represents jobs' order on all machines. The computational result of a particle's position coordinate may be a real value such as 3.265. It is meaningless for job number. Therefore, in the algorithm we usually round off real optimum values to the nearest integer numbers. In this way, we convert a continuous optimization problem to a discrete optimization problem.

The computational results of Eq. 1b will generate repetitive code (job number) in every segment, i.e. one job is processed on the same machine repeatedly. This breaches the constraint conditions in JSP. We call the computation results that breach constraints illegal solutions. Illegal solutions can be converted to legal solutions by modification. The process of modifying solutions is as follows:

1. Check a particle on a machine-by-machine basis and record repetitive job numbers on every machine.
2. Check absent job numbers on every machine of a particle.
3. Sort absent job numbers on every machine (of a particle) according to the increment order of their processes.
4. Substitute absent job numbers for repetitive codes on every machine of a particle from low to high dimensions, accordingly.

We get legal solutions by this process, but some solutions may be infeasible. By computing start-time and end-time of each job, the infeasible solutions can be checked out quickly. We give infeasible solutions large fitness values in evaluation process. As such, infeasible solutions cannot be the pbest or gbest in the search process.

3 Simulated annealing

Ever since its introduction by Kirkpatrick, Gelatt and Vecchi [30], the simulated annealing (SA) algorithm has been applied to many combinatorial optimization problems. On the one hand, the algorithm can be considered as a generalization of the well-known iterative improvement approach to combinatorial optimization problems; on the other hand, it can be viewed as an analogue of an algorithm used in statistical physics for computer simulation of the annealing of a solid to the state with minimum energy [14].

The SA approach can be interpreted as an enhanced version of local search or iterative improvement, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. SA randomizes this procedure in a way that allows occasional alterations that worsen the solution in an attempt to increase the probability of leaving a local optimum. The application of SA as a local search algorithm assumes a cost function (fitness function in this paper) calculated for each possible solution, a neighborhood comprising alternative solutions to a given solution and a mechanism for generating possible solutions.

3.1 SA algorithm

Starting from an initial solution, SA generates a new solution S' in the neighborhood of the original solution S. Then, the change of objective function value, $\Delta = f(S') - f(S)$, is calculated. For a minimization problem, if $\Delta < 0$, the transition to the new solution is accepted. If $\Delta \geq 0$, then the transition to the new solution is accepted with probability, usually denoted by the function $\exp(-\Delta/T)$, where T is a control parameter called the temperature. The SA algorithm generally starts from a high temperature, and then the temperature is gradually lowered. At each temperature, a search is carried out for a certain number of iterations, called the epoch length. When the termination condition is satisfied, the algorithm will stop.

For some reasons, we may be dissatisfied at the solution obtained from the SA algorithm in the first run. We can improve the solution by using the SA algorithm a few times. It is helpful to find a better solution, especially for combinatorial optimization problems.

3.2 Neighborhood solutions

In the SA search algorithm, the choice of neighborhood can greatly influence algorithm performance. While choosing a rich neighborhood containing a large number of candidate solutions will increase the likelihood of finding good solutions, the computation time required to search from the available neighbors will also increase. As a simple method for generating neighborhood solutions, the pair-exchange method is used on each machine of a particle in this paper as follows:

$$(1 \leftrightarrow 2), (2 \leftrightarrow 3), (3 \leftrightarrow 4), \ldots (n-1 \leftrightarrow n)$$

After exchanging jobs pairwise on the same machine of a particle every time, the obtained new solution is evaluated with an objective function. The new solution is accepted if the objective function value is improved. Otherwise, the new solution is accepted with probability, represented by $\exp(-\Delta/T)$. By this process, we can get satisfactory search results in a short time. The efficiency of this method can be proved by the computational results in Sect. 5.

3.3 Cooling schedule

The SA process can be controlled by the cooling schedule. In general, the cooling schedule is specified by several parameters and/or methods, namely the initial temperature T_0, the epoch length L, the rule designated how to lower the temperature, and the termination condition.

