Proceeding of the 2004 American Control Conference
Boston, Massachusetts June 30 - July 2, 2004

A New Hybrid Optimization Algorithm for the Job-shop Scheduling Problem

Xia Weijun, Wu Zhiming, Zhang Wei, and Yang Genke,
Department of Automation, Shanghai Jiao Tong University
Shanghai 200030, P. R. China

***Abstract*—A new hybrid optimization 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 employs a collaborative population-based search, which combines local search (by self experience) and global search (by neighboring experience), possessing high search efficiency. Simulated annealing (SA) employs certain probability to avoid becoming trapped in a local optimum. By reasonably combining these two different search algorithms, we develop a general, fast and easily implemented hybrid optimization algorithm, named HPSO. The effectiveness and efficiency of the new algorithm are demonstrated by comparing results with other algorithms on some benchmark problems. Comparing results indicate that PSO-based algorithm is a viable and effective approach for the job-shop scheduling problem.**

***Keywords*:Particle swarm optimization, Simulated annealing, Hybrid optimization, Job-shop scheduling.**

0 INTRODUCTION

Scheduling is concerned with allocating limited resources to tasks to optimize certain objective functions. One of the most popular models in scheduling area is that of the job-shop. The classic job-shop scheduling problem (JSP) can be described as follows: Given n jobs, each must be processed on m machines. Each job consists of a sequence of operations, which 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 the operations to time intervals on all machines. The problem is to find the schedule that the makespan (the maximum of job complete-time) is minimal subject to the following constraints: (i) the operation precedence is respected for every job, (ii) each machine can process at most one operation at a time and (iii) an operation can not be interrupted if it initiates processing on a given machine.

It is well-known that JSP is NP-hard and belongs to the most intractable problems considered. Historically JSP was treated via exact methods or approximation algorithms. Exact methods are based chiefly on the Branch and Bound (BB) method[1]. Because it is time-consuming and only can solve small problems, such algorithm lost its attraction to practitioners. On the other hand, approximation algorithms, which are a quite good alternative, have been developed largely during the past decade, such as the shifting bottleneck approach (SB)[2], simulated annealing (SA)[3], taboo search (TS)[4], and genetic algorithm (GA)[5]. In recent years, Pezzella et al. (2000) described a hybrid optimization strategy (TS-SB)[6] for JSP, and Aiex et al. (2003) proposed a parallel greedy randomized adaptive search procedure (GRASP)[7] to solve JSP.

In this paper, we introduce a very fast and easily implemented hybrid algorithm based on particle swarm optimization (PSO) and simulated annealing algorithm. The remainder of this paper is organized as follows: Section 1 describes general PSO algorithm and how to apply it in JSP. Section 2 focuses on basic ingredients of SA for the JSP, describing some rules of parameters selection in SA. The hybrid optimization algorithm is described in section 3. In section 4, the new optimization algorithm is used to solve some benchmark job-shop scheduling problems, presenting results and analyzing difference among different algorithms. Some concluding remarks are made in section 5.

1 PSO ALGORITHM

PSO is an evolutionary computation technique developed by Kennedy and Eberhart in 1995[8]. The particle swarm concept was motivated from the simulation of social behavior. The original intent was to simulate the graceful but unpredictable choreography of bird flock. PSO requires only primitive mathematical operators, and is inexpensive in terms of both memory requirements and time.

Manuscript received September 24, 2003. This work was supported by the National Natural Science Foundation of China under Grant 70071017.

Xia Weijun is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200030, P. R. China. Phone: 86-21-62933428-24, 86-21-62932070; E-mail: weijunxia@sjtu.edu.cn.
Wu Zhiming is a professor of Department of Automation, Shanghai Jiao Tong University, Shanghai 200030, P. R. China. E-mail: ziminwu@sjtu.edu.cn.
Zhang Wei is with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200030, P. R. China. E-mail: zhang_wi@sjtu.edu.cn.
Yang Genke is a professor of Department of Automation, Shanghai Jiao Tong University, Shanghai 200030, P. R. China. E-mail: gkyang@sjtu.edu.cn.