A proper initial temperature should be high enough such that all possible solutions have an equal chance of being visited; at the same time, it should not be excessively high such that a lot of unnecessary searches in high temperature will be avoided. It produces a great effect on computation time. Initial temperature T_0 in this paper is set by $T_0 = \Delta f_{\max}$, where $\Delta f_{\max}$ is the maximal difference in fitness value between any two neighboring solutions. It should be adjusted experimentally.

The epoch length L denotes the number of moves made at the same temperature. According to the method of generating neighborhood solutions, the epoch length can be set as S_N, where S_N is the number of neighborhood solutions for a given solution. In our algorithm, S_N is set to be the number of $(n-1) \times m$.

In the SA algorithm, the temperature should be lowered in such a way that the cooling process will not take too long. The method, which is often believed to be excellent in the current literature because it provides a good compromise between a computationally fast schedule and the ability to reach low-energy state, specifies the temperature with $T_k = B * T_{k-1}$ during the kth epoch ($k = 1, 2, 3, \dots$), where B is a parameter called decreasing rate and has a value less than 1. A higher decreasing rate corresponds to a slower process, and therefore more moves are required before the process is terminated. We set the B as value 0.97 or 0.98 by experiments. For the second and third repetitions of SA, we set the values of B as 0.995 and 0.997.

As a criterion to terminate the algorithm, we use a simple and general approach in which a termination temperature T_{end} is set. If current temperature $T_k < T_{\text{end}}$, then the algorithm will be terminated. A T_{end} with a value near zero directly influences the search "granularity" of the algorithm. A smaller T_{end} value implies a finer search in the problem space when algorithm termination is forthcoming. In our algorithm, we set $T_{\text{end}} = 0.1$ when the SA algorithm is used for the first time, and $T_{\text{end}} = 0.01$ in the second or third iteration of the SA.

```
Begin
  Step 1.  Initialization
    1)  PSO
    *  Initialize swarm, including swarm size, each particle's position and velocity;
    *  Evaluate each particle's fitness;
    *  Initialize gbest position with the particle with the lowest fitness in the swarm;
    *  Initialize pbest position with a copy of particle itself;
    *  Give initial value: $W_{max}$, $W_{min}$, $C_1$, $C_2$, and generation = 0.
    2)  SA
    *  Determine $T_0$, $T_{end}$, $B$.
  Step 2.  Computation
    1)  PSO
    While (the maximum of generation is not met)
      Do {
          generation ++;
          Generate next swarm by equation (1a) and (1b);
          Evaluate swarm {
                          Find new gbest and pbest;
                          Update gbest of the swarm and pbest of each particle;
                         }
         }
    2)  SA
    For gbest particle S of swarm
     { $T_k = T_0$;
      While ( $T_k > T_{end}$ )
        Do {
             Generate a neighbor solution S' from S by pair-exchange method;
             Compute fitness of S';
             Evaluate S' {
                          $\Delta = f(S') - f(S)$;
                          if ( min [1, $exp(-\Delta/T_k)$] > random[0, 1] )   { Accept S'; }
                          Update the best solution found so far if possible;
                         }
             $T_k=B*T_{k-1}$;
           }
     }
  Step 3.  Output optimization results.
End
                          Fig. 3   Hybrid optimization algorithm HPSO
```

Fig. 3. Hybrid optimization algorithm HPSO

4 Hybrid PSO algorithm

The PSO algorithm is problem-independent, which means little specific knowledge relevant to a given problem is required. All we have to know is the fitness evaluation of each solution. This advantage makes PSO more robust than many other search algorithms. However, PSO, since is a stochastic search algorithm, it is prone to inadequate global search-ability at the end of a run. PSO may fail to find the required optima in cases when the problem to be solved is too complicated and complex. SA employs certain probability to avoid becoming trapped in a local optimum, and the search process can be controlled by the cooling schedule. By designing the neighborhood structure and cooling schedule of SA, we can control the search process and avoid individuals being trapped in local optimum more efficiently. Thus, a hybrid algorithm of PSO and SA, named HPSO, is presented in Fig. 3.