1.1 Standard PSO Algorithm

PSO is initialized with a population (named swarm in PSO) of random solutions. Each individual or potential solution, named particle, flies in the D-dimensional problem space with a velocity which is dynamically adjusted according to the flying experiences of its own and its colleagues. During the past years, researchers have explored several models about PSO. In this paper, we use the global model equations as follows[9]:

$$V_{id} = W * V_{id} + C1 * Rand\,(\;) * (P_{id} - X_{id})$$

$$+ C2 * rand\,(\;) * (P_{gd} - X_{id}) \qquad (1a)$$

$$X_{id} = X_{id} + V_{id} \qquad (1b)$$

Where V_{id}, called the velocity for particle *i*, represents the distance to be traveled by this particle from its current position, X_{id} represents the particle position, P_{id}, which is also called *pbest* (local best solution), represents *ith* particle's best previous position, and P_{gd}, which is also called *gbest* (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 *C1* and *C2* 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 equation (1a), the first part represents the inertia of previous velocity. The second part is the "cognition" part, which represents the private thinking by itself. The third part is the "social" part, which represents the cooperation among the particles[10]. The process for 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) Compare particle's fitness value with particle's *pbest*. If current value is better than *pbest*, then set *pbest* value equal to the current value, and the *pbest* position equal to the current position in D-dimensional space.

4) Compare fitness evaluation value with the swarm's overall previous best. If current value is better than *gbest*, then reset *gbest* to the current particle's value.

5) Change the velocity and position of the particle according to equations (1a) and (1b) respectively.

6) Loop to step 2) until termination criterion is met, usually a sufficiently good fitness value or a specified number of generations.

In PSO, each particle of the swarm shares mutual information globally and benefits from the discoveries and previous experiences of all other colleagues during the search process. So the PSO should be effective in solving practical optimization problems.

1.2 PSO for JSP

a) The 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 mapping 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. Each dimension has discrete set of possible values limited to s = $\{P_i\!: 1 \le i \le n\}$. 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 6 jobs and 6 machines (FT06[11]) is considered. Fig. 1 shows an instance of a mapping from one possible assignment (on machine 1) to a particle position coordinates in the PSO domain, the first segment of a particle.

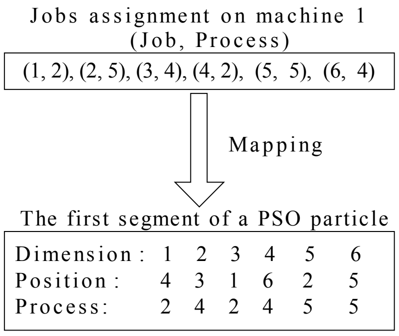

Fig. 1 Jobs assignment to PSO particle mapping

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

b) Setting Parameters

In Equation (1a), inertia weight (*W*) is an important parameter to search ability of PSO algorithm. A large inertia weight facilitates searching 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 results to less iterations on average to find a sufficiently

Initial particle 1 (the first segment)

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

Fig. 2 Two possible expressions on machine 1

optimal solution. Therefore, consider by linearly decreasing the inertia weight from a relatively large value to a relatively small value through the course of PSO run, 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 instances in this paper, the inertia weight is set to the following equation:

$$W = W_{\max} - \frac{W_{\max} - W_{\min}}{iter_{\max}} * iter \qquad (2)$$

Where, W_{max}: Initial value of weighting coefficient,
W_{min}: Final value of weighting coefficient,
$iter_{max}$: Maximum of iteration or generation,
$iter$: Current iteration or generation number.

In following computational instances, the inertia weight is set starting with a value 1.2 and linearly decreasing to 0.4 according to equation (2) through the course of the run.

The acceleration constants *C1* and *C2* in equation (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 toward, or past, target regions [12]. According to experiences of other researchers, let us set the acceleration constants *C1* and *C2* each equal to 2.0 for all following instances.

By computation of equation (1a) and (1b), the absolute value of V_{id} and X_{id} may be great. So the particle may overshoot the problem space. Therefore, V_{id} and X_{id} should be limited to 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. A larger V_{max} facilitates global exploration while a smaller V_{max} encourages local exploitation. In JSP, the maximum velocity V_{max} is set to n (number of jobs), i.e. V_{id} is 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. So X_{id} is an integer value in the range [1, *n*].

c) Fitness Function

Fitness is used as the performance evaluation of particles in the swarm. Fitness is usually represented with a function f: $S \rightarrow R^+$ (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 evaluation function. In JSP, the objective function is to minimize the maximum of complete-time on all machines. Therefore, in our algorithm, we use the maximum complete-time among all machines as the fitness function of a candidate. Particle with the lowest fitness will be superior to other particles and should be reserved in the search process.

d) Modifying Solutions

In encode scheme, only position vector is used to compute in practical computational process. Position vector represents jobs' arrangement on all machines. 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 the real optimum values to its nearest integer number. By this way, we convert a continuous optimization problem to a discrete optimization problem.

The computation results of equation (1b) will generate repetitive code (job number) in every segment, i.e. one job is processed on the same machine repeatedly. It 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 according to machine order 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 increment order of their processes.