It can be seen that PSO provides initial solution for SA during the hybrid search process. Such a hybrid algorithm can be converted to general PSO by omitting the SA unit, and it can be converted to traditional SA by setting swarm size to one particle. HPSO implements easily and reserves the generality of PSO and SA. Moreover, such HPSO can be applied to many combinatorial optimization problems by simple modification.

5 Computational results

To illustrate the effectiveness and performance of the proposed algorithm in this paper, various kinds of benchmark job-shop instances with different sizes have been selected to compute. All instances tested were downloaded from the OR-Library (http://mscmga.ms.ic.ac.uk). LA01~LA40 in Table 1 are forty instances of eight different sizes cited from Lawrence [31]. In

Table 1. Computational results

Problem	n	m	BKS	(H)PSO	RE(%)	T_{av}	T_{SA}
LA01	10	5	666	666	0.000	2	1
LA02	10	5	655	655	0.244	3	1
LA03	10	5	597	597	0.381	5	1
LA04	10	5	590	590	0.537	3	1
LA05	10	5	593	593*	0.000	2	0
LA06	15	5	926	926*	0.453	5	0
LA07	15	5	890	890	0.000	5	1
LA08	15	5	863	863	0.000	5	1
LA09	15	5	951	951	0.000	5	1
LA10	15	5	958	958*	0.000	1	0
LA11	20	5	1222	1222*	0.968	4	0
LA12	20	5	1039	1039*	1.299	12	0
LA13	20	5	1150	1150*	0.941	4	0
LA14	20	5	1292	1292*	0.000	2	0
LA15	20	5	1207	1207	0.000	11	1
LA16	10	10	945	945	1.284	127	3
LA17	10	10	784	784	0.127	127	3
LA18	10	10	848	848	1.005	127	3
LA19	10	10	842	842	0.772	127	3
LA20	10	10	902	907	1.136	127	3
LA21	15	10	1046	1047	0.669	387	3
LA22	15	10	927	927	1.121	863	1
LA23	15	10	1032	1032	0.000	92	1
LA24	15	10	935	938	1.569	766	1
LA25	15	10	977	977	1.842	95	2
LA26	20	10	1218	1218	0.640	89	1
LA27	20	10	1235	1236	1.187	1415	1
LA28	20	10	1216	1216	1.225	476	1
LA29	20	10	1157	1164	1.642	1442	1
LA30	20	10	1355	1355	0.000	94	1
LA31	30	10	1784	1784	0.000	56	1
LA32	30	10	1850	1850	0.172	56	1
LA33	30	10	1719	1719	0.000	62	1
LA34	30	10	1721	1721	0.000	73	1
LA35	30	10	1888	1888	0.000	100	1
LA36	15	15	1268	1269	1.341	2473	3
LA37	15	15	1397	1401	1.861	2512	3
LA38	15	15	1184	1208	2.872	2586	3
LA39	15	15	1233	1240	1.784	2492	3
LA40	15	15	1222	1226	1.759	2534	3
FT06	6	6	55	55	0.000	1	1
FT10	10	10	930	930	1.008	142	3
FT20	20	5	1165	1178	1.173	21	1
ORB1	10	10	1059	1059	2.913	334	1
ORB2	10	10	888	889	0.664	182	1
ORB3	10	10	1005	1020	2.607	297	1
ORB4	10	10	1005	1006	0.719	141	3
ORB5	10	10	887	887	1.527	404	1
ORB6	10	10	1010	1010	1.594	337	1
ORB7	10	10	397	397	1.272	366	1
ORB8	10	10	899	899	1.402	340	1
ORB9	10	10	934	934	1.065	365	1
ORB10	10	10	944	944	1.181	351	1
ABZ5	10	10	1234	1234	0.741	330	1
ABZ6	10	10	943	943	0.620	158	1
ABZ7	20	15	656	666	2.652	4332	1
ABZ8	20	15	(645,669)	681	6.054	4356	1
ABZ9	20	15	(656,707)	694	6.364	4374	1

[a]n: Number of jobs
m: Number of machines
BKS: Best known solution so far
(H)PSO: The best objective value of only PSO (represented by adding *) or HPSO algorithm found over 20 runs
RE(%): Percentage of average objective value of algorithm over BKS
T_{av}: Average CPU time (s) on Celeron 300 with 128M RAM
T_{SA}: Times of simulated annealing

Table 1, FT06, FT10 and FT20 are three problem instances cited from [28], which are occasionally called MT06, MT10 and MT20. We selected ten instances of size $n \times m$ = 10×10 (denoted ORB1∼ORB10) from Applegate and Cook [8], and five instances (ABZ5∼ABZ9) from Adams et al. [12]. The optimal solutions of the ABZ8 and ABZ9 instances are still unknown.

The algorithms for JSP mentioned above can be easily implemented on a personal computer. We program the algorithms in Borland C++ and run it on an Intel Celeron 300 with 128M RAM. Swarm size is set to 20, and the maximum number of iterative generations is set to 300 when dimensions are less than 100. Swarm size is set to 30, and maximal generation is set to 500 for other instances. Each instance is randomly run 20 times for each algorithm.

Table 1 shows the computational results of different benchmark instances. The best known solution (BKS) or near-BKS are found for many problem instances in a reasonable amount of computing time. Among the 58 instances, PSO/HPSO finds the BKS in 41 cases (71%), and the difference between computation result and BKS is one in five instances (LA21, LA27, LA36, ORB2, ORB4) among 17 near BKS instances. It is within 1% of the percent average objective value over BKS in 30 instances (52%). In 52 cases (90%), the PSO/HPSO solution is within 2% of the percent average objective value over BKS. Moreover, LA05, LA06 and LA10∼LA14 instances can reach the BKS using only general PSO, which demonstrates the powerful explore-ability of the PSO algorithm. More importantly, the PSO/HPSO algorithm is efficient in running time. From LA01 to LA15 (except LA12 and LA15), almost each instance can reach the BKS in less than 10 seconds of CPU running time. This was unimaginable for researchers in the past. For instances LA31∼LA35 with the same size $n \times m = 30 \times 10$, HPSO can reach the BKS rapidly – within 100 seconds for all instances.

Table 2 shows the comparison of HPSO with well-known algorithms from the literature on the ten tough problems. The column labelled BB refers to the Applegate and Cook [8] algorithm, the next column (GA3) is Mattfeld's [32] algorithm, and the next three columns are Kolonko's [21] SAGen, Pezzella and Merelli's [22] TS-SB method, and Aiex et al.'s [23] GRASP. For the selected problems, GA3, SAGen, TS-SB, HPSO are almost equal in their solution quality. But HPSO has its own advantages: it offers an easily understandable model, simplicity and ease of implementation, as well as a robustness to problem changes.

6 Conclusions

We have discussed a new approach to job-shop scheduling problems based on PSO. The performance of the HPSO algorithm is evaluated in comparison with results obtained from other authors' algorithms for a number of benchmark instances. The new algorithm is very effective and efficient. It can find optima for most test instances, and running time is less than almost all other algorithms. Because of the generality of HPSO, it can be applied to many optimization problems. These results indicate that the proposed algorithm is an attractive alternative for solving the

Table 2. Comparison with other algorithms for the ten tough problems

Problem	*n*	*m*	BKS best	HPSO best	BB best	GA3 best	SAGen best	TS-SB best	GRASP
ABZ7	20	15	656	666	668	668	658	666	692
ABZ8	20	15	(645,669)	681	687	684	670	678	705
ABZ9	20	15	(656,707)	694	707	702	–	693	740
LA21	15	10	1046	1047	1053	1047	1047	1046	1057
LA24	15	10	935	938	935	938	938	938	954
LA25	15	10	977	977	977	977	977	979	984
LA27	20	10	1235	1236	1269	1236	1236	1235	1269
LA29	20	10	1157	1164	1195	1180	1167	1168	1203
LA38	15	15	1196	1208	1209	1201	1201	1201	1218
LA40	15	15	1222	1226	1222	1228	1226	1233	1244

job-shop scheduling problem and other optimization problems. Because the PSO algorithm was originally proposed for continuous optimization problems, our attempt attempts to extend it to discrete optimization problems. Furthermore, applying PSO to other combinatorial optimization problems is also possible in further research.