4) Substitute absent job numbers for repetitive codes on every machine of a particle from low dimension to high dimension 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. For each infeasible solution, we give it a large fitness value in evaluation process. So infeasible solutions cannot be the *pbest* or *gbest* in search process.

2 SIMULATED ANNEALING

Ever since its introduction, independently by Kirkpatrick, Gelatt and Vecchi[13], simulated annealing 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 minimal energy[3].

SA approach can be viewed 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.

2.1 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. 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 SA algorithm. The solution can be improved by using SA algorithm several times. It is helpful for us to find better solution, especially for complex problems.

2.2 Neighborhood Solutions

In 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 as follows:

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

By exchanging jobs in pair on the same machine of a particle and evaluating pair-exchange result every time, we can get satisfactory search results in a short time.

2.3 Cooling Schedule

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 so that all possible solutions have equal chance of being visited. In this paper, the initial temperature is determined by experiments and experiences. If problem dimensions $(n \times m) \leq 50$, $T_0 = 100$. If problem dimensions $(n \times m) > 50$, $T_0 = 500$. For the second and third time of SA, T_0 is set to the value 10 and 2 respectively.

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

In SA algorithm, the temperature should be lowered in such a way that the cooling process would not take too long. The method, which is often believed to be excellent in the current literature, specifies the temperature with $T_k = B * T_{k-1}$, during the *kth* epoch (k=1, 2, 3, …), where B is a parameter, called the decreasing rate, with a value less than 1. Higher decreasing rate corresponds to slower process, and therefore more moves are required before the process is terminated. We set B as value 0.97 or 0.98 by experiments. For the second and third time 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 way, in which a termination temperature T_{end} is set. If current temperature $T_k < T_{end}$, the algorithm will be terminated. T_{end}, with a value near zero, influences the search "granularity" of algorithm directly. Smaller T_{end} implies finer search in problem space when algorithm termination is forthcoming. In our algorithm, we set T_{end} = 0.1 when SA algorithm is used for the first time and T_{end} = 0.01 in the second or third time of SA.

3 HYBRID PSO ALGORITHM

PSO algorithm is problem-independent, which means little specific knowledge relevant to a given problem is required. What we have to know is just the fitness evaluation for each solution. This advantage makes PSO more robust than many other search algorithms. However, PSO, as a stochastic search algorithm, is prone to lack global search ability at the end of a run. PSO may fail to find the required optima in case 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 cooling schedule. We can control the search process and avoid individuals being trapped in local optimum more efficiently by designing the neighborhood structure and cooling schedule. Thus, a new hybrid algorithm of PSO and SA, named HPSO, is presented in Fig. 3.

```
Begin
  Step 1.  Initialization
   1)    PSO
  *  Initialize swarm size, each particle's position and velocity;
  *  Evaluate each particle's fitness;
  *  Initialize gbest position  with the lowest fitness particle in the swarm;
  *  Initialize pbest position with a copy of particle itself;
  *  Initialize $W_{max}$, $W_{min}$, $C1$, $C2$, maximal generation, 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 swarm and pbest of  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;
        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 The hybrid optimization algorithm HPSO

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

4 COMPUTATIONAL RESULTS

To illustrate the performance of proposed algorithm in this paper, various kinds of benchmark instances with different sizes have been selected to compute. FT06, FT10 and FT20 are three problem instances cited from [11]. LA01~LA40 are forty instances of eight different sizes cited from [14] authored by Lawrence.

The algorithms for JSP mentioned above can be easily implemented on computer. We program the algorithms in Borland C++ and run it on Intel Celeron 300 with 128M RAM. Moreover, swarm size is set to 20 and maximum 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 performed 20 times for each algorithm. Table 1 shows the computational results of different benchmark instances.

Table 1. Computational results by PSO/HPSO for the problem instances of classes FT and LA[a].

Pro.	n	m	BKS	(H)PSO	RE(%)	T_{av}	T_{SA}
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
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	1196	1208	2.872	2586	3
LA39	15	15	1233	1240	1.784	2492	3
LA40	15	15	1222	1226	1.759	2534	3

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

Among the 43 instances, PSO/HPSO finds the BKS in 32 cases (74%). It is within 0.5% of the percentage of average objective value over BKS in 22 instances (51%). In 42 cases (98%), the PSO/HPSO solution is within 2% of the percentage of average objective value over BKS. And for all cases the results are within 3% of the percentage of average objective value over BKS. Moreover, LA05, LA06 and LA10~LA14 instances (*) can be reached BKS only by using general PSO, which demonstrates the powerful explore ability of the PSO algorithm. What is more important, the new algorithm is efficient in running time. From LA01 instance to LA15 instance (except LA12 and LA15), almost each instance can reach BKS less than 10 seconds in CPU running time. It is unimaginable for researchers in the past.

Table 2 shows the comparison of HPSO with well-known algorithms from the literature. The column labeled SB-GA refers to the Dondorf and Pesch algorithm[15], next column GA3 is Mattfeld algorithm[16] and the next three columns are the algorithms SAGen by Kolonko[17], TS-SB method by Pezzella and Merelli[6], and GRASP method by Aiex et al.[7]. For the selected problems set GA3, SAGen, TS-SB, HPSO are almost equal in their solution quality. But HPSO has its own evident advantages: easy understandable model, also the simplicity and the ease of implementation, as well as robustness to problem changes.

Table 2. Comparison with other algorithms

Pro.	BKS	HPSO	SB-GA	GA3	SAGen	TS-SB	GRASP
		Best	Best	Best	Best	Best	Best
FT10	930	930	938	930	----	930	930
LA19	842	842	848	842	842	842	842
LA21	1046	1047	1074	1047	1047	1046	1057
LA22	927	927	936	927	931	927	927
LA24	935	938	957	938	938	938	954
LA25	977	977	1007	977	977	979	984
LA27	1235	1236	1269	1236	1236	1235	1269
LA29	1157	1164	1210	1180	1167	1168	1203
LA38	1196	1208	1241	1201	1201	1201	1218
LA40	1222	1226	1252	1228	1226	1233	1244

5 CONCLUSIONS

We have discussed a new approach to job-shop scheduling problems based on PSO. The performance of HPSO algorithm is evaluated in comparison with the 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 job-shop scheduling problem and other optimization problems. Because PSO algorithm was originally proposed for continuous optimization problems, new attempt has been made by us recently to extend it to discrete optimization problems. Furthermore, applying PSO to other combinatorial optimization problems is also possible in further research.

ACKNOWLEDGMENT

The authors acknowledge the financial support of National Natural Science Foundation of China (No: 70071017).

REFERENCES

[1] Lageweg BJ, Lenstra JK, Rinnooy Kan AHG. Job-shop scheduling by implicit enumeration. Management Science, 1977, 24: 441~450.

[2] Adams J., E. Balas, D. Zawack. The shifting bottleneck procedure for job shop scheduling. Management Science, 1988, 34: 391~401.

[3] Van Laarhoven PJM, Aarts EHL, Lenstra JK. Job shop scheduling by simulated annealing. Operations Research, 1992, 40: 113~125.

[4] Dell'Amico M, Trubian M. Applying tabu search to the job shop scheduling problem. Annals of Operations Research, 1993, 40: 231~252.

[5] T. Yamada, R. Nakano. A genetic algorithm applicable to large-scale job-shop problems. In 2nd PPSN, Proceedings of the 2nd International Workshop on Parallel Problem Solving from Nature, 1992: 281~290.

[6] Pezzella F., Merelli E.. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. European Journal of Operational Research, 2000, 120: 297~310.

[7] Aiex R.M., Binato S., Resende M.G.C.. Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing, 2003, 29: 393~430.

[8] Kennedy J., Eberhart R.. Particle swarm optimization. Proceeding of IEEE international conference on Neural Network, 1995, IV: 1942~1948.

[9] Y. Shi, Eberhart R.. Empirical study of particle swarm optimization. Proceeding of Congress on Evolutionary Computation, 1999: 1945~1950.

[10] Kennedy J.. The particle swarm: social adaptation of knowledge. IEEE International Conference on Evolutionary Computation, 1997: 303~308.

[11] Muth JF, Thompson GL. Industrial scheduling. Englewood Cliffs, NJ: Prentice Hall, 1963.

[12] Eberhart R., Y. Shi. Particle swarm optimization: developments, applications and resources. IEEE International Conference on Evolutionary Computation, 2001: 81~86.

[13] Kirkpatrick S., Gelatt C.D., Vecchi M.P.. Optimization by simulated annealing. Science, 1983, 220, 671~680.

[14] Lawrance S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Graduate school of industrial administration, Carnegie Mellon University: Pittsburgh, 1984.

[15] Dorndorf U., Pesch E.. Evolution based learning in a job shop environment. Computers & Operations Research, 1995, 22: 25~40.

[16] Mattfeld D.. Evolutionary search and the job shop ---- Investigations on genetic algorithms for production scheduling. Springer-Verlag, 1995.

[17] Kolonko M.. Some new results on simulated annealing applied to the job shop scheduling problem. European Journal of Operational Research, 1999, 113: 123~136.

[18] Salman A., Ahamd I., AI-Madani S. Particle swarm optimization for task assignment problem. Microprocessors and Microsystems, 2002, 26: 363~371.