Acknowledgement The authors acknowledge the financial support of the National Natural Science Foundation of China (Grant no. 70071017, 60574049).

References

1. Garey EL, Johnson DS, Sethi R (1976) The complexity of flowshop and job-shop scheduling. Math Oper Res 1:117–129
2. Brooks GH, White CR (1965) An algorithm for finding optimal or near-optimal solutions to the production scheduling problem. J Ind Eng 16:34–40
3. Greenberg HH (1968) A branch and bound solution to the general scheduling problem. Int J Oper Res 16:353–361
4. Balas E (1969) Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. Int J Oper Res 17:941–957
5. Florian M, Trepant P, McMahon G (1971) An implicit enumeration algorithm for the machine sequencing problem. Manage Sci 1:B782–B792
6. Lageweg BJ, Lenstra JK, Rinnooy Kan AHG (1977) Job-shop scheduling by implicit enumeration. Manage Sci 24:441–450
7. Carlier J, Pinson E (1989) An algorithm for solving the job shop problem. Manage Sci 35:164–176
8. Applegate D, Cook W (1991) A computational study of the job-shop scheduling problem. ORSA J Comput 3:149–156
9. Brucker P, Jurisch B, Sievers B (1994) A branch and bound algorithm for the job-shop scheduling problem. Discret Appl Math 49: 107–127
10. Perregaard M, Clausen J (1995) Parallel branch-and-bound methods for the job-shop scheduling problem. Working Paper, University of Copenhagen, Denmark
11. Martin PD (1996) A time-oriented approach to computing optimal schedules for the job-shop scheduling problem. Dissertation, Cornell University, New York 14853–3801
12. Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manage Sci 34:391–401
13. Blazewicz J, Domschke W, Pesch E (1996) The job shop scheduling problem: Conventional and new solution techniques. Eu J Oper Res 93:1–33
14. Van Laarhoven PJM, Aarts EHL, Lenstra JK (1992) Job shop scheduling by simulated annealing. Int J Oper Res 40:113–125
15. Taillard E (1994) Parallel taboo search techniques for the job-shop scheduling problem. ORSA J Comput 16:108–117
16. Yamada T, Nakano R (1992) A genetic algorithm applicable to large-scale job-shop problems. In: Manner R, Manderick B (eds) Parallel problem solving from nature II, Elsevier Science, Amsterdam, pp 281–290
17. Aarts EHL, Van Laarhoven PJM, Lenstra JK, Ulder NLJ (1994) A computational study of local search algorithms for job-shop scheduling. ORSA J Comput 6:118–125
18. Dell'Amico M, Trubian M (1993) Applying taboo search to the job shop scheduling problem. Ann Oper Res 40:231–252
19. Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job-shop problem. Manage Sci 42:797–813
20. Balas E, Vazacopoulos A (1998) Guided local search with shifting bottleneck for job-shop scheduling. Manage Sci 44:262–275
21. Kolonko M (1999) Some new results on simulated annealing applied to the job shop scheduling problem. Eu J Oper Res 113:123–136
22. Pezzella F, Merelli E (2000) A tabu search method guided by shifting bottleneck for the job shop scheduling problem. Eu J Oper Res 120:297–310
23. Aiex RM, Binato S, Resende MGC (2003) Parallel GRASP with path-relinking for job shop scheduling. Parallel Comput 29:393–430
24. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Network, pp 1942–1948
25. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Japan, pp 39–43
26. Shi Y, Eberhart R (1999) Empirical study of particle swarm optimization. In: Proceedings of Congress on Evolutionary Computation, pp 1945–1950
27. Kennedy J (1997) The particle swarm: social adaptation of knowledge. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp 303–308
28. Muth JF, Thompson GL (1963) Industrial scheduling. Prentice-Hall, Englewood Cliffs, NJ
29. Eberhart R, Shi Y (2001) Particle swarm optimization: developments, applications and resources. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp 81–86
30. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
31. Lawrance S (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Dissertation, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pensylvania
32. Mattfeld D (1995) Evolutionary search and the job shop – investigations on genetic algorithms for production scheduling. Springer, Berlin Heidelberg New York